Handbuch

Version 9.0.0

Quality First Software GmbH[1]

20. Februar 2025

[1] https://www.qftest.com

Inhaltsverzeichnis

Vorwort

Wie unser Firmenname unschwer erkennen lässt, hat sich die Quality First Software GmbH dem Thema Qualität in der Softwareentwicklung verschrieben. Unser Beitrag hierzu ist das Programm QF-Test, dessen Handbuch Sie gerade lesen.

Mit Hilfe von QF-Test können Funktionalitätstests von Java-Programmen oder Webseiten automatisiert werden, sofern diese eine grafische Benutzeroberfläche aufweisen. Die undankbare Aufgabe des Testens fällt je nach Größe und Struktur einer Firma und ihrer Projekte mal einem eigenen Team zur Qualitätssicherung zu, mal wird sie von den Entwicklern nebenbei erledigt und manchmal fällt sie ganz unter den Tisch, dann testet eben der Kunde. Anwender von QF-Test sind daher sowohl Entwickler als auch Tester, die zum Teil sehr unterschiedliche Vorkenntnisse in Bezug auf Java oder Web GUIs mitbringen.

Video

Das Video

'Überblick'
https://www.qftest.com/de/yt/ueberblick-42.html

bietet eine allgemeine Übersicht über QF-Test.

Im Video

'Technische Einführung'
https://www.qftest.com/de/yt/technische-einfuehrung-42.html

erhalten Sie einen technischen Einblick in QF-Test.

Wir versuchen in diesem Handbuch sowohl Entwicklern als auch Testern gerecht zu werden und die erforderlichen Grundlagen über den Aufbau einer grafischen Benutzeroberfläche und die Handhabung von Ereignissen unter Java zu vermitteln. Sollten dabei Fragen offen bleiben, wenden Sie sich als Tester am besten an Ihre Entwickler, die Ihnen sicherlich weiterhelfen können. Die Entwickler unter Ihnen werden wir hoffentlich nicht zu sehr langweilen. Sie können die entsprechenden Stellen gegebenenfalls einfach überspringen.

Ursprünglich unterstützte QF-Test nur Java-Swing-GUIs, mit Version 2.0 kam Eclipse/SWT hinzu mit Version 3.0 die Unterstützung für das Testen von Web-Anwendungen. Teile von QF-Test und dieses Handbuch wurden entsprechend geprägt, so dass die Din-

ge aus Sicht des Testens von Swing GUIs erklärt sind. In den meisten Fällen sind die Konzepte universell für alle GUIs anwendbar. Wo sie sich unterscheiden werden mit speziellen Hinweisen die Eigenheiten von SWT oder Web GUIs erklärt.

Und noch ein Wort zur Sprache: Deutsche Bücher zu Computerthemen leiden fast immer unter dem Problem, dass viele Begriffe aus dem Englischen kommen. Diese haben im technischen Zusammenhang oft eine spezielle Bedeutung, die von der normalsprachlichen abweicht. Die Übersetzung solcher Begriffe ist schwierig, führt zu eigenartigen Wortkonstruktionen und verschleiert schlimmstenfalls die eigentliche Bedeutung. Belässt man sie dagegen in ihrer englischen Form, ergibt sich ein etwas holpriger Satzbau. Wir werden in diesem Handbuch beide Wege gehen und von Fall zu Fall entscheiden.

Hinweise zur Benutzung dieses Handbuchs

Dieses Handbuch wird in HTML und PDF Versionen ausgeliefert. Die HTML-Version ist zur besseren Navigation in kleinere Dateien aufgeteilt. Dank vieler Querverweise ist die HTML-Version besser zum Lesen am Rechner geeignet, während die PDF Version hohe Druckqualität liefert.

Die PDF Version des Handbuchs befindet sich unter `qftest-9.0.0/doc/manual_de.pdf`, die Einstiegsseite der HTML-Version unter `qftest-9.0.0/doc/manual/de/manual.html`.

Ein Webbrowser für die HTML-Version kann direkt aus QF-Test heraus gestartet werden. Der Menüeintrag Hilfe→Handbuch... führt Sie zur Einstiegsseite, Hilfe→Neuigkeiten... zur Dokumentation der Änderungen seit der letzten Version. Kontextsensitive Hilfe wird ebenfalls auf diesem Weg geboten. Klicken Sie dazu mit der rechten Maustaste auf einen beliebigen Baumknoten, ein Attribut oder eine Option und wählen Sie Was ist das? im Popupmenü. Das alles funktioniert nur, wenn der Browser des Systems für QF-Test zugänglich ist.

Das Handbuch besteht aus drei Teilen, die aus technischen Gründen in einem Dokument zusammengefasst sind (einfachere Querverweise und Indexgenerierung). Bei diesen Teilen handelt es sich um:

Anwenderhandbuch[(2)]

Dieser Teil erklärt, wie QF-Test installiert und gestartet wird und wie die Benutzeroberfläche zu bedienen ist. Er zeigt, wie Tests erstellt und aufgebaut werden und widmet sich dann fortgeschrittenen Themen. Um doppelte Erklärungen zu vermeiden bezieht sich das Anwenderhandbuch an vielen Stellen auf den Referenzteil, der den jeweiligen Punkt ausführlich erläutert. Es wird dringend empfohlen, diesen Verweisen zu folgen.

Ein ”learning-by-doing” Tutorial ist ebenfalls in HTML und PDF-Versionen verfügbar. Die HTML-Version, die auch direkt mittels Hilfe→Tutorial... aufgerufen werden kann, befindet sich unter `qftest-9.0.0/doc/tutorial/de/tutorial.html`. Die PDF-Version liegt in `qftest-9.0.0/doc/tutorial_de.pdf`.

Das Handbuch folgt folgenden Konventionen:

- Menü→Untermenü stellt ein Menü oder einen Menüeintrag dar.
- Modifier-Taste steht für einen Tastendruck. Mögliche Modifier sind Shift/⇧ (Hochstellen), Strg/^, Alt/⌥, ⌘ oder eine Kombination daraus.
- Der Schrifttyp `Courier` wird für Datei- und Verzeichnisnamen, Programmein- und -ausgaben verwendet.
- Um die Vorzüge von Querverweisen wenigstens ansatzweise zu Papier zu bringen, werden Verweise(iii) in der PDF-Version unterstrichen und geben die Seitenzahl des Ziels klein und in Klammern an.

In der HTML-Version des Handbuchs stehen folgende Tastaturkürzel zur Verfügung:

- K: Zur nächste Seite springen
- J: Zur vorherigen Seite springen
- L: Zu Übersetzung der aktuellen Seite springen
- T: Zum Inhaltsverzeichnis springen
- S: Navigation ein/ausklappen

Während einer lokalen Suche sind außerdem folgende Tastaturkürzel aktiv:

- ↓: Zum nächsten Suchtreffer springen

- ⇧: Zum vorherigen Suchtreffer springen
- Escape: Suche abbrechen

Suchtreffer bei der lokalen Suche enthalten immer alle eingegebenen Begriffe im gleichen Abschnitt. Dabei werden, nach Möglichkeit, auch Wörter mit dem gleichen Wortstamm einbezogen und Teilwörter vervollständigt. Möchte man nach exakten Begriffen suchen, so sind diese in doppelte Anführungszeichen (″ . . . ″) einzuschließen.

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Anwenderhandbuch

Kapitel 1

Installation und Start

Video

Das Video

'Installation & Testlizenz'
https://www.qftest.com/de/yt/installation-testlizenz.html

erläutert zunächst den Download und die Installation von QF-Test, dann (ab Minute 8:00) das Einrichten einer Testlizenz.

Die Installation von QF-Test auf den unterstützten Betriebssystemen wird in den folgenden Abschnitten ausführlich beschrieben. Zum Download stehen folgende Paketvarianten zur Verfügung:

Windows (Abschnitt 1.2[6])
: Normalerweise sollten Sie unter Windows QF-Test über das Setup-Programm `QF-Test-9.0.0.exe` installieren. Dieses benötigt Administratorrechten, um in die gängigen Windows Verzeichnisse schreiben zu können. Für eine lokale Installation ohne besondere Rechte entpacken Sie stattdessen das selbst extrahierende Archiv `QF-Test-9.0.0-sfx.exe`.

Linux (Abschnitt 1.3[9])
: Für Linux entpacken Sie bitte das Archiv `QF-Test-9.0.0.tar.gz`.

macOS (Abschnitt 1.4[10])
: Für die Installation unter macOS dient das Disk-Image `QF-Test-9.0.0.dmg`.

Generell kann man Versionen mit unterschiedlichen Versionsnummern von QF-Test parallel installiert haben. Bei der Installation werden vorhandene Konfigurationsdateien nicht überschrieben.

Im Abschnitt 36.2[459] finden Sie Best Practices für die Installation von QF-Test.

1.1 Systemvoraussetzungen

1.1.1 Hard- und Software

QF-Test selbst startet mit Java 17. Das 64 Bit Java Runtime Environment (JRE) wird dabei mit QF-Test bereitgestellt, so dass Java nicht auf Ihrem System installiert sein muss, sofern dies nicht für das SUT erforderlich ist.

Hinweis Wenn Ihr zu testendes System (SUT) auf Java basiert, sollte es normalerweise sein eigenes JRE nutzen, nicht das von QF-Test. Das Java-Kommando für das SUT wird getrennt beim Erstellen der Startsequenz für das SUT festgelegt. Unterstützte Java-Versionen für das SUT werden im Folgenden aufgelistet.

Eine QF-Test Installation belegt etwa 1 GB auf der Festplatte. Bei der Arbeit mit QF-Test wird etwa ebenso viel an Arbeitsspeicher benötigt, aber das hängt auch von der Größe der Testsuiten und der Länge eines Testlaufs ab, siehe Mein Test läuft über lange Zeit und QF-Test geht der Speicher aus. Wie kann ich das verhindern? A(1368). Denken Sie auch daran, dass die für das SUT benötigten Ressourcen noch hinzu kommen.

1.1.2 Unterstützte Technologien - QF-Test

Die folgende Tabelle fasst die offiziell unterstützten Versionen von Betriebssystemen und erforderlicher Software für diese QF-Test Version 9.0.0 zusammen. Unterstützung für zusätzliche Systeme und Versionen können auf Anfrage verfügbar sein, jedoch ohne Gewähr. Eine weitere Möglichkeit, um Unterstützung für ältere Software zu erlangen, kann die Nutzung einer der älteren QF-Test Versionen sein, die herunterladbar sind über die Seite https://www.qftest.com/de/qf-test/download.html.

Hinweis Die Unterstützung für 32 Bit Software wurde in QF-Test Version 7.0 abgekündigt und in Version 8.0 entfernt. Das Testen von nativen 32 Bit Windows-Anwendungen bleibt aber weiterhin unterstützt.

Technologie	Versionseinschränkungen	Einschränkungen bei SUT-Technologie
Windows	10, 11, Server 2016, Server 2019, Server 2022	kein iOS
Linux		kein Windows und iOS
macOS	macOS 12 und höher	kein Windows und SWT

Tabelle 1.1: Unterstützte Betriebssysteme für QF-Test

1.1.3 Unterstützte Technologien - zu testende Systeme

Die folgenden Tabellen fassen die offiziell unterstützte Software zusammen, auf welcher eine mit QF-Test zu testende Applikation (SUT) basieren kann. Unterstützung für zusätzliche Versionen kann auf Anfrage verfügbar sein, jedoch ohne Gewähr. Eine weitere Möglichkeit, Unterstützung für ältere Software zu erlangen, kann die Nutzung einer der älteren QF-Test Versionen sein, die herunterladbar sind über die Seite https://www.qftest.com/de/qf-test/download.html.

Die zu testende Software kann auf den gleichen Betriebssystemen wie QF-Test ausgeführt werden, hinsichtlich Einschränkungen siehe Unterstützte Betriebssysteme für QF-Test(3)

Hinweis In QF-Test Version 7.0 wurde die Unterstützung für 32 Bit Software abgekündigt und wird in einer der nächsten QF-Test Versionen entfernt.

Technologie	Versionseinschränkungen	Bemerkung
JDK/JRE	8 - 24 für das SUT	
Swing		Alle Plattformen.
JavaFX	8 und höher	Alle Plattformen.
SWT	3.7 - 4.35 bzw. 2025-03	64 Bit, nur Windows und Linux GTK, GTK3 ab SWT 4.6. Für 32 Bit Eclipse/SWT Versionen verwenden Sie bitte QF-Test 7.1 oder älter. Für Eclipse/SWT 3.5 - 3.6 können Sie https://archive.qfs.de/pub/qftest/swt_legacy.zip herunterladen und in das `swt` Verzeichnis Ihrer QF-Test Installation entpacken.

Tabelle 1.2: Unterstützte Java-Versionen

Technologie	Versionseinschränkungen	Bemerkung
Chrome	Version 131 mit QF-Driver, aktuelle Versionen über das Chrome DevTools Protokoll (CDP-Driver) und automatischen ChromeDriver-Download (WebDriver).	Inkl. Headless Chrome. Siehe auch Browser Verbindungsmodus[1128]
Firefox (WebDriver)	Wie vom enthaltenen GeckoDriver unterstützt, aktuell 128esr und höher	Inkl. Headless Firefox. Siehe auch Browser Verbindungsmodus[1128]
Microsoft Edge	Aktuelle Versionen über das Chrome DevTools Protokoll (CDP-Driver) und automatischen MSEdgeDriver-Download (WebDriver).	Inkl. Headless Edge.
Opera	Aktuelle Versionen über das Chrome DevTools Protokoll (CDP-Driver).	
Safari		WebDriver mit Safari[1135]
JxBrowser	Version 6, 7 and 8, eingebettet in Swing, JavaFX oder SWT	
Electron	1.7 und neuer	
Web-Komponentenbibliotheken	Detaillierte Aufstellung der unterstützten Toolkits in Abschnitt 51.2[1122]	

Tabelle 1.3: Unterstützte Web-Browser und Toolkits

Technologie	Versionseinschränkungen	Bemerkung
Native Windows-Applikationen	Es können Anwendungen getestet werden, die die Microsoft UI Automation oder die Microsoft Active Accessibility (MSAA) Schnittstellen unterstützen.	Weitere Informationen siehe Abschnitt 15.1[234]
Android	Android API 24 oder höher. Dies entspricht der Android Version 7 Nougat oder neuer.	Weitere Informationen siehe Voraussetzungen und bekannte Einschränkungen[246]
iOS	iOS 15 oder neuer - Systembedingte Einschränkungen aufgrund der installierten Xcode-Version sind möglich.	iOS-Anwendungen können nur auf einem macOS-System getestet werden, worauf die Entwicklungsumgebung Xcode mindestens in Version 13 installiert sein muss. Weitere Informationen siehe Voraussetzungen und bekannte Einschränkungen[270]
PDF		Allgemeine Informationen siehe Testen von PDF-Dokumenten[286]

Tabelle 1.4: Weitere unterstützte Software

1.2 Windows Installation

Die Installation von QF-Test für Windows kann auf zwei Arten erfolgen:

1.2.1 Installation mit dem Windows Setup-Programm `QF-Test-9.0.0.exe`

Dieses Installationsprogramm erfordert Administratorrechte und folgt den Standardvorgaben von Windows, die eine Trennung von nur lesbaren Programmdateien und beschreibbaren Konfigurationsdateien vorsehen. Besteht bereits eine ältere QF-Test Installation, kann QF-Test inklusive Systemkonfiguration optional auch am Windows-Standard vorbei komplett zur vorhandenen Installation geschrieben werden.

4.2+ **Installation gemäß Windows Richtlinien**
Die Programmdateien werden in `C:\Programme\QFS\QF-Test` bzw. einem frei wählbaren Zielverzeichnis abgelegt. Die Systemkonfiguration mit beschreibbaren Dateien wird unabhängig vom Zielverzeichnis nach `%PROGRAMDATA%\QFS\QF-Test` geschrieben.

Hinweis

Das Verzeichnis `%PROGRAMDATA%` kann je nach Windows Version anders heißen, üblich ist `C:\ProgramData`. Im Windows Explorer ist es standardmäßig ausgeblendet. Ein einfacher Weg, in dieses Verzeichnis zu navigieren, ist die Eingabe von `%PROGRAMDATA%` in der Adresszeile des Windows Explorers. In der PowerShell verwenden Sie `cd $env:PROGRAMDATA`, in einem cmd Konsolenfenster ("Eingabeaufforderung") `cd /d %PROGRAMDATA%`, um in das Verzeichnis und auf das zugehörige Laufwerk zu wechseln.

4.2+

Installation zusammen mit bereits vorhandener QF-Test Version

Falls eine ältere QF-Test Installation gefunden wird und noch keine Systemkonfiguration in `%PROGRAMDATA%\QFS\QF-Test` vorliegt, können Sie wählen, ob Sie die Installation gemäß den Windows Richtlinien mit `%PROGRAMDATA%` vorziehen oder QF-Test weiterhin in der alten Struktur installieren möchten.

In ersterem Fall wird nach Auswahl des Zielverzeichnisses die Systemkonfiguration von der alten Installation einmalig übernommen und nach `%PROGRAMDATA%\QFS\QF-Test` kopiert.

Wollen Sie die alte Struktur beibehalten, wird QF-Test in dieses Verzeichnis installiert und nutzt die dort vorliegende Systemkonfiguration.

In beiden Fällen wird `%PROGRAMDATA%\QFS\QF-Test\qftestpath` in den Systempfad aufgenommen und die beiden Programme qftest.exe und qftestc.exe dorthin kopiert. Dies ermöglicht den Start von QF-Test von einem beliebigen Ort aus.

Unabhängig von der Wahl der Installation können die alte und neue QF-Test Version parallel betrieben werden. Bei der Installation gemäß Windows Richtlinien mit `%PROGRAMDATA%` sind die alte und neue Systemkonfiguration voneinander unabhängig. Wird die alte Struktur beibehalten, teilen sich alle Versionen die gemeinsame Systemkonfiguration. Perspektivisch raten wir zum Übergang zu `%PROGRAMDATA%`, da die für die alte Struktur notwendige Änderung von Zugriffsrechten im Programmverzeichnis kritisch zu sehen ist. In der Übergangsphase von QF-Test 4.1 zu 4.2 kann es aber einfacher sein, beide Versionen gemeinsam vorzuhalten. Der Wechsel zur Installation gemäß Windows Richtlinien mit `%PROGRAMDATA%` kann auch im Zuge einer späteren Installation erfolgen.

Unbeaufsichtigte (Silent) Installation

Für die automatische Verteilung auf Testsystemen kann es gewünscht sein, dass QF-Test unbeaufsichtigt installiert werden kann. QF-Test unterstützt diese Art der Installation, da das Installationsprogramm von QF-Test Inno Setup verwendet. Damit sind fast alle unter https://jrsoftware.org/ishelp/index.php?topic=setupcmdline beschriebenen Parameter auch beim Installationsprogramm von QF-Test verwendbar.

Eine Standardinstallation kann einfach mittels `QF-Test-9.0.0.exe /VERYSILENT` durchgeführt werden. Um zum Beispiel kein Desktop-Icon anzulegen, können Sie `QF-Test-9.0.0.exe /VERYSILENT /MERGETASKS="!desktopicon"`
ausführen. Damit wird eine Standardinstallation ohne den Task "desktopicon"

ausgeführt.
Auch die beiden `minisetup-admin.exe` und `minisetup-noadmin.exe` lassen sich so unbeaufsichtigt installieren. Zum Beispiel mittels `minisetup-admin.exe /VERYSILENT`.

Beachten Sie bitte, dass die Installation für alle Benutzer erhöhte administrative Rechte benötigt. Um also auch den Windows UAC-Dialog nicht anzuzeigen muss bereits der aufrufende Prozess erhöhte Rechte besitzen. Der Parameter `/CURRENTUSER` ändert daran nichts, da die Installation immer erhöhte Rechte benötigt.
Eine Ausnahme davon ist explizit `minisetup-noadmin.exe`, das die Konfiguration eines bereits installierten QF-Tests für den aktuellen Benutzer ohne administrative Rechte erlaubt.

Eine Alternative für die unbeaufsichtigte Installation ist das portable selbstentpackende Archiv `QF-Test-9.0.0-sfx.exe`.

1.2.2 Auspacken des selbstextrahierenden Archivs `QF-Test-9.0.0-sfx.exe`

Wenn Sie nicht über Administratorrechte verfügen oder alle Dateien von QF-Test zusammen an einer Stelle haben möchten, packen Sie das Archiv `QF-Test-9.0.0-sfx.exe` zunächst an einem geeigneten Ort aus. Hierzu kopieren Sie es an die gewünschte Stelle und führen es dort aus. Falls 7-Zip auf Ihrem System installiert ist können sie das Archiv alternativ per Rechtsklick mit 7-Zip öffnen und dann damit extrahieren. Am Zielort wird das Verzeichnis `qftest` erstellt, das im Weiteren als Wurzelverzeichnis von QF-Test bezeichnet wird und gleichzeitig die Systemkonfiguration beinhaltet.

Anschließend können Sie das Programm `minisetup-noadmin.exe` im Unterverzeichnis `qftest-9.0.0` ausführen, um Verknüpfungen für die zu QF-Test gehörenden Dateiendungen und einen Eintrag im Windows Startmenü sowie optional ein Desktop Icon zu erstellen. Falls Sie über Administratorrechte verfügen, können Sie stattdessen `minisetup-admin.exe` ausführen, um diese Einstellungen für alle Anwender vorzunehmen und zusätzlich das Verzeichnis `%PROGRAMDATA%\QFS\QF-Test\qftestpath` in den Systempfad aufzunehmen und die beiden Programme `qftest.exe` und `qftestc.exe` dorthin zu kopieren.

Wenn Sie stattdessen eine vollständig portable Installation bevorzugen, können Sie im Verzeichnis `qftest` einen neuen Ordner namens `userdir` anlegen. Dieser wird dann anstelle von `%APPDATA%\QFS\QF-Test` als benutzerspezifisches Konfigurationsverzeichnis verwendet, so dass wirklich alle zu QF-Test gehörenden Dateien an einem Ort zusammen liegen und keine Änderungen am System vorgenommen werden.

1.2.3 Abschluss der Installation und Java-Konfiguration

Zum Abschluss der Installation bietet jedes der Setup-Programme an, den für QF-Test verwendeten Speicher sowie die Anzeigesprache zu konfigurieren. Dies geschieht mit Hilfe eines kleinen Dialogs, in dem Sie diese Einstellung vornehmen können. Bei einer portablen Installation kann der Dialog durch Ausführen von `qftest\qftest-9.0.0\bin\qfconfig.exe` aufgerufen werden.

QF-Test wird mit einem 64 Bit Java 17 Runtime Environment ausgeliefert. Dieses befindet sich im Installationsverzeichnis von QF-Test. Es wird empfohlen, dieses zu verwenden.

Der Dialog enthält eine Einstellmöglichkeit des maximal für QF-Test zur Verfügung stehenden Speichers. Als Vorgabe werden 1024 MB verwendet.

Weiterhin können Sie die Sprache für QF-Test festlegen. Normalerweise richtet sich diese nach den Systemeinstellungen, Sie können aber auch gezielt die deutsche oder die englische Version auswählen.

Diese Einstellungen werden in der Datei `launcherwin.cfg` in QF-Test's Verzeichnis für die Systemkonfiguration gespeichert und von dort durch das `qftest.exe` Startprogramm gelesen. Sie können das Konfigurationsprogramm jederzeit aus dem System Menü heraus aufrufen, um diese Einstellungen zu ändern.

1.3 Linux Installation

Wählen Sie zunächst ein geeignetes Verzeichnis aus, das diese und zukünftige Versionen von QF-Test beherbergen wird, z.B. `/opt` oder `/usr/local`. Wechseln Sie in dieses Verzeichnis und stellen Sie sicher, dass Sie Schreibzugriff darauf haben. Wenn Sie auf eine neue Version von QF-Test aufrüsten, verwenden Sie das selbe Verzeichnis wieder.

Packen Sie das `.tar.gz` Archiv mittels `tar xfzv QF-Test-9.0.0.tar.gz` aus. Dabei wird das Verzeichnis `qftest` erstellt, das im Weiteren als Wurzelverzeichnis von QF-Test bezeichnet wird. Unter Linux ist dies gleichzeitig das Systemverzeichnis, das die Systemkonfiguration von QF-Test enthält.

Nach der ersten Installation von QF-Test enthält das Wurzelverzeichnis nur das versionsspezifische Unterverzeichnis `qftest-9.0.0`. Bei einem Update wird ein neues Unterverzeichnis für die aktuelle Version hinzugefügt.

Um die Installation abzuschließen, wechseln Sie mittels `cd qftest/qftest-9.0.0` in das Unterverzeichnis für die aktuelle QF-Test Version und führen dort das Shell-Skript `setup.sh` aus.

Dieses Setup Skript erzeugt die Verzeichnisse `log`, `jython`, `groovy` und

`javascript` im QF-Test Wurzelverzeichnis, sofern diese noch nicht vorhanden sind. Anschließend wird auf Wunsch ein symbolischer Link vom Verzeichnis `/usr/local/bin` (oder `/usr/bin`, falls es kein `/usr/local/bin` gibt) zum Startskript für den `qftest` Befehl erstellt. Sie benötigen Schreiberlaubnis für `/usr/local/bin`, um diesen Link zu erstellen.

Unter Linux sollte QF-Test normalerweise sein mitgebrachtes JRE verwenden. Alternativ kann nun das `java` Programm festgelegt werden, mit dem QF-Test gestartet wird, sofern nicht mittels `-java <Programm>` (abgekündigt)[977] ein anderes Java-Programm angegeben wird. Das `setup` Skript durchsucht hierzu den `PATH` und schlägt das erste gefundene `java` Programm vor. Befindet sich kein `java` Programm im `PATH` oder wollen Sie ein anderes verwenden, können Sie ein `java` Programm angeben. Das Skript ermittelt dann selbstständig die JDK Version.

Als nächstes steht die Angabe des maximal für QF-Test zur Verfügung stehenden Speichers an, vorgegeben sind 1024 MB. Alternativ kann QF-Test mit der Kommandozeilenoption `-J-XmxZZZm` gestartet werden, wobei `ZZZ` den Speicher in MB definiert.

Zuletzt haben Sie die Möglichkeit, die Sprache für QF-Test auszuwählen. Diese richtet sich normalerweise nach den Systemeinstellungen, kann aber auch gezielt auf Deutsch oder Englisch eingestellt werden. Allerdings gilt diese Einstellung dann für alle Anwender. Alternativ können Sie QF-Test mit der Option `-J-Duser.language=XX` aufrufen, mit `XX` gleich `de` für Deutsch oder `en` für Englisch.

Sofern oben genannte Werte vom Standard abweichen, werden sie in die Datei `launcher.cfg` in QF-Tests Wurzelverzeichnis geschrieben. Diese Datei wird vom `qftest` Startskript ausgewertet und auch bei einem Update von QF-Test herangezogen.

1.4 macOS Installation

Zur Installation von QF-Test als macOS-Anwendung aktivieren Sie das Disk-Image `QF-Test-9.0.0.dmg` und kopieren Sie die QF-Test Anwendung in den Ordner `Programme` (oder einen anderen Ordner nach Wahl) und starten Sie QF-Test von dort.

Hinweis

Um eine spezielle Konfiguration wie zum Beispiel den für QF-Test zur Verfügung stehenden Speicher oder die Sprache anzupassen gibt es für QF-Test unter macOS einen speziellen Unterpunkt in den QF-Test Optionen (Allgemein->Programmstart). Hier können die gewünschten Einstellungen gesetzt werden, welche dann beim Neustart von QF-Test gesetzt werden.

1.5 Lizenzdatei

Video

Das Video

'Installation & Testlizenz'
https://www.qftest.com/de/yt/installation-testlizenz.html

erläutert zunächst den Download und die Installation von QF-Test, dann (ab Minute 8:00) das Einrichten einer Testlizenz.

Im Video

'Lizenzupdate'
https://www.qftest.com/de/yt/lizenz-update.html

wird die Vorgehensweise bei einem Lizenzupdate gezeigt.

Zum Start benötigt QF-Test eine Lizenzdatei, die Sie von Quality First Software GmbH erhalten.

4.0+

Seit QF-Test 4.0 ist der empfohlene Weg, Lizenzen direkt über das Menü Hilfe→Lizenz aktivieren... zu aktivieren oder zu aktualisieren.
Alternativ funktioniert auch noch der folgende traditionelle Weg:

Stellen Sie die Lizenzdatei in das Systemverzeichnis von QF-Test. Unter Windows ist dies je nach Installationsvariante `%PROGRAMDATA%\QFS\QF-Test` (vgl. Abschnitt 1.2[6]) oder, wie unter Linux, das Wurzelverzeichnis von QF-Test. Achten Sie darauf, dass die Datei den Namen `license` **ohne Erweiterungen** erhält. Manche Mailprogramme versuchen den Dateityp zu erraten und hängen eigenständig eine Erweiterung an. Wenn Sie auf eine neue QF-Test Version aufrüsten, können Sie die Lizenzdatei einfach beibehalten, sofern diese für die neue Version gültig ist.

Hinweis

Für eine Aufstellung der für QF-Test relevanten Verzeichnisse öffnen Sie den Info-Dialog über das Menü Hilfe→Info und wählen den Reiter "Systeminfo".

Wenn Sie Ihre Lizenz erweitern, z.B. auf eine größere Benutzerzahl oder eine neue Version von QF-Test, erhalten Sie von Quality First Software GmbH dazu eine Datei namens `license.new`. Diese Datei ist normalerweise selbst keine gültige Lizenz, sondern muss mit Ihrer aktuellen Lizenz kombiniert werden. Gehen Sie dazu wie folgt vor:

- Speichern Sie die Datei `license.new` in das selbe Verzeichnis, in dem Ihre aktuelle Lizenz liegt. Stellen Sie sicher, dass dieses Verzeichnis und die alte Lizenzdatei namens `license` von Ihnen beschreibbar sind.
- Starten Sie QF-Test im interaktiven Modus. QF-Test erkennt selbstständig die neue Datei, überprüft sie auf Gültigkeit und bietet Ihnen an, Ihre Lizenz zu aktualisieren.
- Wenn Sie zustimmen, wird die aktuelle Lizenz in `license.old` umbenannt und

die neue, kombinierte Lizenz in `license` geschrieben. Wenn Sie sich vergewissert haben, dass alles in Ordnung ist, können Sie die Dateien `license.old` und `license.new` löschen.

- Sollte QF-Test das Update nicht erkennen, überprüfen Sie, ob der Zeitstempel der Datei `license.new` neuer ist als der von `license`. Achten Sie auch darauf, dass nicht bereits eine Instanz von QF-Test auf Ihrem Rechner läuft.

Wenn Sie einen besonderen Namen oder Pfad für die Lizenzdatei verwenden oder mit mehr als einer Lizenzdatei arbeiten möchten, so lässt sich dies durch Verwendung des Kommandozeilenarguments `-license <Datei>`(983) erreichen, das in Kapitel 44(971) beschrieben wird.

1.6 Konfigurationsdateien

Hinweis

Für eine Aufstellung der für QF-Test relevanten Verzeichnisse öffnen Sie den Info-Dialog über das Menü Hilfe→Info und wählen den Reiter "Systeminfo".

Beim Beenden speichert QF-Test die Fensterpositionen und persönliche Einstellungen in einer Datei namens `config` im benutzerspezifischen Konfigurationsverzeichnis von QF-Test, welches außerdem die Protokolle von Tests enthält, die im interaktiven Modus durchgeführt werden sowie Profil-Verzeichnisse für das Testen von Web-Anwendungen und temporäre Dateien zur Bearbeitung und Ausführung von Skripten.

4.2+

Unter Windows ist das benutzerspezifische Konfigurationsverzeichnis normalerweise `%APPDATA%\QFS\QF-Test` für neue Installationen. Falls dieses Verzeichnis nicht existiert und Sie bereits eine QF-Test Version älter als 4.2 auf diesem System genutzt haben, welches das Verzeichnis `.qftest` in Ihrem Anwenderverzeichnis erstellt hat, nutzt QF-Test dieses Verzeichnis weiter.
Sie können manuell den Inhalt des Verzeichnisses `.qftest` nach `%APPDATA%\QFS\QF-Test` verschieben und anschließend `.qftest` entfernen. QF-Test ab Version 4.2.0 nutzt dann nur noch dieses Verzeichnis. Sie sollten diesen Vorgang aber nur durchführen, wenn Sie keine Version älter als 4.2.0 mehr verwenden möchten!
Unter Linux ist das benutzerspezifische Konfigurationsverzeichnis immer `~/.qftest`.
Unter macOS befindet es sich unter `/Users/<username>/Library/Application Support/de.qfs.apps.qftest`.
Die persönliche Konfigurationsdatei wird nicht gelesen, wenn QF-Test im Batchmodus ausgeführt wird (vgl. Abschnitt 1.7(13)). Unabhängig von den Systemvorgaben können Sie das benutzerspezifische Konfigurationsverzeichnis jederzeit über das Kommandozeilenargument `-userdir <Verzeichnis>`(994) festlegen oder nur die Konfigurationsdatei mittels `-usercfg <Datei>`(994).

Systemeinstellungen, die von mehreren Anwendern gemeinsam genutzt werden, speichert QF-Test in der Datei `qftest.cfg` im systemspezifischen Konfigurationsverzeichnis, welches auch die Lizenzdatei, Module für Skripte, Java-Plugins und andere Dateien beinhaltet. Unter Windows hängt der Ort des Systemverzeichnisses von der Installations-Variante ab (vgl. Abschnitt 1.2(6)). Es befindet sich entweder in `%PROGRAMDATA%\QFS\QF-Test` oder im Wurzelverzeichnis von QF-Test.
Unter Linux und macOS ist das Systemverzeichnis normalerweise identisch zum Wurzelverzeichnis von QF-Test.
Den Ort der System-Konfigurationsdatei können Sie über das Kommandozeilenargument `-systemcfg <Datei>`(992) ändern, das gesamte Systemverzeichnis via `-systemdir <Verzeichnis>`(992).

1.7 Aufruf von QF-Test

Es gibt zwei Modi, in denen QF-Test ausgeführt werden kann. Im normalen Modus ist QF-Test der Editor für Testsuiten und Protokolle und das Kontrollzentrum zum Starten von Programmen, Aufzeichnen von Events und Ausführen von Tests. Wird QF-Test mit dem Argument `-batch`(976) aufgerufen, geht es in den *Batchmodus*. Anstatt ein Editor Fenster zu öffnen, lädt es die auf der Kommandozeile angegebenen Testsuiten und führt sie automatisch aus, ohne dass es weiterer Eingriffe bedarf. Das Ergebnis des Testlaufs wird durch den Rückgabewert(996) von QF-Test sowie den optionalen Protokollen (siehe Abschnitt 7.1(138)) und Reports (siehe Abschnitt 7.1(138)) ausgedrückt.

Das Setup Skript für Linux erstellt auf Wunsch einen symbolischen Link von `/usr/local/bin` zum `qftest` Startskript im Verzeichnis `qftest-9.0.0/bin` unter QF-Tests Wurzelverzeichnis. Damit können Sie QF-Test einfach durch Eingabe von `qftest` starten.

Unter Windows wird ein Menüeintrag und auf Wunsch auch ein Desktopicon für QF-Test erstellt. Sie können QF-Test entweder darüber, oder durch einen Doppelklick auf eine Testsuite oder ein Protokoll aufrufen, da diese Dateien mit QF-Test verknüpft sind. Um QF-Test von der Konsole zu starten, geben Sie einfach `qftest` ein.

Für den Aufruf von der Kommandozeile stehen diverse Argumente, wie z.B. die zu verwendende Java-VM, zur Verfügung, die ausführlich in Kapitel 44(971) beschrieben werden.

Wenn gleichzeitig unterschiedliche Versionen von QF-Test installiert sind, kann eine bestimmte Version durch den gezielten Aufruf des `qftest` Programms direkt aus dem jeweiligen `qftest-X.Y.Z/bin` Verzeichnis gestartet werden.

Mac

Wenn QF-Test aufgrund von inkorrekten Einstellungen unter Optionen->Allgemein->Programmstart nicht mehr korrekt startet, können die Standard Einstellung wieder hergestellt werden. Dazu müssen in einem macOS Terminal

folgende zwei Kommandos ausgeführt werden.

```
defaults write de.qfs.qftest /de/qfs/qftest/ \
        -dict-add JVMOptions/ '{"Xmx"="-Xmx1024m";"Xms"="-Xms16m";}'
defaults write de.qfs.qftest /de/qfs/qftest/ \
        -dict-add JVMArguments/ '{"args"="";}'
```

Beispiel 1.1: Zurücksetzen der Programmstart Optionen unter macOS

1.8 Firewall Sicherheitswarnung

Beim Start von QF-Test und/oder der zu testenden Anwendung über QF-Test kann eine Sicherheitswarnung der Windows-Firewall auftreten mit der Frage, ob Java geblockt werden soll oder nicht. Da QF-Test Netzwerkprotokolle für die Kommunikation mit dem SUT (System under Test) nutzt, darf dies **nicht** geblockt werden, um das automatisierte Testen zu ermöglichen.

Kapitel 2

Bedienung von QF-Test

Dieses Kapitel beschreibt den Aufbau des QF-Test Hauptfensters und wie Sie sich darin zurechtfinden. Spätestens nach diesem Kapitel ist es an der Zeit, QF-Test zu starten und verschiedene Dinge auszuprobieren. Über den Menüeintrag Hilfe→Tutorial können Sie Ihren Browser mit dem QF-Test ”learning-by-doing” Tutorial starten. Sollte das mit Ihrer Systemkonfiguration nicht klappen, finden Sie das Tutorial im Verzeichnis `qftest-9.0.0/doc/tutorial`. Dort gibt es auch eine PDF Variante.

Video

Der erste Teil des Videos

’Das Hauptfenster und das System under Test’
https://www.qftest.com/de/yt/hauptfenster-sut-40.html

befasst sich mit den Bestandteilen des QF-Test Hauptfensters.

2.1 Die Testsuite

Zum Automatisieren von GUI Tests benötigt man zwei Dinge: Kontrollstrukturen und Daten. Die Kontrollstrukturen legen fest, was zu tun ist und wann es zu tun ist. Die Daten für die Tests setzen sich aus den Informationen über die GUI-Komponenten des SUT (System under test), den zu simulierenden Events und den erwarteten Ergebnissen zusammen.

QF-Test vereint alles zusammen in einer hierarchischen Baumstruktur, die wir als *Testsuite* bezeichnen. Die Elemente des Baums werden *Knoten* genannt. Knoten können weitere Knoten enthalten, in bestem ”Denglish” *Childknoten* (oder kurz *Children*) genannt. Beim übergeordneten Knoten reden wir vom *Parentknoten* (kurz *Parent*). Der *Wurzelknoten* des Baums repräsentiert die Testsuite als Ganzes.

Es gibt mehr als 60 verschiedene Arten von Knoten, die alle im Detail im Referenzteil[595] beschrieben werden. Manche Knoten fungieren nur als Datencontainer, während ande-

re den Ablauf von Tests steuern. Alle haben ihre speziellen Eigenschaften in Form von Attributen.

Die jeweiligen Attribute des gerade selektierten Knotens werden zur Rechten des Baums in der Detailansicht dargestellt. Diese ist über das Menü Ansicht→Details ein- oder ausschaltbar.

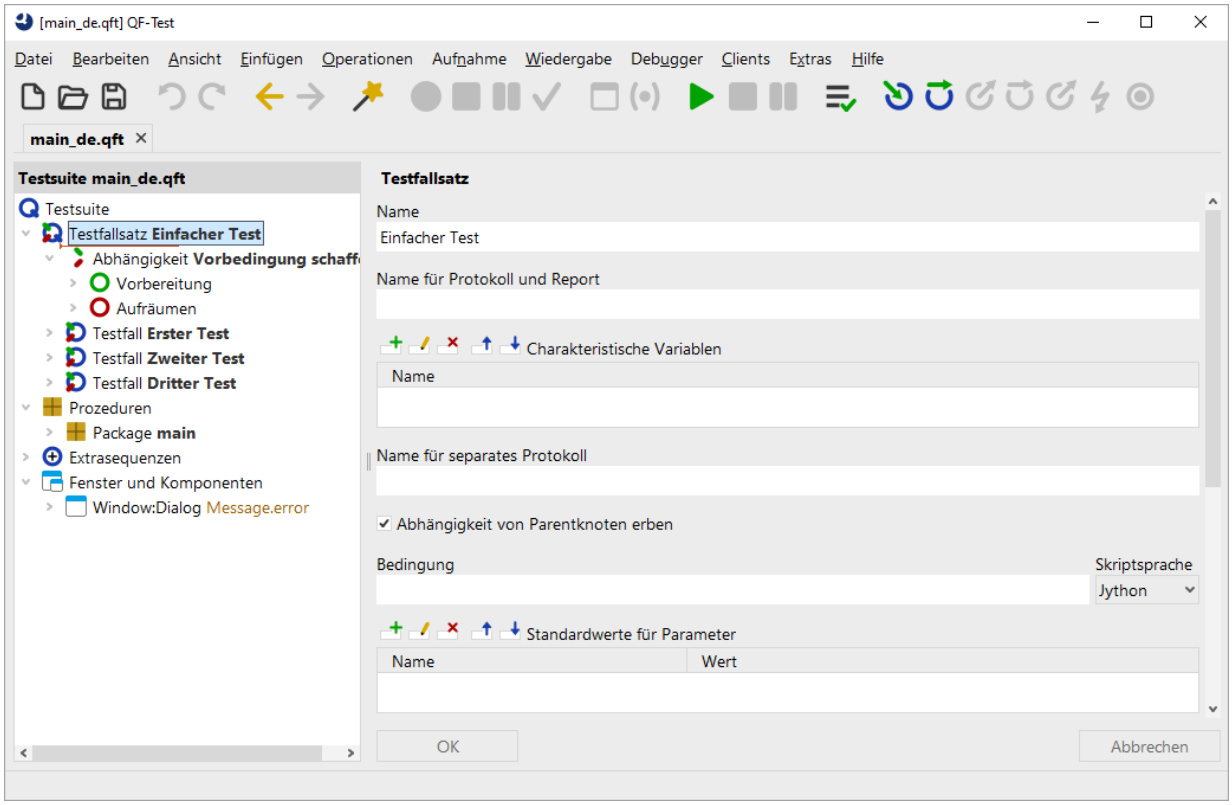


Abbildung 2.1: Struktur einer Testsuite

Die obige Abbildung zeigt ein einfaches Beispiel einer Testsuite. Die Attribute des Knotens ”Einfacher Test” können rechts in der Detailansicht bearbeitet werden.

Eine Testsuite - und damit der Testsuite(595) Wurzelknoten - besitzt die folgende Grundstruktur: Eine beliebige Anzahl von Testfallsatz(606) oder Testfall(599) Knoten, gefolgt von den Prozeduren(682), Extrasequenzen(629) und Fenster und Komponenten(942) Knoten. Der Prozeduren Knoten enthält die Packages(680) und Prozeduren(672), die in Abschnitt 8.5(157) beschrieben werden. Der Extrasequenzen Knoten ist eine Art Spielwiese oder Clipboard, wo alle Arten von Knoten für Experimente oder zur Zwischenlagerung abgelegt werden können. Die Fenster und Komponenten, die die Oberfläche des SUT bilden, werden durch Fenster(919) und Komponente(930) Knoten repräsentiert und sind unterhalb des Fenster und Komponenten Knotens angesiedelt.

Um detaillierte Informationen zu einem Knoten oder einem seiner Attribute zu bekommen, klicken Sie darauf mit der rechten Maustaste und wählen Sie im Kontextmenü Was ist das?. Dies öffnet einen HTML-Browser mit dem entsprechenden Abschnitt des Handbuchs.

2.2 Bearbeiten einer Testsuite

Zum Editieren einer Testsuite gibt es sowohl Funktionen zur Bearbeitung der Knoten als Ganzes, wie Ausschneiden/Kopieren/Einfügen, als auch für deren Attribute. Letztere können direkt in den Feldern der Detailansicht editiert werden, was durch Drücken des *OK* Knopfs oder Return abgeschlossen werden muss. Alternativ kann mittels Bearbeiten→Eigenschaften oder Alt-Return ein eigener Dialog für die Attribute des selektierten Knotens geöffnet werden. Wenn Sie Werte in der Detailansicht ändern und vergessen, *OK* zu drücken, bevor Sie einen anderen Knoten selektieren, wird ebenfalls ein Dialog geöffnet, in dem Sie Ihre Änderungen abschließen oder verwerfen können. Dieses Feature kann über die Option Vor implizitem Übernehmen von Änderungen in den Details nachfragen(494) abgeschaltet werden.

Für die Bearbeitung von mehrzeiligen Textfeldern gibt es einige nützliche Tastaturkürzel: Strg-TAB und Shift-Strg-TAB bewegen den Tastaturfokus aus dem Textfeld heraus, Strg-Return entspricht einem Klick auf den *OK* Knopf.

Eine äußerst nützliche Funktion ist Bearbeiten→Rückgängig machen (Strg-Z), auch *Undo* genannt. Sie macht jede Art von Änderung an der Testsuite rückgängig, inklusive Aufnahmen oder Änderungen mit Hilfe des Suchen/Ersetzen Dialogs. Änderungen werden dabei Schritt für Schritt aufgehoben. Die Anzahl der verfügbaren Schritte ist in der Option Anzahl der zurücknehmbaren Änderungen pro Suite(495) einstellbar, Standard ist 30. Sollten Sie zu viele Schritte rückgängig gemacht haben, können Sie diese mittels Bearbeiten→Wiederherstellen (Strg-Y) wieder herstellen.

2.2.1 Navigation im Baum

Auch wenn die Tastaturkürzel für die Navigation im Baum den üblichen Konventionen entsprechen, sollen sie hier noch einmal genannt werden. Darüber hinaus bietet QF-Test einige hilfreiche Zusatzfunktionen.

Die Pfeiltasten dienen zur grundlegenden Navigation. Hoch und Runter sind offensichtlich, Rechts expandiert entweder einen geschlossenen Knoten, oder geht zum nächsten Knoten, während Links entweder einen geöffneten Knoten schließt, oder zu seinem Parentknoten springt.

Bäume in QF-Test erlauben eine spezielle Variante von Mehrfachselektion. Es können mehrere nicht zusammenhängende Regionen selektiert werden, aber nur unter den Knoten eines gemeinsamen Parentknotens. Wäre Mehrfachselektion über den ganzen Baum erlaubt, würde Ausschneiden und wieder Einfügen in totalem Chaos enden. Tastaturkürzel zum Ausprobieren sind Shift-Hoch und Shift-Runter um die Selektion zu erweitern, Strg-Hoch und Strg-Runter für Bewegungen ohne Änderung der Selektion und Leertaste zum Umschalten der Selektion des aktuellen Knotens. Analog erweitern Mausklicks in Kombination mit der Shift Taste die Selektion, während Klicks mit Strg die Selektion des angeklickten Knotens umschalten.

Spezielle Kürzel sind Alt-Rechts und Alt-Links, welche rekursiv einen Knoten mit all seinen Childknoten aus- oder einklappen,. Alt-Runter und Alt-Hoch navigieren zum nächsten oder vorhergehenden Knoten auf der selben Ebene, dazwischen liegende Childknoten werden übersprungen.

Intern merkt sich QF-Test die zuletzt angesprungenen Knoten. Strg-Backspace navigiert zum zuletzt selektierten Knoten, auch mehrere Schritte zurück.

Zu guter Letzt sollten noch Strg-Rechts und Strg-Links erwähnt werden, mit deren Hilfe der Baum horizontal gescrollt werden kann, wenn er zu groß für seinen Rahmen geworden ist.

2.2.2 Einfügemarkierung

Wenn Sie neue Knoten anlegen oder eine Aufnahme oder kopierte Knoten in die Testsuite einfügen, zeigt Ihnen die Einfügemarkierung an, wo die neuen Knoten landen werden.

Abbildung 2.2: Einfügemarkierung

Knoten werden immer nach dem selektierten Knoten eingefügt. Ist der selektierte Knoten geöffnet, werden der oder die neuen Knoten Children des selektierten Knotens, andernfalls landen sie auf der gleichen Ebene.

Im Beispiel der obigen Abbildung 2.2[18] würde ein neuer Knoten als erstes Child der Sequenz "Events" vor dem Mausevent eingefügt.

2.2.3 Knoten verschieben

Knoten können kopiert und eingefügt werden oder innerhalb einer Testsuite oder über Testsuite Grenzen hinweg verschoben werden. Die üblichen Tastaturkürzel Strg-X, Strg-C und Strg-V werden dazu ebenso unterstützt wie entsprechende Einträge im Kontextmenü.

2.0+ Alternativ können Knoten auch durch Ziehen mit der Maus, genannt *Drag&Drop*, verschoben werden. Normalerweise werden der oder die markierten Knoten bei dieser Operation verschoben. Durch Drücken der Strg Taste während des Loslassens können die Knoten stattdessen kopiert werden.

Während die Maus über den Baum einer Testsuite gezogen wird zeigt die Einfügemarkierung an, wo die Knoten beim Loslassen der Maus eingefügt werden und ob die Operation so erlaubt ist. Eine grüne Markierung signalisiert erwartungsgemäß eine gültige Operation, eine rote Markierung eine ungültige. In letzterem Fall passiert beim Loslassen der Maus gar nichts.

Während des Ziehens können Sie Knoten im Baum auf- oder zuklappen indem Sie mit dem Mauszeiger über den entsprechenden Schalter fahren und dort kurz verharren. Auf diesem Weg können Sie einfach zur Zielposition navigieren, ohne die Drag&Drop Operation abbrechen und neu starten zu müssen.

Die Drag&Drop Operation kann jeder Zeit durch Drücken der Esc Taste abgebrochen werden.

2.2.4 Knoten konvertieren

Manche Knoten können in einen anderen Knotentyp konvertiert werden, was wesentlich bequemer ist als erst den gewünschten Knoten anzulegen und dann die benötigten Attribute zu kopieren. Beispiele von austauschbaren Knoten sind Sequenz[618] und Testschritt[621] oder Server-Skript[717] und SUT-Skript[720]. Die Konvertierung eines Knotens ist nur dann möglich, wenn seine Childknoten und seine aktuelle Position im Baum auch für den gewünschten Zielknoten gültig sind. Mögliche Ziele der Konvertierung können unter dem Eintrag Knoten konvertieren in im Kontextmenü ausgewählt werden. Ist dieser Eintrag nicht verfügbar, gibt es keine gültigen Zielknoten. In diesem Fall kann es hilfreich sein, den Ausgangsknoten zunächst unter die Extrasequenzen[629] zu verschieben.

Sie finden weitere Details über den Konvertierungsmechanismus unter Details zu Knotenkonvertierungen[1306].

2.2.5 Tabellen

An verschiedenen Stellen verwendet QF-Test Tabellen zur Darstellung und Bearbeitung von Werten, z.B. um Variablen(116) zu definieren oder für Checks(805) von mehreren Elementen.

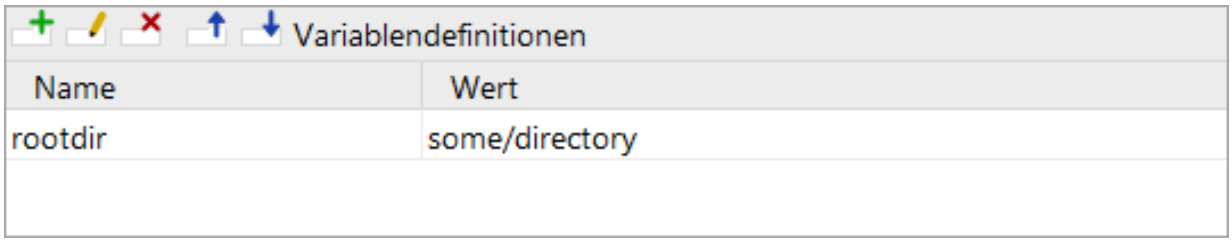


Abbildung 2.3: Beispieltabelle

Die Buttons oberhalb der Tabelle haben folgende Tastaturkürzel und Bedeutung:

Shift-Einfg
Zeile einfügen. Legt eine neue Zeile an.

Shift-Return, Alt-Return
Zeile bearbeiten. Öffnet einen Dialog mit Eingabefeldern für alle Zellen der aktuellen Zeile.

Shift-Entf
Zeile löschen. Entfernt die aktuelle Zeile.

Shift-Strg-Hoch
Zeile nach oben verschieben. Verschiebt die aktuelle Zeile um eine Position nach oben.

Shift-Strg-Runter
Zeile nach unten verschieben. Verschiebt die aktuelle Zeile um eine Position nach unten.

Manche Tabellen bieten zusätzlich die Möglichkeit, Spalten anzulegen und zu löschen, sowie die Überschriften der Spalten zu bearbeiten und stellen dazu die folgenden Buttons bereit:

Spalte einfügen. Fügt eine neue Spalte ein.

Spalte löschen. Löscht die selektierte Spalte.

Spalte bearbeiten. Öffnet einen Dialog zum Bearbeiten der Spaltenüberschrift.

Um einen Wert direkt in der Tabelle zu editieren, können Sie einfach losschreiben. Dabei wird der aktuelle Wert überschrieben. Um den vorhandenen Wert zu verändern, drücken Sie zunächst F2 oder selektieren Sie die Zelle mit einem Doppelklick. Zum Abschluss der Bearbeitung drücken Sie Return, abbrechen können Sie mit Escape. Wenn Sie versuchen, einen unzulässigen Wert einzugeben, färbt sich der Rand der Zelle rot und Sie können die Eingabe nicht übernehmen.

Die Tabellen unterstützen Mehrfachselektion der Zeilen (Mausklicks mit Shift/Strg, Shift/Strg-Hoch/Runter) sowie Ausschneiden, Kopieren und Einfügen von ganzen Zeilen (Strg-X/C/V). Das Einfügen ist dabei auf Tabellen mit gleichartigen Spalten beschränkt.

Im Kontextmenü der Tabelle können noch weitere Aktionen verfügbar sein, z.B. Anzeigen von Zeilennummern, Komponente finden, etc.

Ein Mausklick auf die Spaltenüberschrift aktiviert die Sortierung der Zeilen. Ein Doppelklick auf die Spaltenüberschrift passt die Spaltenbreite automatisch an den breitesten Wert innerhalb der Spalte an oder erlaubt das Editieren des der Spaltenüberschrift (bei Datentabelle).

2.2.6 Packen und Entpacken

Während der Entwicklung von Tests ist es ab und zu notwendig, mehrere Knoten unter einen neuen Elternknoten zu verschieben. Eine typische Situation hierfür ist das Refactoring von Prozeduren bzw. das Reorganisieren dieser Knoten in Packages oder einen Ablauf in einen Try/Catch Block zu packen.

Für solche Anforderungen erlaubt QF-Test dem Benutzer Knoten zu packen. Dies kann mittels Rechtsklick auf die entsprechenden Knoten und Auswahl von Knoten einpacken und des gewünschten neuen Elternknotens erreicht werden.

QF-Test erlaubt ebenfalls Knoten wieder zu entpacken und deren Elternknoten zu löschen. Diesen Ansatz kann man z.B. verwenden, um Test unnötige Packages bzw. Testfallsätze zu löschen oder einen Try/Catch Block wieder zu entfernen. Um Knoten zu entpacken selektieren Sie den entsprechenden Knoten und wählen im Kontextmenü Knoten entpacken aus.

Hinweis Die Pack- und Entpackaktionen werden nur dann im Kontextmenü angezeigt, wenn das Einfügen der Kindknoten in der aktuellen Struktur erlaubt ist.

2.2.7 Sortieren von Knoten

QF-Test gibt Ihnen die Möglichkeit Knoten einer Testsuite zu sortieren. Dies erreichen Sie mittels Rechtsklick auf den entsprechenden Knoten und Auswahl von

Knoten rekursiv sortieren. Alternativ können Sie auch mehrere Knoten markieren und mittels Knoten sortieren nur die markierten Knoten in eine geordnete Reihenfolge bringen.

Der Algorithmus sortiert zur Zeit Ziffern vor Großbuchstaben und diese vor Kleinbuchstaben. Neben der bestehenden Grundstruktur von QF-Test, die eingehalten werden muss, besteht noch die Regel, dass Package Knoten immer vor Abhängigkeit Knoten und diese vor Prozedur Knoten einsortiert werden.

Hinweis Die Grundstruktur einer Testsuite wird vom Sortieren nicht verändert. In einer Testsuite können Sie zwar Testfälle und Prozeduren sortieren, allerdings wird der Prozeduren Knoten immer vor dem Fenster und Komponenten Knoten sein.

2.3 Erweiterte Bearbeitungsmöglichkeiten

Dieser Abschnitt erläutert die etwas komplexeren Bearbeitungsmöglichkeiten wie Suchen/Ersetzen oder die Arbeit mit mehreren Ansichten der selben Testsuite.

2.3.1 Suchen

QF-Test bietet zwei Arten von Suchoperationen, eine allgemeine Suche durch alle Knoten und deren Attribute in Testsuiten und Protokollen sowie eine inkrementelle Suche in Textfeldern, darunter auch Script-Konsolen und Programmausgaben.

Allgemeine Suche

Obwohl die Funktionen zum Suchen und Ersetzen vieles gemeinsam haben, gibt es signifikante Unterschiede, vor allem in der Reichweite der Operation. Eine Suche startet normalerweise beim selektierten Knoten und geht von dort den Baum bis zum Ende durch. Nach einer entsprechenden Rückfrage wird die Suche am Anfang des Baums fortgesetzt und endet spätestens beim ursprünglichen Startknoten, so dass jeder Knoten des Baums genau einmal durchlaufen wird. Dieses Vorgehen ist analog zur Suchfunktion in gängigen Textverarbeitungen und sollte intuitiv zu benutzen sein.

Anfangs öffnet QF-Test die 'einfache' Suche. Diese erlaubt Ihnen nach einen bestimmten Text zu suchen.

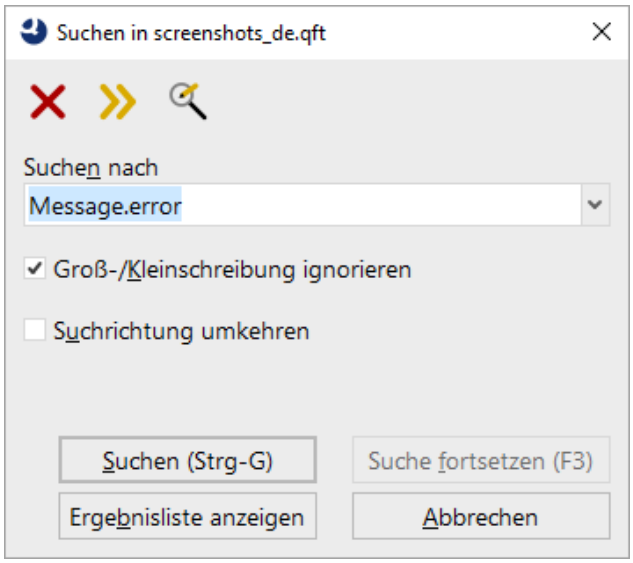


Abbildung 2.4: Der Dialog für die einfache Suche

Für eine spezifischere Suche, z.B. nach Attributen, Knotentypen oder bestimmten Zuständen von Knoten können Sie in den 'Fortgeschrittenen' Suchmodus wechseln. Hierfür klicken Sie auf den 'Suchmodus wechseln' Knopf in der Werkzeugleiste.

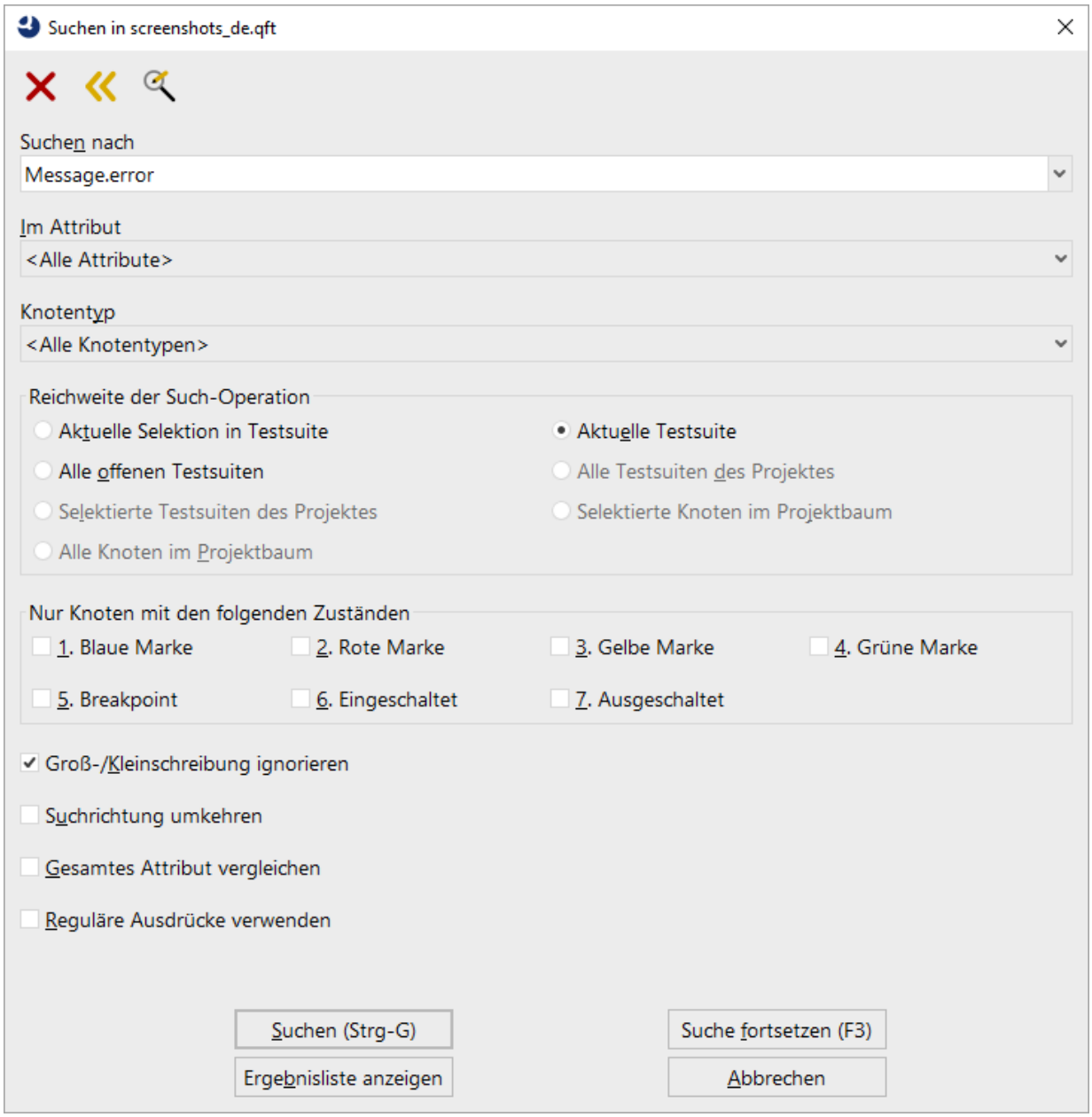


Abbildung 2.5: Der fortgeschrittene Suchdialog

Normalerweise durchsucht QF-Test alle Attribute in allen Arten von Knoten nach dem gewünschten Text.

Mit Hilfe der Option ”Im Attribut” lässt sich die Suche auf ein spezielles Attribut einschränken.

Über die Option ”Knotentyp” kann man die Suche auf eine spezielle Art von Knoten einschränken.

Die Option ”Reichweite der Such-Operation” setzt fest, wo nach dem angegebenen Ausdruck gesucht werden soll. Dies kann entweder in den aktuell selektierten Knoten, in der gesamten Testsuite oder in allen offenen Testsuiten geschehen.

Wenn Sie die Option ”Nur Knoten mit den folgenden Zuständen” auswählen, wird nur nach Knoten gesucht, welche den angegebenen Status besitzen, z.B. einer grünen Marke oder einen Breakpoint.

Ist ”Gesamtes Attribut vergleichen” aktiviert, muss der gesuchte Text genau einem Wert entsprechen, andernfalls genügt es, wenn der Suchtext im Attribut enthalten ist.

Reguläre Ausdrücke werden in Abschnitt 49.3(1023) erklärt.

Hinweis

Um nach Werten von Booleschen Attributen wie Als ”harten” Event wiedergeben(778) zu suchen, verwenden Sie ”true” oder ”false” (ohne Anführungsstriche) als Suchtext. Falls Sie nach leeren Werten suchen möchten, müssen Sie ”Gesamtes Attribut vergleichen” anhaken.

Wenn die Suche erfolgreich ist, wird der gefundene Knoten markiert und eine Meldung in der Statuszeile ausgegeben, die den Namen des gefundenen Attributs angibt.

3.4+

Wie bereits erwähnt, startet die Suche normalerweise vom gerade selektierten Knoten. Falls Sie allerdings während des Suchprozesses die Selektion ändern möchten und trotzdem die vorherige Suche fortsetzen wollen, dann können Sie dies mit dem Button ”Suche fortsetzen” erreichen.

Wenn der Suchdialog geschlossen wurde, können Sie die Suche immer noch fortsetzen, in dem Sie F3 drücken. Mittels Ctrl-G können Sie sogar die selbe Suche von einem neuen Knoten aus starten.

Ein sehr nützliches Feature ist die Fähigkeit nach allen Prozeduraufruf Knoten zu suchen, die eine bestimmte Prozedur aufrufen, oder nach allen Events, die eine bestimmte Komponente benötigen. Wählen Sie hierzu einfach den Menüeintrag Referenzen finden... im Kontextmenü des Knotens, der aufgerufen oder referenziert wird. Danach wird ein neues Fenster mit allen Referenzen auf den ausgewählten Knoten angezeigt. In diesem Fenster können Sie mittels Doppelklick auf einen Eintrag zur jeweiligen Stelle in der Testsuite springen.

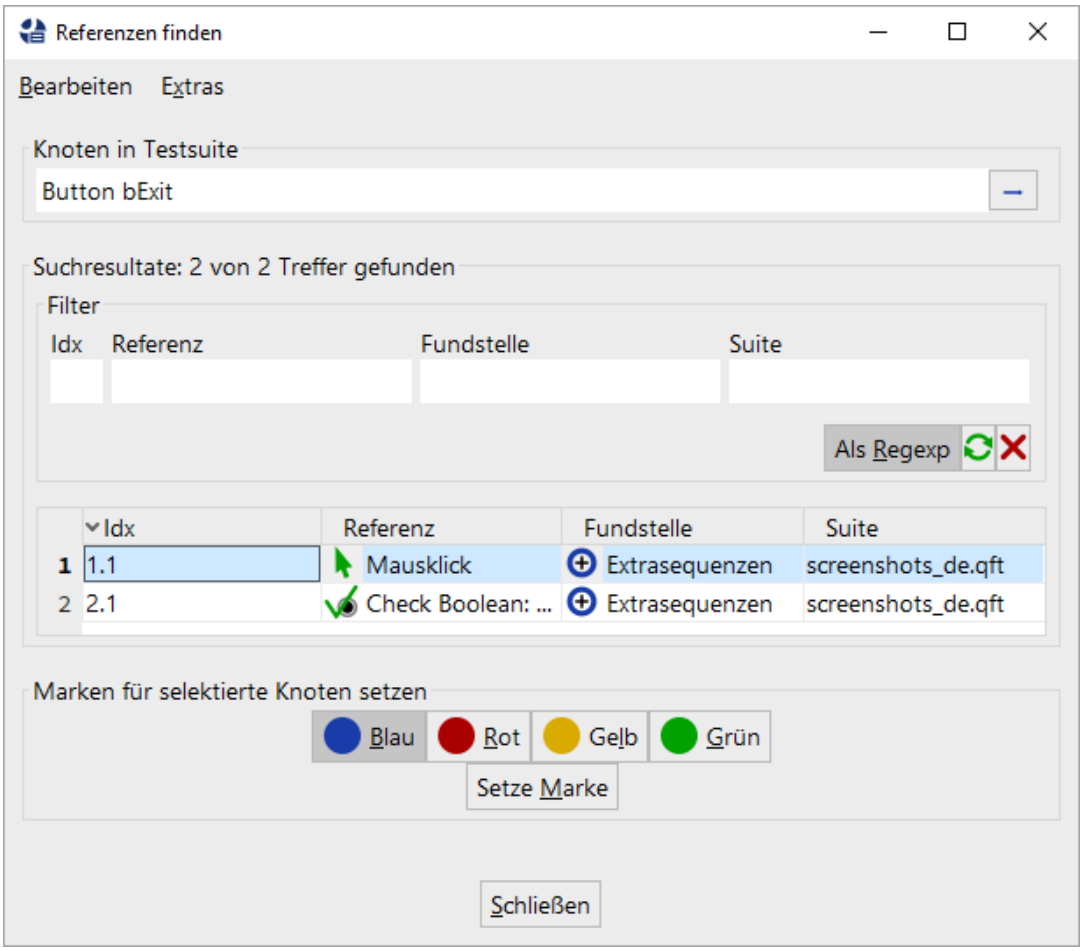


Abbildung 2.6: Resultatsliste von 'Referenzen finden'

3.1+ Es ist ebenso möglich alle gefundenen Knoten aus dem Suchdialog mittels Klick auf "Ergebnisliste anzeigen" zu erreichen. Von diesem Fenster aus, können Sie auch wiederum jeden einzelnen Knoten der Testsuite erreichen.

Inkrementelle Textsuche

3.1+ Zusätzlich zur Suche im Baum gibt es die Möglichkeit für inkrementelles Suchen in Komponenten, die Text enthalten, wie z.B. das Terminal oder entsprechende Attribute in der Detailansicht. Diese Features kann über den Suchen... Eintrag im Kontextmenü der Komponente aktiviert werden oder durch Drücken von Strg-F, wenn die Komponente ausgewählt ist und den Tastaturfokus besitzt. Dann erscheint das Eingabefenster für die inkrementelle Suche am rechten oberen Rand der entsprechenden Komponente. Die folgende Abbildung zeigt eine inkrementelle Suche für das Terminal mit hervorge-

hobenen Suchtreffern.

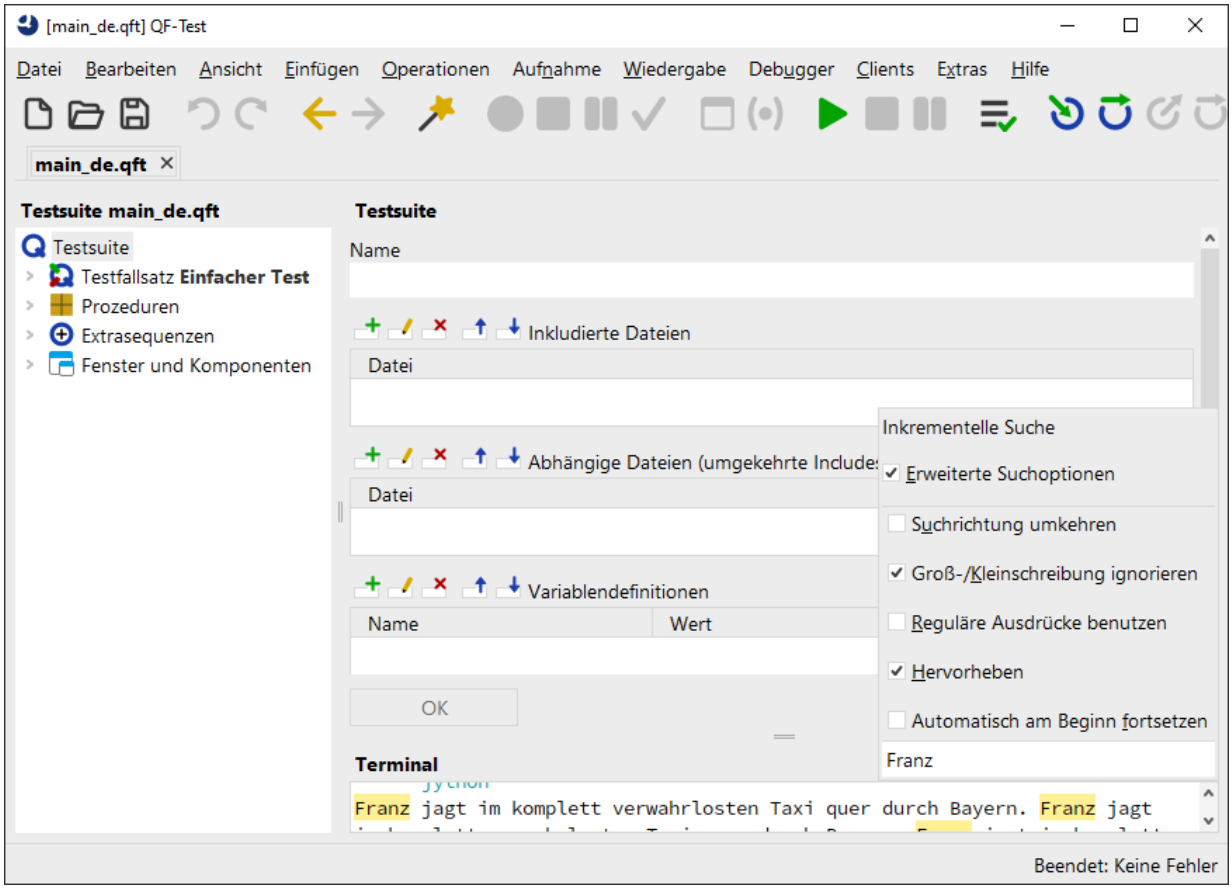


Abbildung 2.7: Die inkrementelle Suche

Die Suche kann auf einen bestimmten Teil des enthaltenen Textes beschränkt werden, in dem man den gewünschten Bereich selektiert und den Suchen in Auswahl... Eintrag im Kontextmenü der Komponente wählt oder Strg-Umschalt-F drückt.

Ansonsten sollte die inkrementelle Suche und auch die verfügbaren Optionen selbsterklärend sein.

2.3.2 Ersetzen

Wenn man einmal die Besonderheiten beim Gültigkeitsbereich verinnerlicht hat, ist der Ersetzen Dialog ebenso intuitiv zu benutzen wie der Suchdialog. Beim Ersetzen haben Sie die Möglichkeit, einen Treffer nach dem anderen zu ersetzen, oder alle auf einmal. Um bei Letzterem unerwartete Resultate zu vermeiden bedarf es einer Möglichkeit, den Bereich für das Ersetzen einzuschränken. Daher werden beim Ersetzen

nur die selektierten Knoten sowie deren direkte oder indirekte Childknoten durchsucht. Um Ersetzungen im gesamten Baum durchzuführen, selektieren Sie den Wurzelknoten oder wählen Sie im Ersetzen Dialog die entsprechende Einstellung unter ”Reichweite der Ersetzen-Operation”.

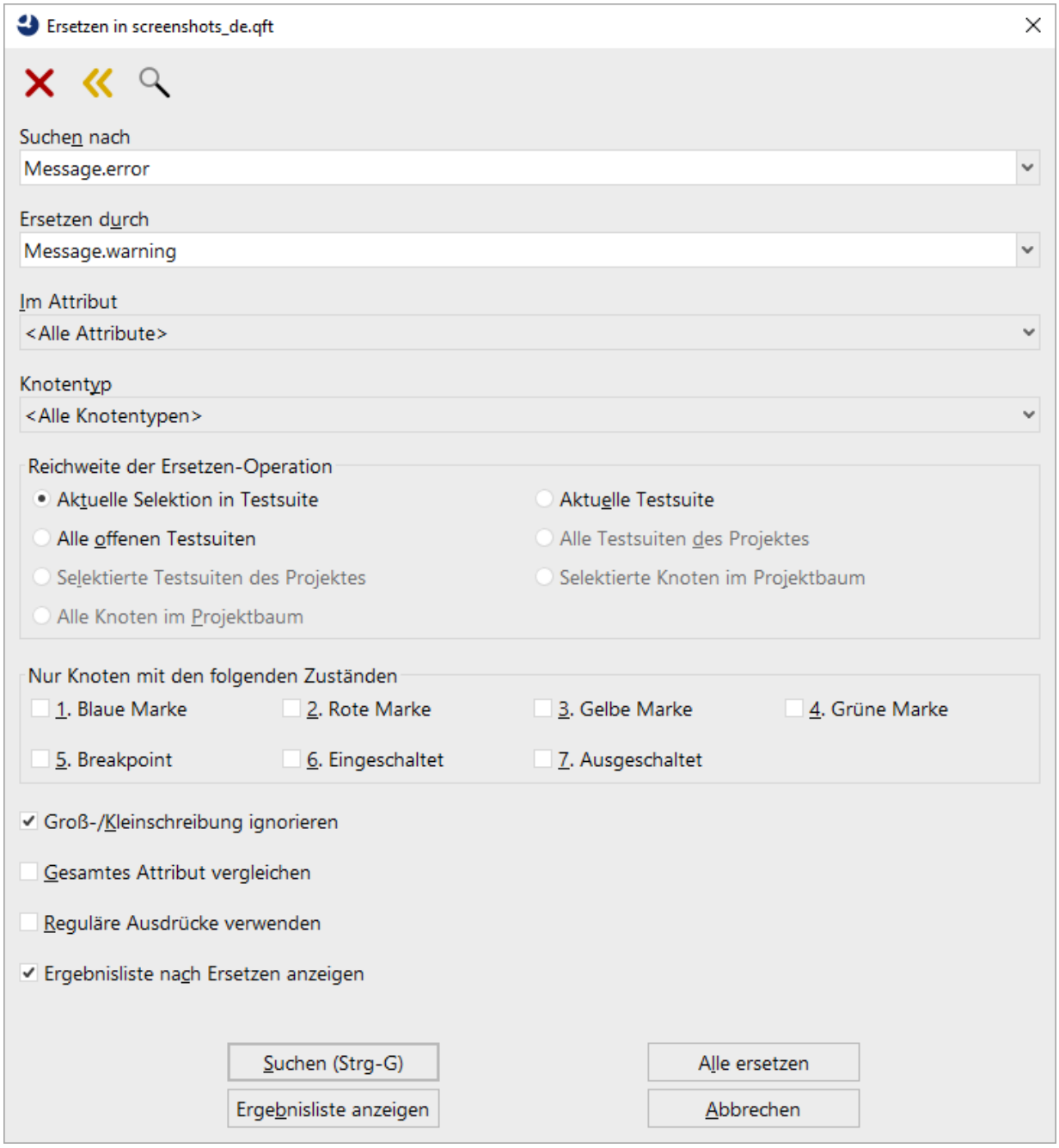


Abbildung 2.8: Der Ersetzen Dialog

Die Optionen entsprechen denen im Suchdialog. Ist ”Gesamtes Attribut vergleichen” ausgeschaltet, sind mehrere Ersetzungen innerhalb eines Attributs möglich. Zum Beispiel würde das Ersetzen von ”a” durch ”b” im Wort ”ananas” zu ”bnbnbs” führen. Für den Einsatz von regulären Ausdrücken lesen Sie bitte Abschnitt 49.3(1023).

Ist die Suche erfolgreich, wird der gefundene Knoten selektiert und ein Dialog geöffnet, der das gefundene Attribut mit dem alten und neuen Wert anzeigt.

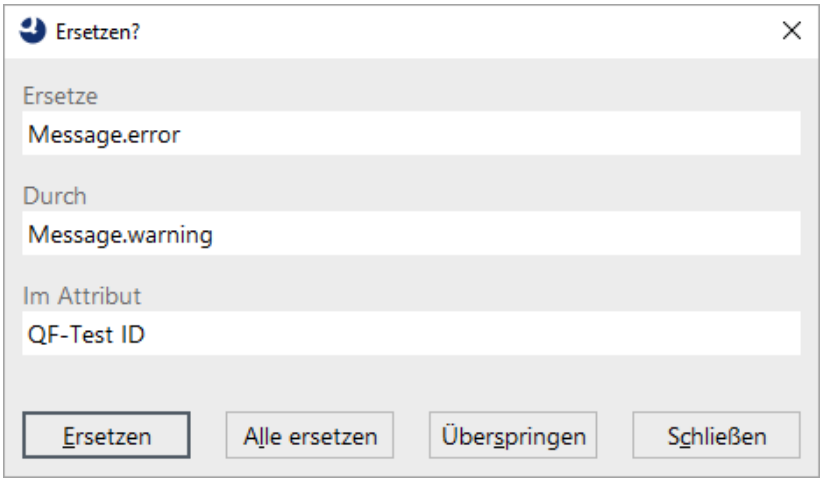


Abbildung 2.9: Der Rückfrage Dialog beim Ersetzen

Sie haben folgende Möglichkeiten:

- Ein Klick auf *Ersetzen* ändert das Attribut und setzt die Suche nach dem nächsten Wert fort.
- *Alle ersetzen* ändert dieses Attribut sowie alle weiteren Treffer ohne nochmalige Rückfrage.
- *Überspringen* lässt dieses Attribut unverändert und setzt die Suche fort.
- Erwartungsgemäß beendet *Abbrechen* den gesamten Vorgang.

Wenn Sie sich bei den Eingaben sicher sind, können Sie gleich im Ersetzen Dialog *Alle ersetzen* wählen und damit die Rückfrage überspringen. Nachdem die Attribute geändert wurden, wird die Anzahl der betroffenen Knoten in einem Meldungsdialog ausgegeben.

3.1+ Nach dem Ersetzen wird von QF-Test ein Ergebnisfenster mit allen angepassten Knoten angezeigt. Sie können dieses Fenster auch vor der eigentlichen Aktion mittels Klick auf ”Ergebnisliste anzeigen” im Ersetzen Dialog erreichen.

Hinweis Ob Werte einzeln oder in einem Rutsch ersetzt werden, beeinflusst das anschließende

Verhalten der Undo Funktion. Sämtliche durch *Alle ersetzen* ausgelösten Änderungen werden in einem Schritt zurückgenommen, Änderungen in Einzelschritten werden einzeln rückgängig gemacht.

2.3.3 Kompliziertere Such- und Ersetzungsvorgänge

3.3+ Manchmal ist eine einfache Suchoperation nicht ausreichend. Stellen Sie sich zum Beispiel vor, Sie wollen eine Wartezeit von 3000 Millisekunden für alle Check Text Knoten einer bestimmten Komponente setzen. Sie kennen die QF-Test ID der Komponente, aber wenn Sie danach suchen erhalten Sie auch alle anderen Events und Checks für diese. Wenn Sie nach dem Knotentext 'Check Text' suchen, finden Sie dagegen die Check Text Knoten für alle Komponenten, nicht nur die für die gesuchte Komponente.

Anstelle mehrerer kombinierbarer Ebenen von Suchkriterien bietet QF-Test durch seine Marken völlige Flexibilität. Führen Sie zunächst eine Suche nach Ihrem ersten Kriterium durch, z.B. dem Knotentext und wählen Sie 'Ergebnisliste anzeigen'. Im folgenden Ergebnisdialog selektieren Sie durch drücken von Ctrl-A sämtliche Einträge in der Tabelle, wählen dann 'Setze Marke' um allen gefundenen Knoten die blaue Marke zuzuweisen und schließen den Dialog. Sie können nun eine weitere Suche oder einen Ersetzungsvorgang durchführen, dessen Gültigkeitsbereich auf Knoten mit einer bestimmten Marke beschränkt ist. In unserem Beispiel würden Sie nun den leeren Text durch '3000' ersetzen, und zwar auf alle Wartezeit Attribute von Knoten mit blauer Marke im gesamten Baum.

2.3.4 Mehrere Ansichten

Es können mehrere Fenster geöffnet werden, die verschiedene Teile der selben Baumstruktur gleichzeitig anzeigen. Dies kann beim Umgang mit großen Testsuiten oder zum Vergleichen von Attributen mehrerer Knoten nützlich sein.

Zusätzliche Ansichten werden mittels Ansicht→Neues Fenster... geöffnet. Der aktuelle Knoten wird dabei zum Wurzelknoten für die neue Ansicht. Die Zusatzfenster sind analog zu den Hauptfenstern, jedoch mit eingeschränktem Funktionsumfang.

2.3.5 Toolbar-Buttons ausblenden

Sie können die Zahl der angezeigten Knöpfe in der Werkzeugleiste reduzieren, indem Sie den Knopf, den Sie ausblenden wollen, mit der rechten Maustaste anklicken und im Kontextmenü Aus Werkzeugleiste entfernen auswählen.

Um die ursprüngliche Ansicht wiederherzustellen, klicken Sie eine beliebige Stelle in der

Werkzeugleiste und wählen Ganze Werkzeugleiste anzeigen. Wenn Sie alle Buttons ausgeblendet haben, wählen Sie unter Anzeigen→Werkzeugleiste anzeigen.

Kapitel 3

Schnellstart Ihrer Anwendung

2.0+

Dieses Kapitel enthält eine Anleitung, wie Sie auf kürzestem Weg Ihre Anwendung als SUT (System Under Test) einrichten.

Video

Video-Anleitung für Java-Anwendungen:

'Der Schnellstart Assistent Java'
https://www.qftest.com/de/yt/schnellstart-assistent-java-42.html

Video-Anleitung für Web-Anwendungen:

'Der Schnellstart Assistent Web'
https://www.qftest.com/de/yt/schnellstart-assistent-web-42.html

Android

Anleitung für Android-Anwendungen: Eine QF-Test Startsequenz für Android Tests erzeugen(254).

iOS

Anleitung für iOS-Anwendungen: QF-Test Vorbereitung Sequenz für iOS Tests(277).

Damit Ihre Anwendung von QF-Test als SUT wahrgenommen wird, muss sie im einfachen Fall aus QF-Test heraus gestartet werden. Es gibt eine Anzahl von speziellen Prozessknoten unter Einfügen→Prozess Knoten, um diese Aufgabe zu erfüllen. Der empfohlene, komfortablere und sichere Weg ist jedoch die Benutzung des Schnellstart-Assistenten, der gleich im Anschlusskapitel beschrieben wird. Wer jedoch eine strikte Abneigung gegen Assistentendialoge hat, für den wird der händische Weg in Abschnitt 46.1(1000) beschrieben.

Eine Voraussetzung für das Testen von Java basierten SUTs ist, dass sich QF-Test in das GUI-Toolkit einklinken kann:

Swing

Für reine Swing-Anwendungen oder Kombinationen aus Swing und SWT hängt sich QF-Test in die JVM Tools Schnittstelle von Java ein. QF-Test kann dies normalerweise direkt, nur für spezielle JDKs kann es nötig sein, diese zunächst zu instrumentieren. Details dazu finden Sie in Deinstrumentieren eines JRE(1011).

JavaFX Für JavaFX und entsprechende Kombinationen wird die Verbindung über den QF-Test Agent hergestellt. Bitte stellen Sie sicher, dass die Option Über QF-Test Agent verbinden(594) aktiviert ist.

SWT Für das Testen von SWT oder Eclipse basierten Anwendungen kann eine Instrumentierung der SWT Bibliothek erforderlich sein. Der im folgenden beschriebene Schnellstart-Assistent baut bei Bedarf einen entsprechenden Schritt in die Startsequenz ein. Detaillierte technisch Informationen hierzu finden Sie in Abschnitt 47.2(1012).

Web Das Testen von Web-Anwendungen erfordert keine Instrumentierung, jedoch gibt es Einschränkungen zu beachten, die in Kapitel 14(227) näher beschrieben sind.

3.1 Erzeugung der Startsequenz - Schnellstart-Assistent

QF-Test bietet Ihnen mit dem Schnellstart-Assistenten einen komfortablen Weg zum Erzeugen einer Startsequenz, um Ihre Applikation als SUT zu betreiben.

Den Schnellstart-Assistenten können Sie über den Menüeintrag Extras→Schnellstart-Assistent... oder den  Knopf in der Werkzeugleiste aufrufen. Er führt Sie Schritt für Schritt durch die Erstellungsprozedur der Startsequenz und sollte selbsterklärend sein.

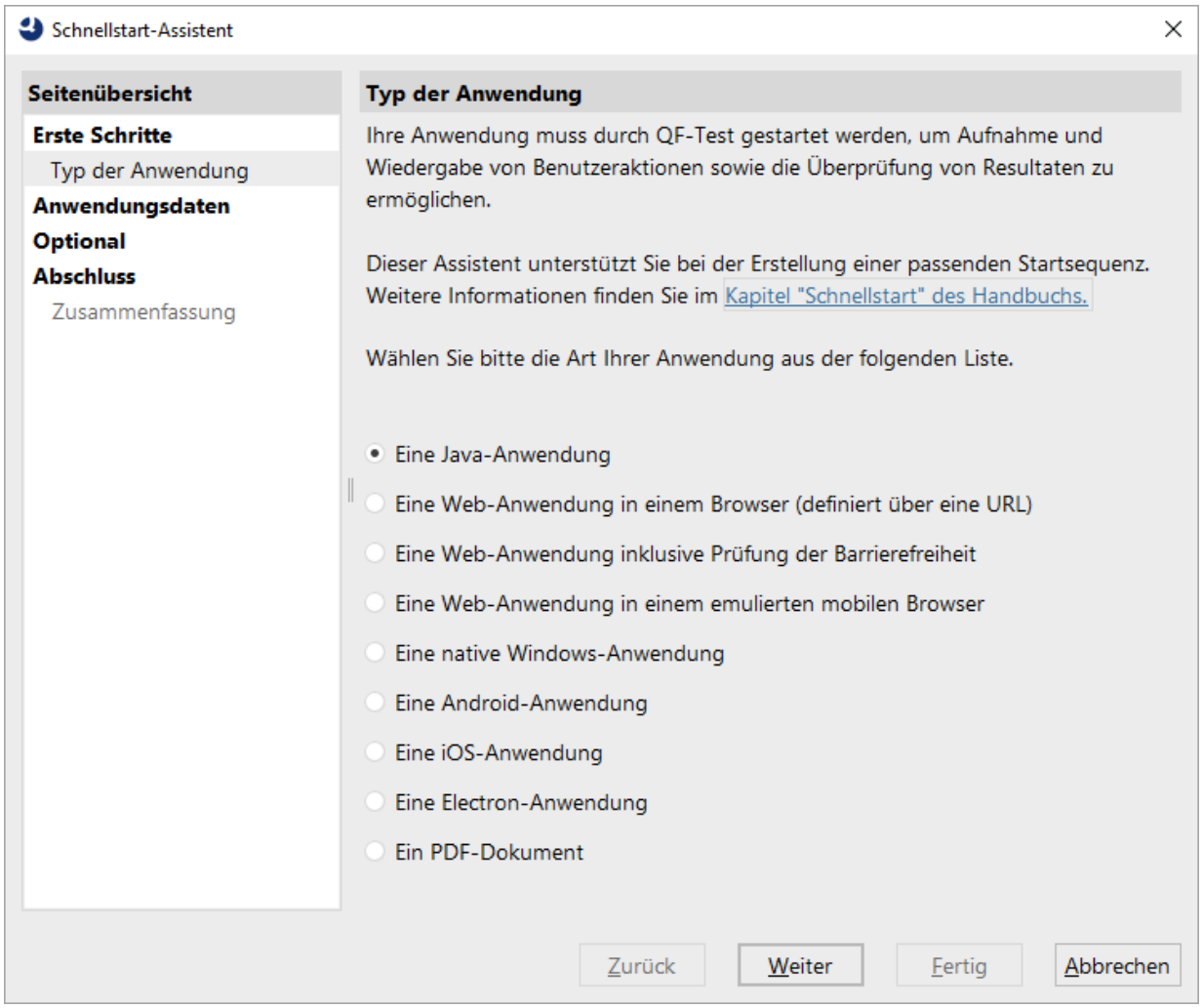


Abbildung 3.1: Der Schnellstart-Assistent

Als Resultat liefert der Assistent unter dem Knoten ”Extrasequenzen” eine Sequenz ”Vorbereitung” zum Starten Ihrer Anwendung, ähnlich der, wie sie im folgenden Bild gezeigt wird:

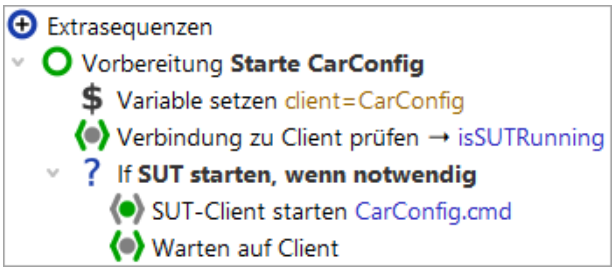


Abbildung 3.2: Startsequenz als Resultat des Schnellstart-Assistenten

Die erstellte Vorbereitungssequenz sieht je nach Typ der Applikation unterschiedlich aus, sie folgen aber alle einen gewissen Aufbau. Zuerst finden Sie einen Variable setzen Knoten, der den Namen des SUT-Clients für QF-Test festlegt. Danach wird ein Warten auf Client Knoten erstellt, der prüft, ob das SUT bereits läuft. Dessen Ergebnis wird im darauf folgenden If Knoten verwendet, in dem das SUT gestartet wird. Der Start erfolgt mittels eines Startknotens, dessen Typ und Attribute von der Art Ihrer Applikation abhängen, gefolgt wiederum von einem Warten auf Client Knoten, welcher überprüft, dass während des Startvorgangs eine Verbindung zwischen Ihrer Applikation und QF-Test zustande kommt. (Details zu den Startknoten für die verschiedenen Anwendungstypen finden Sie in Abschnitt 46.1(1000).)

SWT Bei SWT basierten Anwendungen wird zusätzlich ein Prozeduraufruf für die SWT-Instrumentierung(1012) eingefügt.

Web Die Startsequenz für Web enthält zusätzliche Knoten zum Setzen von Variablen, zur Initialisierung des Browser-Cache und der Einstellungen für Cookies und ggf. zur Installation eines Toolkit-Resolvers. Weitere Information finden Sie in Kapitel 14(227).

3.2 Ausführen der Startsequenz

Die Startsequenz kann nun direkt von Ihnen gestartet werden, in dem Sie den grünen Vorbereitungsknoten im Baum auswählen und den Knopf ”Wiedergabe starten” in der Werkzeugleiste drücken.

Ihre Applikation sollte dabei ordnungsgemäß gestartet und der rote ”Aufnahme starten” Knopf in der QF-Test Werkzeugleiste aktiv werden. Letzteres zeigt an, dass QF-Test erfolgreich die Verbindung zum SUT hergestellt hat.

Nun sind Sie bereit, Ihre ersten Testsequenzen aufzunehmen und wieder ablaufen zu lassen, so wie es in Kapitel 4(39) beschrieben ist. Im Hilfemenü finden Sie das sehr empfehlenswerte Learning-by-doing Tutorial, welches Sie Schritt für Schritt durch die Anwendungskonzepte von QF-Test führt.

Sollten Sie wider Erwarten eine Fehlermeldung bekommen oder der rote ”Aufnahme starten” Knopf inaktiv bleiben, dann gehen Sie bitte zum folgenden Abschnitt.

3.3 Wenn sich der Client nicht verbindet ...

Falls Ihre Anwendung (oder das Browser-Fenster im Falle des Testens von Web-Anwendungen) nicht erscheint:

- Der Fehlerdialog, den QF-Test im Regelfall anzeigt, sollte eine erste Beschreibung liefern.
- Suchen Sie bitte im Terminal-Fenster nach Fehlermeldungen. Wenn das Terminal nicht im unteren Bildbereich von QF-Test sichtbar ist, können Sie es über das Menü Ansicht→Terminal→Anzeigen aktivieren. Weitere Informationen zur Programmausgabe finden Sie auch in Abschnitt 3.4(37).
- Überprüfen Sie die Attribute der Knoten in der Startsequenz nochmals auf Korrektheit. Vielleicht hat sich irgendwo ein Tippfehler eingeschlichen. Details zu den Startknoten für die verschiedenen Anwendungstypen finden Sie in Abschnitt 46.1(1000).

Web

- Da die Entwicklungszyklen von Browsern, besonders bei Firefox, immer kürzer werden, stellen Sie sicher, dass der installierte Browser von der von Ihnen genutzten QF-Test Version unterstützt wird. Die Ausgabe im Terminal-Fenster sollte eine entsprechende Fehlermeldung zeigen. Siehe Abschnitt 1.1.3(4) für den letzten Stand der unterstützten Browser Versionen. Ggf. müssen Sie QF-Test auf einen neuere Version updaten oder temporär mit einem anderen Browser arbeiten. Weitere Informationen finden Sie in Kapitel 14(227).

Falls das SUT erscheint, aber der Aufnahmeknopf inaktiv bleibt:

- Überprüfen Sie bitte die Ausgabe im Terminal (siehe oben) auf mögliche Fehlermeldungen.
- Sollte der rote Aufnahmeknopf nach Erscheinen des Fehlerdialogs ”Keine Verbindung zum Client” doch noch aktiv werden, muss ggf. der Timeout Wert im Warten auf Client(758) Knoten erhöht werden.

SWT

- Für eine Eclipse/SWT-Anwendung stellen Sie zunächst sicher, dass Sie das korrekte Verzeichnis für die Anwendung angegeben haben. Eventuell hilft ein Blick auf das Protokoll (siehe Abschnitt 7.1(138)) um zu sehen, ob bei der Ausführung der Prozedur `qfs.swt.instrument.setup` Fehler oder Warnungen ausgegeben wurden.

- Überprüfen Sie das allgemein das Protokoll auf mögliche Fehlerindikatoren (siehe Abschnitt 7.1[138]).

Nach möglichen Anpassungen an Testsuite oder Einstellungen können Sie ein erneutes Ausführen der Startsequenz probieren. Sollten Sie mit den hier gegebenen Hinweisen nicht weiter kommen, können Sie ggf. die Beispiel-Testsuite aus dem Tutorial ausprobieren oder Sie kontaktieren unseren Support.

3.4 Programmausgaben und das Clients Menü

Die Ausgaben aller von QF-Test gestarteten Prozesse werden von QF-Test umgeleitet und im Protokoll unter dem Knoten gespeichert, der den Prozess gestartet hat. Dabei macht QF-Test keinen Unterschied zwischen SUT Clients und sonstigen Prozessen oder Shellskripten, die mittels eines Programm starten[731] oder Shell-Kommando ausführen[734] Knotens gestartet wurden.

Das Hauptfenster enthält ein gemeinsames Terminal für die Ausgaben aller Prozesse, die von einem Test aus diesem Fenster gestartet wurden. Das Untermenü Ansicht→Terminal enthält einige Menüeinträge, mit deren Hilfe Sie das Terminal ein- und ausschalten oder festlegen können, ob Baum oder Terminal den Bereich links unten nutzen sollen, ob lange Zeilen umgebrochen werden und ob automatisch an das Ende gescrollt werden soll, wenn neue Ausgaben eintreffen. Außerdem können Sie das Terminal löschen oder seinen Inhalt in eine Datei speichern. Die maximale Größe des Terminals wird über die Option Maximalgröße des gemeinsamen Terminals (kB)[539] festgelegt.

Zusätzlich zum gemeinsamen Terminal gibt es für jeden aktiven Prozess und die letzten beendeten Prozesse ein individuelles Terminalfenster, das ebenfalls die Ausgaben des Prozesses anzeigt. Terminalfenster sind über das Clients Menü zugänglich. Das gemeinsame Terminal dient hauptsächlich dazu, auf das Eintreffen neuer Ausgaben aufmerksam zu machen, während die individuellen Terminalfenster besser zum gründlichen Studieren der Ausgaben geeignet sind.

Über das Clients Menü können Prozesse auch beendet werden, entweder einzeln oder alle auf einmal mittels Clients→Alle Clients beenden.

Die Anzahl der beendeten Prozesse, die im Clients Menü verfügbar sind, wird über die Option Wie viele beendete Clients im Menü[536] festgelegt (Standard ist 4). Wenn Ihre Prozesse sehr viele Ausgaben erzeugen, kann es sinnvoll sein, diese Zahl zu reduzieren um Speicher zu sparen.

Hinweis

Das Clients Menü ist auch hilfreich, wenn man nicht sicher ist, welches spezifische QF-Test Produkt man kaufen soll. Die von Ihrer Anwendung genutzten GUI-Technologien werden neben dem aktiven Client-Namen in '[ ]' dargestellt. Das Beispiel unten zeigt

zwei Clients, die Java Swing bzw. Web benutzen, was nahelegt eine QF-Test/swing+web Lizenz zu erwerben.

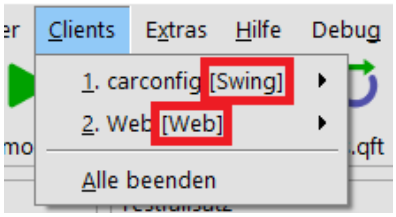


Abbildung 3.3: Information zu genutzten GUI-Technologien

3.5 Indirektes Starten eines zweiten SUT als Kindprozess eines bereits verbundenen SUT

Wird eine zweite Java-VM aus einem bereits verbundenen SUT gestartet, erkennt QF-Test beim Verbindungsaufbau, dass es sich um einen Kindprozess des ersten SUT handelt und vergibt automatisch einen neuen Client-Namen. Hierzu wird dem Namen des ersten SUT ':2' angefügt, was betont, dass es sich um den zweiten Prozess für diesen Client handelt. Einem weiteren derart gestarteten SUT wird ':3' an den Namen angefügt, es sei denn, das zweite SUT ist bereits beendet, so dass ':2' wieder verfügbar ist.

Eine Sequenz zum indirekten Start eines SUT besteht also üblicherweise aus einem Event Knoten der z.B. einen Buttonklick oder eine Menüauswahl auslöst und das erste SUT zum Start des zweiten SUT veranlasst, gefolgt von einem Warten auf Client(758) Knoten für den um ':2' erweiterten Client-Namen.

Kapitel 4

Aufnahme und Wiedergabe

Wenn das SUT erst einmal unter QF-Test läuft, ist der nächste Schritt, Testsequenzen aufzunehmen und wieder abzuspielen.

Video

Die Aufnahme und das Abspielen von Sequenzen ist auch in dem Video

'Capture und Replay'
https://www.qftest.com/de/yt/capture-replay-40.html

erläutert.

Im Video

'Erstellung eines Testfalls'
https://www.qftest.com/de/yt/testfall-40.html

sehen Sie wie ein Testfall erstellt wird.

Android
iOS

Bei Android- und iOS-Applikationen läuft die Aufnahme über ein spezielles Aufnahmefenster. Wenn im Folgenden Aktionen im SUT erwähnt werden, beziehen sich diese bei Android und iOS auf das Aufnahmefenster. Weitere Informationen zum Aufnahmefenster finden Sie für Android in Abschnitt 16.6(265), für iOS in Abschnitt 17.5(282).

4.1 Aufnahme von Testsequenzen

Um eine Sequenz von Events im SUT aufnehmen zu können, muss das SUT aus QF-Test heraus gestartet worden sein (vgl.Kapitel 46(1000)) und die Verbindung zum SUT muss bestehen. Letzteres ist an Hand des Aufnahmeknopfs zu erkennen, der rot dargestellt wird, wenn er aktivierbar ist.

Abbildung 4.1: Gesperrter und aktivierbarer Aufnahmeknopf

Um eine Sequenz aufzunehmen, starten Sie die Aufnahme einfach mittels des Aufnahmeknopfs oder Aufnahme→Start. Wechseln Sie dann zum SUT und führen Sie einige Kommandos aus. Schalten Sie dann zurück zu QF-Test und beenden Sie die Aufnahme mit dem Stopknopf oder Aufnahme→Stop. Die aufgezeichneten Events werden in die Testsuite eingefügt, entweder direkt an der Einfügemarkierung(18), oder als neue Sequenz(618) unterhalb des Extrasequenzen(629) Knotens, je nach Einstellung der Option Aufnahme bei aktueller Selektion einfügen(508). Sie können die Aufnahme mit Aufnahme→Pause oder dem Pausenknopf unterbrechen, wenn Sie einige Schritte im SUT ausführen wollen, die nicht aufgezeichnet werden sollen.

Der Aufnahmemodus kann auch direkt im SUT über den Hotkey für Aufnahme(507) gestartet und gestoppt werden. Standardmäßig ist diese Taste F11.

Alle Komponenten(930), auf die sich die aufgezeichneten Events beziehen, werden automatisch unterhalb des Fenster und Komponenten(942) Knotens in die Testsuite integriert, falls sie dort noch nicht vorhanden sind.

Es gibt viele Optionen, die die Aufnahme von Events und die Verwaltung von GUI-Komponenten beeinflussen. Diese werden ausführlich in Abschnitt 41.2(507) des Referenzteils erläutert. Wenn Sie sich mit QF-Test vertraut gemacht haben, sollten Sie sich die Zeit nehmen und sich zumindest einen Überblick über diese Optionen verschaffen.

Es folgen einige allgemeine Tipps zum Aufnehmen:

- Nehmen Sie nur kurze Sequenzen am Stück auf.
- Sehen Sie sich jede neu aufgenommene Sequenz an, versuchen Sie zu verstehen, was Sie erhalten haben und ob es Ihren Aktionen im SUT entspricht.
- Bearbeiten Sie die Sequenz, um überflüssige Events zu entfernen, insbesondere solche, die von den Wechseln zwischen dem SUT und QF-Test stammen. QF-Test hat ausgezeichnete Filter für solche Events. Dennoch können einzelne Events übrigbleiben, die von Hand entfernt werden müssen.
- Schließlich sollten Sie ausprobieren, ob die neue Sequenz korrekt abgespielt wird. Anschließend können Sie diese weiterbearbeiten und zu größeren Tests zusammensetzen.

Mac

Für SUTs, die auf Mac OS X laufen, wird die Benutzung der Bildschirm-Menüleiste

unterdrückt und die normale Menüleiste aktiviert, äquivalent wie auf anderen Plattformen. Grund dafür ist, dass QF-Test die Bildschirm-Menüleiste nicht vollständig ansprechen kann, was sauberes Aufnehmen/Wiedergeben von Menüaktionen verhindert. Falls das typische Mac Verhalten der Bildschirm-Menüleiste aus irgendwelchen Gründen erforderlich sein sollte, kann es durch das Hinzufügen der Zeile `qfs.apple.noScreenMenuBar=false` in der Datei `qfconnect.properties`, die sich im Wurzelverzeichnis von QF-Test befindet, erzwungen werden. Nach einem Neustart zeigt das SUT dann das für den Mac übliche Verhalten.

Windows-Tests

Bitte beachten Sie bei nativen Windows-Applikationen auch Abschnitt 15.4(237).

4.2 Abspielen von Tests

Um einen oder mehrere Test abzuspielen, markieren Sie den oder die entsprechenden Knoten und drücken Sie Return oder den Start Knopf  oder wählen Sie Wiedergabe→Start. QF-Test hebt Knoten, die gerade ausgeführt werden, durch einen kleinen Pfeil hervor. Außerdem werden Meldungen über den Fortschritt des Tests in der Statuszeile ausgegeben. Beides kann die Geschwindigkeit der Testausführung leicht verringern und ist über die Optionen Aktive Schritte markieren(531) und Aktive Schritte in der Statuszeile anzeigen(531) abschaltbar.

Nach dem Ende des Tests wird das Ergebnis in der Statuszeile angezeigt. Im Idealfall steht dort ”Wiedergabe beendet - keine Fehler”. Andernfalls wird die Zahl der Warnungen, Fehler und Exceptions angezeigt. Im Fehlerfall wird zusätzlich ein entsprechender Meldungsdialog geöffnet.

Auch das Abspielen von Tests wird von zahlreichen Optionen gesteuert. Einige dienen lediglich zur Anpassung an persönliche Vorlieben, während andere starken Einfluss auf den Ausgang von Tests haben. Bitte lesen Sie Abschnitt 41.3(530), um sich mit diesen Optionen vertraut zu machen.

Um einen Testlauf vorzeitig Abzubrechen, drücken Sie den Stop Button  oder wählen Sie Wiedergabe→Stop. Sie können den Test auch mittels des Pause Buttons oder Wiedergabe→Pause vorübergehend unterbrechen, wodurch auch der Debugger aktiviert wird (vgl. Problemanalyse und Debugging(137)). Zum Fortsetzen drücken Sie erneut den Pause Button.

Wenn ein Test in vollem Tempo abläuft, kann es schwierig werden ihn anzuhalten, insbesondere wenn der Mauszeiger über den Bildschirm mitbewegt wird oder die Fenster des SUT ständig nach vorne gebracht werden. Mit Hilfe der Hotkey für Wiedergabe unterbrechen (”Keine Panik”-Taste)(530) (Standard ist Alt-F12) können Sie die Kontrol-

le wiedererlangen. Alle laufenden Tests werden sofort unterbrochen. Zum Fortsetzen drücken Sie die selbe Kombination ein weiteres mal.

Beim Zusammenstellen einer Testsuite müssen immer wieder Teile von Tests ausgeführt werden, um das SUT in den richtigen Zustand für die nächste Aufnahme zu bringen. Dabei können andere Teile eher hinderlich sein. Ebenso gibt es oft langwierige Sequenzen, die beim Ausprobieren eher im Weg sind. Mittels Bearbeiten→Ein-/Ausschalten können Sie solche Knoten deaktivieren und später für den eigentlichen Testlauf wieder einschalten.

Den aktuellen Fehlerstatus während des Ablaufs eines Tests wie auch das abschließende Resultat zeigt QF-Test in der Statusleiste am unteren Rand des Hauptfensters an. Letztere kann über das Menü Ansicht→Statuszeile anzeigen ein- bzw. ausgeblendet werden. Bei der Wiedergabe von Testfallsatz(606) oder Testfall(599) Knoten, deren Bedeutung in Abschnitt 8.2(153) erklärt wird, enthält die Statuszeile auch relevante Ergebniswerte aus folgender Liste:

Zählersymbol	Beschreibung
#	Anzahl der Testfälle insgesamt. Diese Zählerwert beginnt mit einem '>'-Zeichen im Fall von übersprungenen Testfallsätzen.
!	Anzahl Testfälle mit Exceptions.
−	Anzahl Testfälle mit Fehlern.
−	Anzahl Testfälle mit erwarteten Fehlern. Fehlschlagen erwartet wenn...(604) markiert einen Testfall als erwartet fehlerhaft.
+	Anzahl erfolgreicher Testfälle.
»	Anzahl übersprungene Testfälle. Ein Testfall wird übersprungen, wenn seine (optionale) Bedingung(603) fehlschlägt. Diese Zählerwert beginnt mit einem '>'-Zeichen im Fall von übersprungenen Testfallsätzen.
»	Anzahl übersprungene Testfallsätze. Ein Testfallsatz wird übersprungen, wenn seine (optionale) Bedingung(611) fehlschlägt.
⊘	Anzahl nicht implementierter Testfälle. Ein Testfall gilt als nicht implementiert, wenn er keine Knoten enthält, die während des Testlaufs aufgeführt wurden.
▶	Anzahl ausgeführter Testfälle.
%	Prozent Testfälle erfolgreich.

Tabelle 4.1: Testresultate in der Statusleiste

Die finalen Zählerstände finden sich auch im Report wieder, der für jeden Testlauf generiert werden kann. Reports sind Thema von Kapitel 24(330).

Hinweis

Die Zählerwerte von oben stehen auch als Variablen(127) während dem Testlauf zur Verfügung. Ein TestRunListener(1223) kann hilfreich sein, um die Zählerwerte während des Testlaufs auszuwerten und entsprechende Aktionen anzustoßen.

4.3 Aufnehmen von Checks

Auch wenn es ganz unterhaltsam sein kann, Sequenzen aufzuzeichnen und dem Spiel der Fenster des SUT beim Wiederabspielen zuzusehen, geht es doch eigentlich darum herauszufinden, ob sich das SUT dabei auch korrekt verhält. Diese Aufgabe übernehmen Checks[805]. Der Check Text[806] Knoten zum Beispiel, liest den Text aus einer Komponente aus und vergleicht ihn mit einem vorgegebenen Wert. Stimmen diese nicht überein, wird ein Fehler signalisiert.

Eine Aufstellung aller möglichen Checks und ihrer Funktionsweisen findet sich im Referenzteil[805]. Die Palette reicht vom einfachen Text-Check bis zum mächtigen Check Abbild[828] mit Suchalgorithmen bis hin zur Möglichkeit eigene Check-Typen[1208] zu implementieren. Im Weiteren werden wir uns darauf konzentrieren, wie man Checks am einfachsten erstellt, nämlich indem man sie aufnimmt.

Während der Aufnahme befindet sich das SUT im *Aufnahmemodus*, in dem alle Events gesammelt und an QF-Test geschickt werden. Über den Check Button ✓ oder mittels Aufnahme→Check schalten Sie das SUT in den *Checkmodus*, zu erkennen am geänderten Mauszeiger. In diesem Modus werden Events nicht weiter aufgezeichnet. Stattdessen wird die Komponente unter dem Mauszeiger hervorgehoben. Wenn Sie auf eine Komponente klicken, wird ein Check für diese aufgezeichnet. Der aktuelle Wert der Komponente wird dabei als Maßgabe verwendet. Um zurück in den Aufnahmemodus zu gelangen, verwenden Sie ebenfalls den Check Button oder den Menüeintrag.

Es gibt verschiedene Arten von Checks[805], die man für eine Komponente vornehmen kann. Welche davon in Frage kommen, hängt von der jeweiligen Komponente ab. So stellen manche Komponenten keinen Text dar, so dass z.B. ein Check Text[806] Knoten für einen Scrollbar keinen Sinn ergibt. Wenn Sie im Checkmodus mit der rechten Maustaste auf eine Komponente klicken, erhalten Sie ein Menü mit allen für diese Komponente zur Verfügung stehenden Arten von Checks. In diesem können Sie den gewünschten Check auswählen. Ein Klick mit der linken Maustaste zeichnet dagegen direkt den ersten Check aus dieser Liste auf.

Wenn Sie die Shift oder Strg Taste gedrückt halten, während Sie mit der rechten Maustaste klicken, bleibt das Auswahlmenü für die Checks auch nach einer Selektion offen. Dadurch können Sie sehr einfach mehrere verschiedene Checks für die selbe Komponente aufnehmen.

Checks und Events lassen sich sehr gut mischen und Sie werden bald einen Aufnahmestil ala *klick, klick, tipp, check, klick, klick, check...* entwickeln. Dabei ständig zwischen dem SUT und QF-Test hin- und herzuschalten um den Checkmodus zu (de)aktivieren, ist mehr als mühsam. Stattdessen können Sie mittels einer Tastenkombination direkt im SUT zwischen Aufnahmemodus und Checkmodus hin- und herzuschalten. Standardmäßig ist diese Taste F12. Ist diese Taste anderweitig belegt, oder lässt Ihr Windowmanager diese gar nicht erst bis zur Applikation durch, können Sie über die Option

Hotkey für Checks(508) eine beliebige andere Tastenkombination dafür wählen. Um eine Sequenz mit eingestreuten Checks aufzunehmen, müssen Sie nun nur noch die Aufnahme starten, zum SUT wechseln und die Sequenz aufnehmen. Wann immer Sie einen Check einbauen wollen, drücken Sie F12 (oder was immer Sie konfiguriert haben) und nehmen Sie den Check (oder die Checks) auf. Schalten Sie dann mit F12 wieder zurück, um mit der Aufnahme fortzufahren. Auf diese Weise können Sie komplett im SUT arbeiten und müssen lediglich zu QF-Test zurückschalten, um die Aufnahme zu beenden.

Achtung: Lassen Sie sich durch diese Annehmlichkeiten nicht dazu verführen, allzu lange Sequenzen aufzunehmen. Wenn sich am SUT etwas ändert, das dazu führt dass eine solche Sequenz nicht mehr durchläuft, kann es sehr mühsam werden, herauszufinden was schiefgegangen ist und die Sequenz entsprechend anzupassen.

4.4 Daten aus der GUI auslesen

Oft ist es notwendig, einen Wert aus der Oberfläche des SUT auszulesen, um diese als Eingabe für den Test zu verwenden.

QF-Test bietet für diese Aufgabe einen speziellen Satz von Abfrageknoten(839), verfügbar unter Einfügen→Verschieden Knoten:

- Text auslesen(839) zum Auslesen des Komponenten- oder Elementtextes,
- Index auslesen(843) zum Auslesen des Elementindex,
- Geometrie auslesen(846) zum Auslesen des Komponenten- oder Elementgeometrie.

Die ermittelten Werte werden lokalen oder globalen Variablen zugewiesen, die im Abfragekonten deklariert werden können.

Anstatt einen Abfrageknoten per Hand einzufügen, kann ein solcher leichter erzeugt werden, indem man erst einen Mausklick-Knoten auf die gewünschte Komponente aufzeichnet und diesen dann mit Hilfe der Konvertierungsoperation(19) in den gewünschten Abfrageknoten umwandelt.

4.5 Komponenten aufnehmen

Wie bereits erwähnt wird Komponenteninformation automatisch mitgespeichert wenn Events oder Checks aufgenommen werden. Trotzdem gibt es Situationen in denen die Aufnahme von Komponenten alleine hilfreich ist.

Um den *Modus zum Komponentenaufnehmen* zu starten, braucht nur der Komponentenaufnahme-Knopf  gedrückt oder Aufnahme→Komponenten aufnehmen gewählt werden. Bei dem anschließenden Wechsel zum Fenster des SUT zeigt sich dort ein besonderes Verhalten, in welchem die Komponente unter dem Mauszeiger hervorgehoben wird.

Ein Klick mit der linken Maustaste auf eine Komponente zeichnet diese auf, während bei einem Klick mit der rechten Maustaste ein Menü mit weiteren Möglichkeiten geöffnet wird, z.B. um eine Komponente mit seinen Kinder oder auch alle Komponenten eines Fensters aufzunehmen. In dieser Weise können mehrere Komponenten erfasst werden. Als nächstes wechselt man zurück zu QF-Test und deaktiviert den Komponentenaufnahme-Knopf  oder den Aufnahme→Komponenten aufnehmen Menüeintrag. Nun werden die gespeicherten Komponenteninformationen in Form von Komponente(930) Knoten unter dem Fenster und Komponenten(942) Knoten abgelegt.

Die Aufnahme von Komponenten kann alternativ mit einer konfigurierbaren Tastenkombination gesteuert werden. Durch Drücken von Shift-F11 (Standardeinstellung) *im Fenster des SUTs* wird die Komponentenaufnahme aktiviert. Weitere Informationen finden Sie in der Dokumentation zur Option Hotkey für Komponenten(516).

Hinweis

Nur jeweils eine Testsuite kann die aufgezeichneten Komponenten in Empfang nehmen. Haben Sie mehrere Testsuiten in individuellen Fenster geöffnet, wenn also die Workbench-Ansicht deaktiviert ist, so erhält die Suite die Komponenten, in der die Komponentenaufnahme (mittels Knopf oder Menü) gestoppt wird oder - falls Shift-F11 verwendet wurde, die Suite, die über den Menüeintrag Aufnahme→Suite ist Empfänger für Aufnahmen festgelegt werden kann.

Das Feature der Komponentenaufnahme kann auch dazu verwendet werden, schnell eine Komponente aufzufinden, egal ob sie schon aufgezeichnet wurde oder nicht. Beim Anlegen von Event- oder Checkknoten von Hand oder beim Ändern der Zielkomponente, muss diese im Attribut QF-Test ID der Komponente(776) angegeben werden. Wird eine Komponente aufgezeichnet, steht ihre QF-Test ID anschließend im Clipboard und kann mittels Strg-V direkt in das Feld für die QF-Test ID der Komponente eingefügt werden. Außerdem können Sie mittels Strg-Umschalt-Backspace oder über den Menüeintrag Bearbeiten→Nächsten Knoten anwählen oder den entsprechenden Button in der Werkzeugleiste direkt zum aufgenommenen Komponente Knoten springen.

Das Popup-Menü, das erscheint, wenn man mit der rechten Maustaste auf eine Komponente klickt, enthält auch einen Eintrag Im Inspektor anzeigen, der es erlaubt, Komponenten zu untersuchen, sowie einen Eintrag Methoden anzeigen, mit dem man eine Übersicht der Methoden des GUI-Elements erhält, (siehe Abschnitt 5.12(107)).

Komponenten spielen in der Struktur einer Testsuite eine zentrale Rolle, auf die in

Kapitel 5(47) näher eingegangen wird.

4.6 HTTP-Requests aufnehmen (GET/POST)

Web

Um einen vom SUT gesendeten (GET/POST) Request aufnehmen zu können, muss das SUT aus QF-Test heraus gestartet worden sein (vgl.Kapitel 46(1000)) und die Verbindung zum SUT muss bestehen.

Während der Aufnahme befindet sich das SUT im *Aufnahmemodus*, in dem alle Events gesammelt und an QF-Test geschickt werden. Über Aufnahme→HTTP-Requests aufnehmen schalten Sie das SUT in den *Request Aufnahme Modus*. In diesem speziellen Aufnahme-Modus werden im Gegensatz zum bereits beschriebenen Aufnehmen von Testsequenzen(39) alle vom Webbrowser gesendeten GET bzw. POST Request als spezielle Knoten gespeichert. Um zurück in den Aufnahmemodus zu gelangen, verwenden Sie ebenfalls den Menüeintrag.

Im Abschnitt Web-Optionen(566) wird beschrieben wie der Typ des aufgenommenen Requests beeinflusst werden kann. Standardmäßig wird ein Browser-HTTP-Request(915) erzeugt. Dieser eignet sich zum Automatisieren größerer Formulareingaben, da keine separaten Eingabeknoten verwendet werden müssen. Die Formulardaten werden direkt im Browser abgesendet und der erhaltene Response dargestellt. An dieser Stelle kann der Testablauf direkt im Browser weitergeführt werden. Der Server-HTTP-Request(910) wird hingegen direkt durch QF-Test abgesendet. Der Response steht nur in QF-Test zur Verfügung und beeinflusst den Browser nicht.

Die Attribute eines entsprechend aufgezeichneten HTTP-Request Knotens oder auch die Parametrisierung von Requests sind ausführlich im Abschnitt HTTP-Requests(910) des Referenzteils erläutert.

Kapitel 5

Komponenten

Auch wenn man sie oft gar nicht bemerkt - zumindest bis die erste `ComponentNotFoundException`(958) auftritt - sind die Komponenten(930) das Herz einer Testsuite, denn die stabile Erkennung von Komponenten ist die zentrale Herausforderung an ein gutes GUI Testtool. Um das meiste kümmert sich QF-Test automatisch selbst, jedoch erfordern manche Situationen manuelle Definitionen oder Eingriffe. Deshalb ist das Verständnis von Komponenten und deren Abbildung und Behandlung in QF-Test sehr wichtig. Die Grundlagen sollen in diesem Kapitel erläutert werden.

Videos

Video

Das Video

'Komponentenerkennung'
https://www.qftest.com/de/yt/komponentenerkennung.html

erläutert zunächst die Wiedererkennungskriterien für Komponenten. Danach (ab Minute 13:07) werden generische Komponenten erläutert, zuerst solche mit regulären Ausdrücken, danach solche mit Variablen für die Wiedererkennungsmerkmale.

Es gibt zwei Videos, die die Behandlung einer `ComponentNotFoundException` ausführlich erklären:

-  'ComponentNotFoundException - einfacher Fall'
  https://www.qftest.com/de/yt/componentnotfoundexception-einfach-40.html
-  'ComponentNotFoundException - komplexer Fall'
  https://www.qftest.com/de/yt/componentnotfoundexception-komplex-40.html

Das Video

’Die Explosion der Komplexität in der Web Testautomatisierung eindämmen’
https://www.qftest.com/de/yt/web-testautomatisierung-40.html

zeigt eindrucksvoll den Umgang von QF-Test mit tief geschachtelten DOM-Strukturen.

Video-Mitschnitt des Spezialwebinars

’Komponentenerkennung’
https://www.qftest.com/de/yt/komponentenerkennung-51.html

.

Aktionen im GUI und Komponenten Aktionen, die der Anwender auf den Komponenten eines GUI[49] ausführt, werden von QF-Test in *Events* umgewandelt. Jeder Event hat eine Zielkomponente. Für einen Mausklick ist das die Komponente unter dem Mauszeiger, für einen Tastendruck die Komponente, die den Tastaturfokus (*keyboard focus*) besitzt. Wenn QF-Test einen Event aufzeichnet, nimmt es zusätzlich Informationen über die Zielkomponente auf, so dass der Event später wieder für die gleiche Komponente abgespielt werden kann.

Wiedererkennung Die Wiedererkennung von Komponenten ist einer der komplexesten Teile von QF-Test. Der Grund dafür liegt in der Notwendigkeit, mit einem gewissen Grad an Veränderungen zurechtzukommen. QF-Test ist als Werkzeug für die Durchführung von Regressionstests ausgelegt. Wird eine neue Version des SUT getestet, sollten bestehende Tests idealerweise unverändert durchlaufen. Folglich muss sich QF-Test an ein möglicherweise geändertes GUI anpassen können. Würden z.B. die ”OK” und ”Abbrechen” Knöpfe vom unteren Rand der Detailansicht nach oben verschoben, würde QF-Test Events für diese Knöpfe immer noch korrekt wiedergeben. Das Ausmaß an Änderungen, an die sich QF-Test anpassen kann, hängt von den zur Verfügung stehenden Wiedererkennungskriterien ab. An dieser Stelle kann die Entwicklung häufig mit relativ geringem Aufwand einen großen Beitrag zur Erstellung robuster Regressionstests leisten.

Für die Komponentenerkennung stehen die folgenden Kriterien zur Verfügung:

- Klasse[62], korreliert mit der Funktion der Komponente
- Name[64], basierend auf dem Komponentenbezeichner[66]
- Merkmal[70], ein der Komponente zugeordneter Text
- Weitere Merkmale[73], weitere Erkennungsmerkmale, zum Beispiel Beschriftung oder Tooltip
- Index[77]
- Geometrie[77]
- Komponentenhierarchie[78]

Die Kriterien fließen mit unterschiedlicher Gewichtung in die Wiedererkennung ein. Herausragende Bedeutung haben die Klasse und der Komponentenbezeichner(66). Bei letzteren kann die Entwicklung einen großen Beitrag zur Teststabilität beitragen (siehe Wie erreicht man eine robuste Komponentenerkennung?(54)). Weitere Informationen siehe Gewichtung der Wiedererkennungsmerkmale bei aufgenommenen Komponenten(1015).

Windows-Tests

Bitte beachten Sie bei nativen Windows-Applikationen auch Abschnitt 15.5(239).

Speicherung der Erkennungsinformationen

Die Wiedererkennungsinformationen werden von QF-Test entweder in Komponente-Knoten(78) abgespeichert oder direkt in den Eventknoten als SmartID(81) eingetragen. In Komponente-Knoten versus SmartID(51) erfahren Sie, was für welches Anwendungsgebiet besser geeignet ist.

Standardmäßig zeichnet QF-Test Komponente-Knoten auf.

Unterelemente und geschachtelte Komponenten Es gibt auch Komponenten, die QF-Test relativ zu einer übergeordneten Komponente adressiert. Dazu gehören zum Beispiel Tabellenzellen, Listeneinträge, Baumknoten, Icons in Buttons oder eine Checkbox in einer Tabellenzelle.

Dafür verwendet QF-Test besondere Adressierungsformen. In Unterelemente: Adressierung relativ zur übergeordneten Komponente(92) wird dieses Thema ausführlich behandelt.

Außerdem bietet QF-Test die Möglichkeit Geltungsbereich (Scope)(90) zu definieren, um Aktionen (Mausklick, Eingabe, Check) auf darin enthaltene Komponenten zu beschränken.

5.1 Komponenten eines GUI

Die grafische Oberfläche (*Graphical User Interface*, kurz *GUI*) einer Applikation besteht aus einem oder mehreren Fenstern, die verschiedene Komponenten beinhalten. Diese Komponenten sind ineinander verschachtelt und bilden eine hierarchische Struktur. Komponenten, die andere Komponenten enthalten, heißen Container. Da QF-Test selbst eine komplexe Applikation mit grafischer Oberfläche ist, kann das Hauptfenster gut als Beispiel dienen:

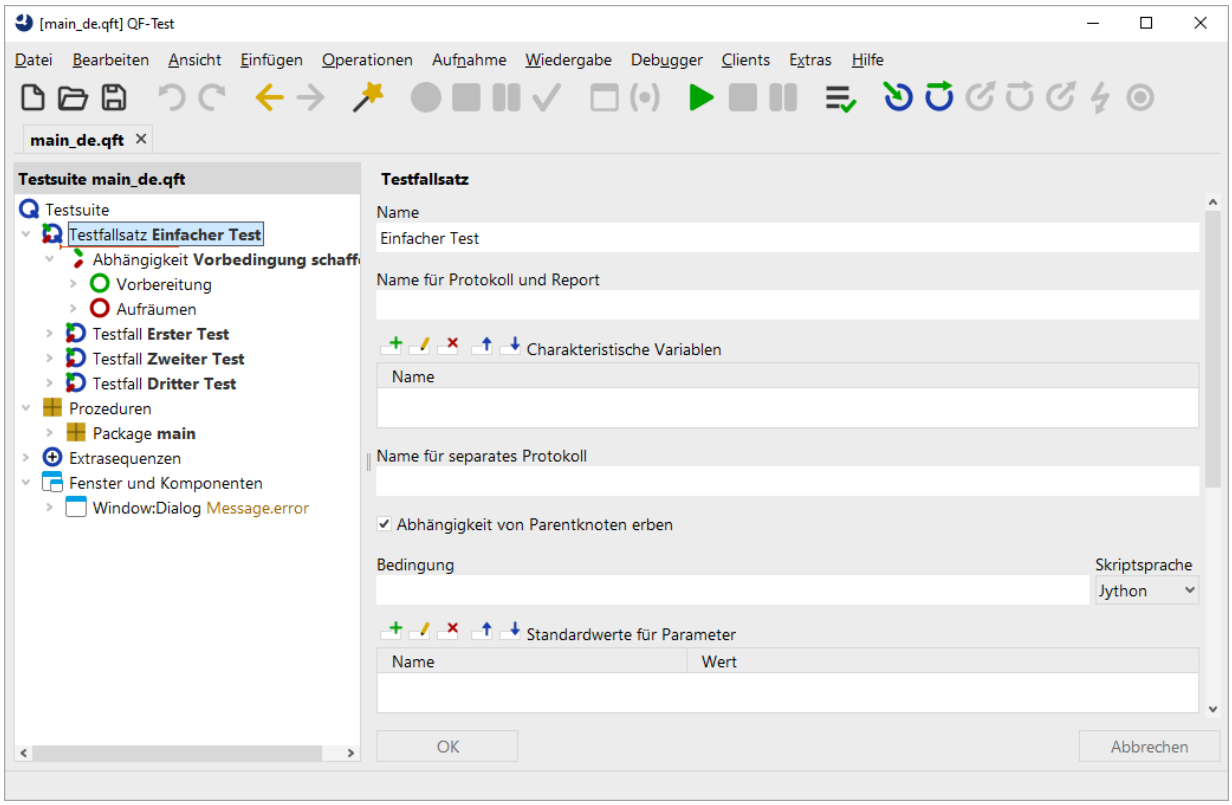


Abbildung 5.1: Komponenten eines GUI

Das Fenster (*window*) enthält eine Menüleiste (*menu bar*), welche wiederum die Menüs (*menu*) für QF-Test enthält. Darunter befindet sich die Werkzeugleiste (*toolbar*) mit den Knöpfen (*toolbar buttons*). Der Hauptteil bedient sich einer *split pane* um die Baumansicht und die Detailansicht zu trennen. Die Baumansicht besteht aus einer Beschriftung (*label*) ”Testsuite” und dem Baum (*tree*). Die Detailansicht enthält selbst wieder eine komplexe Hierarchie von verschiedenen Komponenten wie Textfeldern (*text field*), Knöpfen (*button*), einer Tabelle (*table*), etc. Dabei gibt es noch eine ganze Reihe von Komponenten, welche man nicht auf den ersten Blick erkennt. So steckt der Baum z.B. in einer *scroll pane*, die *scroll bars* anzeigt, wenn der Baum zu groß für seinen Bereich wird. Verschiedene *panes* dienen lediglich als Container und Hintergrund für andere Komponenten, z.B. der Bereich, der die ”OK” und ”Abbrechen” Knöpfe enthält.

Sofern nicht explizit anders angegeben bezieht sich der Begriff ”Komponente” in diesem Handbuch auf die Elemente eines GUI, egal, wie die einzelnen Komponenten in der jeweiligen GUI Technologie genannt werden.

5.2 Komponente-Knoten versus SmartID

Die Wiedererkennungskriterien können auf zwei unterschiedliche Arten mit den Events in den Tests verknüpft werden. Bei der klassischen Methode werden die Wiedererkennungsmerkmale als Attribute eines Komponente(930)-Knotens abgespeichert (siehe auch Komponente-Knoten(78)). In den Tests wird darauf über die QF-Test ID der Komponente referenziert. Alternativ können GUI-Elemente mittels SmartID(81) direkt über die Wiedererkennungskriterien adressiert werden. Komponente-Knoten werden dann keine aufgezeichnet.

SmartIDs und die klassische Methode mit aufgenommenen Komponente-Knoten können alternativ verwendet, im Bedarfsfall aber auch kombiniert, werden. Die folgenden Punkte können Ihnen bei der Überlegung, ob Sie SmartIDs verwenden oder die Komponenten aufnehmen wollen, helfen:

- Verbesserte Lesbarkeit eines Tests(51)
- Testgesteuerte Entwicklung(52)
- Schlüsselwort-basierende Tests(53)
- Wiedererkennungsstabilität(53)
- Wartbarkeit(54)
- Performanz(54)

5.2.1 Verbesserte Lesbarkeit eines Tests

SmartIDs bieten in der folgenden Situation Vorteile gegenüber den aufgenommenen Komponente-Knoten:

Die angesprochenen GUI-Komponenten sollen bei Event und Check-Knoten direkt erkennbar sein. Falls die Komponentenbezeichner(66) der Komponenten kryptisch, aber brauchbare Beschriftungen vorhanden sind, haben SmartIDs Vorteile:

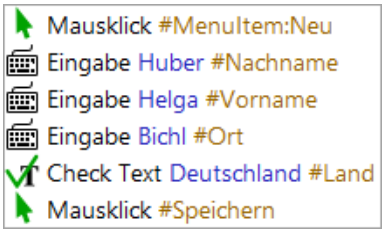


Abbildung 5.2: Lesbarkeit von SmartIDs

Abbildung 5.3: Lesbarkeit von Bezeichnern

Auch bei Prozeduren kann die Lesbarkeit eines Tests verbessert werden, wenn statt kryptischer Bezeichner auf Beschriftung basierende SmartIDs verwendet werden können.

SmartIDs können auch bei Feldern, die die gleichen Wiedererkennungsmerkmale haben, aber in unterschiedlich beschrifteten Panels liegen, die Lesbarkeit erhöhen. Für nachfolgendes Beispiel könnte man zum Beispiel die SmartIDs `#Kundenadresse@#Nachname` und `#Rechnungsadresse@#Nachname` verwenden.

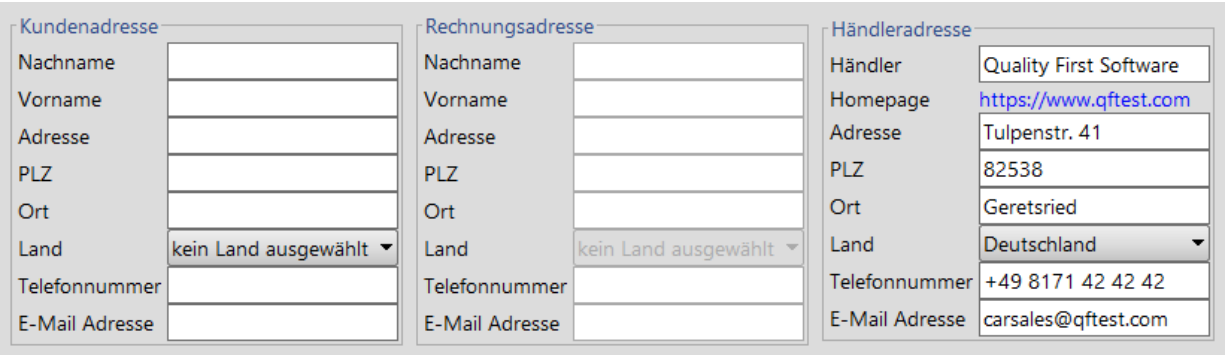


Abbildung 5.4: Lesbarkeit von SmartIDs in Panels mit Beschriftung

5.2.2 Testgesteuerte Entwicklung

Bei testgesteuerter Entwicklung bieten SmartIDs den großen Vorteil, dass keine Komponente-Knoten angelegt werden müssen. Außerdem werden bei testgesteuerter Entwicklung häufig die Komponentenbezeichner(66) im technischen Design festgelegt, die dann für die Testerstellung genutzt werden können. Wenn der Komponentenbezeichner zum Beispiel `btnOK` lautet, kann die Komponente über die SmartID `#btnOK` referenziert werden.

5.2.3 Schlüsselwort-basierende Tests

Schlüsselwort-basierende Tests werden technisch über Prozeduraufrufe und Parameter implementiert. Der Testersteller nimmt somit keine Komponenten auf und ist für die Identifikation der Komponenten auf visuelle Informationen aus dem GUI angewiesen. Dies kann die Beschriftung der Komponente oder deren Funktion (Klasse) sein. In der SmartID können die Erkennungsmöglichkeiten auch kombiniert und mit einem Index versehen werden.

5.2.4 Wiedererkennungsstabilität

Die Stabilität der Wiedererkennung ist bei aufgenommenen Komponenten und SmartIDs gleich gut, wenn für die SmartID der Name, gegebenenfalls in Verbindung mit der Klasse, verwendet wird. Grundsätzlich hängt die Wiedererkennungsstabilität von der Änderungswahrscheinlichkeit der verwendeten Kriterien ab. Wenn zum Beispiel die Beschriftung einer Komponente in einer Applikation über die Versionen hinweg stabil bleibt, wird auch die Wiedererkennung über eine SmartID mittels Beschriftung (Merkmal(70) bzw. `qfs:label*`-Varianten(74)) stabil sein.

Aufgenommene Komponente-Knoten verwenden für die Wiedererkennung einen vorgegebenen Algorithmus, der den einzelnen Wiedererkennungskriterien unterschiedliche Wichtigkeit beimisst. Klasse, Name und Hierarchie haben hierbei Top-Priorität. Wenn kein Name vorhanden ist, wird aus Hierarchie, Beschriftung, Index und Geometrie (in absteigender Wichtigkeit) eine Wahrscheinlichkeit berechnet, an Hand derer entschieden wird, ob es sich bei einem GUI-Element um die gesuchte Komponente handelt.

Dieser Algorithmus hat sich für die meisten Anwendungsfälle als sehr gut erwiesen. Es gibt jedoch auch Fälle, in denen untergeordnete Wiedererkennungskriterien (z.B. die Beschriftung) eine bessere Stabilität bieten als die höher gewichteten Kriterien. Bei aufgenommenen Komponente-Knoten kann man in diesem Fall mittels Resolver eingreifen, siehe Das `resolvers` Modul(1154). Hier kommt jedoch die Stärke der SmartIDs zum tragen, dass gezielt ein stabiles Wiedererkennungskriterium (oder eine Kombination mehrerer Kriterien) angegeben werden kann.

Dies ist zum Beispiel der Fall, wenn die Beschriftung stabiler ist als der Komponentenbezeichner(66).

SmartIDs haben auch Vorteile, wenn die Wahrscheinlichkeit groß ist, dass sich bei Versionswechseln (manchmal sogar bei jedem Programmstart) die Komponentenhierarchie verändert oder sich die Erkennungsmerkmale übergeordneter Komponenten ändern. SmartIDs berücksichtigen die Komponentenhierarchie standardmäßig nicht.

5.2.5 Wartbarkeit

Hinsichtlich Wartbarkeit haben die aufgenommenen Komponente-Knoten die Nase vorn, da die Wiedererkennungskriterien zentral im 'Komponente'-Knoten abgespeichert werden und Änderungen nur an dieser Stelle nachgepflegt werden müssen.

Bei SmartIDs hingegen sind die Wiedererkennungskriterien dezentral hinterlegt. Änderungen können aber auch hier über die mächtige Suchen-Ersetzen-Funktion gut nachgepflegt werden. Bei SmartIDs mit gleichen Wiedererkennungskriterien für unterschiedliche Komponenten muss gegebenenfalls manuell eingegriffen werden.

5.2.6 Performanz

Wenn für SmartIDs die Komponentenbezeichner[66] genutzt werden, können sie performanzmäßig mit aufgenommenen Komponente-Knoten mithalten, da für die erkannten Namen ein Index verwendet wird.

Wenn die SmartID jedoch die Beschriftung (Merkmal[70] bzw. `qfs:label*`-Varianten[74]) oder andere Weitere Merkmale[73] nutzt, reicht die Performanz nicht an die von aufgenommenen Komponente-Knoten heran. Sie kann durch explizite Angabe der Klasse deutlich verbessert werden, da dann die GUI-Elemente bereits danach gefiltert werden, bevor die Merkmale ermittelt werden.

5.2.7 Kombination von Komponente-Knoten und SmartIDs

Aufgenommene Komponente-Knoten können mit SmartIDs kombiniert werden. Details hierzu finden Sie in Unterelemente: Adressierung relativ zur übergeordneten Komponente[92] und QF-Test ID der Komponente als SmartID[90].

Aufgenommene Komponenten können genutzt werden, um die SmartID-Syntax zu überlagern, in dem man ihre QF-Test ID[931] auf eine SmartID inklusive Präfix "#" setzt. Dies ermöglicht einfache, datengetriebene oder im Vorfeld generierte Tests mit SmartID zu erstellen und nur an neuralgischen Punkten einzelne Komponenten spezifischer zu definieren, ohne die Tests oder Prozeduren dafür anpassen zu müssen.

5.3 Wie erreicht man eine robuste Komponentenerkennung?

Die zentrale Funktionalität eines GUI Testwerkzeuges ist die Wiedererkennung der grafischen Komponenten. QF-Test bietet hierfür eine Vielzahl an Konfigurationsmöglich-

keiten. Dieser Abschnitt gibt einen Überblick über die gebräuchlichsten Strategien und Einstellungen, um die Komponentenerkennung so stabil wie möglich zu gestalten.

Hinweis

Sie sollten die Strategie zur Komponentenerkennung **vor** dem breiten Einsatz von QF-Test in Ihrem Projekt festlegen. Andernfalls kann die Wartung von Tests größere Aufwände verursachen.

Die Wiedererkennung von Komponenten im SUT beim Abspielen eines Tests ist sehr komplex. Die Herausforderung ist, dass sich die Oberfläche des SUT schon bei normaler Bedienung ständig verändern kann. Fenster werden geöffnet und geschlossen oder in der Größe variiert, wodurch sich die Position und Größe der darin enthaltenen Komponenten ändert. Menüs und Comboboxen werden auf- und zugeklappt, Komponenten werden hinzugefügt oder entfernt, sichtbar oder unsichtbar, aktiviert oder gesperrt. Darüber hinaus wird sich die zu testende Anwendung selbst im Lauf der Zeit weiterentwickeln, was sich auch in Veränderungen an der Oberfläche widerspiegelt. Auf all diese Änderungen muss QF-Test flexibel reagieren und Komponenten so zuverlässig wie möglich zuordnen können.

Dies gelingt QF-Test bereits in vielen Fällen mit den Standardeinstellungen. QF-Test verwendet einen intelligenten, auf Wahrscheinlichkeiten basierenden Algorithmus um eine möglichst stabile und fehlertolerante Komponentenerkennung zu erreichen. Es wertet die in Für die Komponentenerkennung stehen die folgenden Kriterien zur Verfügung: 5[48] genannten Attribute aus und gewichtet sie. Sind jedoch keine guten Erkennungsattribute vorhanden, tut sich der beste Algorithmus schwer. Für diesen Fall gibt es Konfigurations- und Optimierungsmöglichkeiten, die im Welche Optimierungsmöglichkeiten gibt es?[59] erläutert werden.

Die erste Frage ist, ob die Standardeinstellungen bereits ausreichen, also:

5.3.1 Woran erkennt man eine robuste Komponentenerkennung?

Dieser Abschnitt soll Sie in die Lage versetzen zu beurteilen, ob die aktuelle Komponentenerkennung aller Voraussicht nach robust sein wird.

Wichtige Elemente einer robusten Komponentenerkennung sind:

- Klasse[62] der Komponente
- Name[64]
- Beschriftung (Merkmal[70] oder `qfs:label*`-Varianten[74])
- moderate Hierarchietiefe des Komponentenbaums

In den meisten Fällen sind die Klasse und der Name die robustesten Kriterien für die Wiedererkennung. (In selteneren Fällen ändern sie sich aber von einer Version der

Anwendung zur nächsten. Diesen unschönen Fall betrachten wir in Welche Optimierungsmöglichkeiten gibt es?(59), Punkt 2.) Erfahrungsgemäß ändert sich die Beschriftung der Komponente ebenfalls selten und ist somit auch gut geeignet. Ausführliche Informationen zu allen Erkennungsmerkmalen finden Sie in Wiedererkennungskriterien(62).

Bei der Klasse versucht QF-Test aus der vom Entwickler verwendeten Klasse abzuleiten, welche Funktionalität eine Komponente besitzt. Basierend auf dieser generischen Klasse optimiert QF-Test die Aufnahme und bietet funktionsspezifische Checks (zum Beispiel den Check einer ganzen Zeile für eine Tabelle) an.

Als erstes wollen wir Ihnen zeigen wie Sie an Hand der aufgenommenen Komponente(930) Knoten schnell erkennen, ob generische Klassen erkannt wurden und ob Namen oder Beschriftungen vorhanden sind.

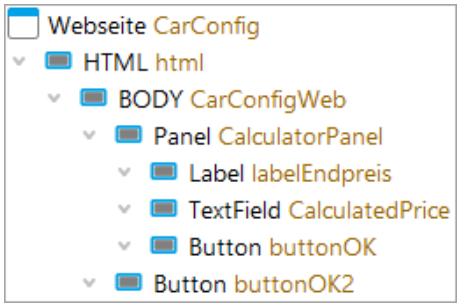


Abbildung 5.5: Komponentenbaum 1

Die Klasse ist der schwarze Text der Komponente-Knoten. Wenn die Klasse mit einem Großbuchstaben, gefolgt von einem Kleinbuchstaben beginnt, handelt es sich im Allgemeinen um eine der Generische Klassen(1329), zum Beispiel `Button`. Wenn die Klasse bei Browser-Elementen nur aus Großbuchstaben besteht, konnte QF-Test die Funktionalität nicht ermitteln. Im Beispiel `HTML` und `BODY`.

Ob Namen oder Beschriftungen vorhanden sind, lässt sich an den braunen Texten ablesen. Hierbei handelt es sich um die QF-Test ID der Komponente, die folgende Rückschlüsse zulässt:

- Wenn die Klasse in der QF-Test ID nicht auftaucht, bedeutet dies, dass entweder ein Name(64) vorhanden ist (im Beispiel `CalculatorPanel` und `CalculatedPrice`) oder, wenn keine generische Klasse erkannt wurde, dass eine Beschriftung (Merkmal(70) oder `qfs:label*`-Varianten(74)) vorhanden ist. Im Beispiel `CarConfigWeb`.
- Wenn die QF-Test ID mit der Klasse beginnt, konnte kein Name(64) ermittelt werden und der darauf folgende Teil ist die Beschriftung der Komponente (Merkmal(70) oder `qfs:label*`-Varianten(74)). Im Beispiel `labelEndpreis` und `buttonOK`.

- Falls weder Name noch Beschriftung gefunden werden, wiederholt die QF-Test ID die Klasse, nur in Kleinbuchstaben. Im Beispiel `html`.
- Falls mehrere Komponenten mit dem beschriebenen Algorithmus die gleiche QF-Test ID erhalten würden, wird eine laufende Zahl angehängt. Im Beispiel "buttonOK2"

Dieser Algorithmus ist auch in Abschnitt 48.2(1017) beschrieben.

Video

Eine gewisse Hierarchie bei den Komponenten ist bei der Wiedererkennung hilfreich. Problematisch sind nur tiefe Verschachtelungen. Für die Komponentenerkennung sind nur wenige Hierarchieebenen tatsächlich relevant. Die anderen können ignoriert werden. Das Video

'Die Explosion der Komplexität in der Web Testautomatisierung eindämmen'
https://www.qftest.com/de/yt/web-testautomatisierung-40.html

zeigt die Problematik von tiefen Verschachtelungen sehr anschaulich - und auch Lösungen dazu. Obiges Beispiel hat nur eine geringe Hierarchietiefe. Dies ist optimal.

Hinweis

Für die Erstellung des Komponentenbaums im obigen Beispiel waren folgende Optionen in der Rubrik Aufnahme→Komponenten gesetzt:

- QF-Test ID des Fensterknotens vor QF-Test ID der Komponente setzen(522) war ausgeschaltet, was der Standardeinstellung entspricht.
- QF-Test ID des Parentknotens vor QF-Test ID der Komponente setzen(522) stand auf `Niemals`, was ebenfalls der Standardeinstellung entspricht.

Die Optionen öffnen Sie über den Menüpunkt Bearbeiten→Optionen

Alternativ zur Auswertung der QF-Test ID im Komponentenbaum können Sie sich über die QF-Test Suche alle Komponenten mit Namen auflisten lassen. Setzen Sie hierzu im Suchdialog `Im Attribut` auf `Name` sowie `Knotentyp` auf `Komponente` und klicken Sie dann `Ergebnisliste anzeigen`.

Wenn Sie eine Aufnahme ausführen, werden die Komponenten, mit denen Sie interagieren, automatisch aufgenommen. Um jedoch alle Komponenten für die Analyse auf einmal aufzunehmen, wählen Sie Aufnahme→Komponenten aufnehmen. Dann im GUI einen Rechtsklick ausführen und `Ganzes Fenster` wählen. (Nach der Analyse sinnvollerweise wieder löschen, um nicht unnötig Ballast herumzuschleppen.)

Hier noch zwei Beispielkomponentenbäume mit Bewertung wie robust die Komponentenerkennung ist.

Beispiel 1

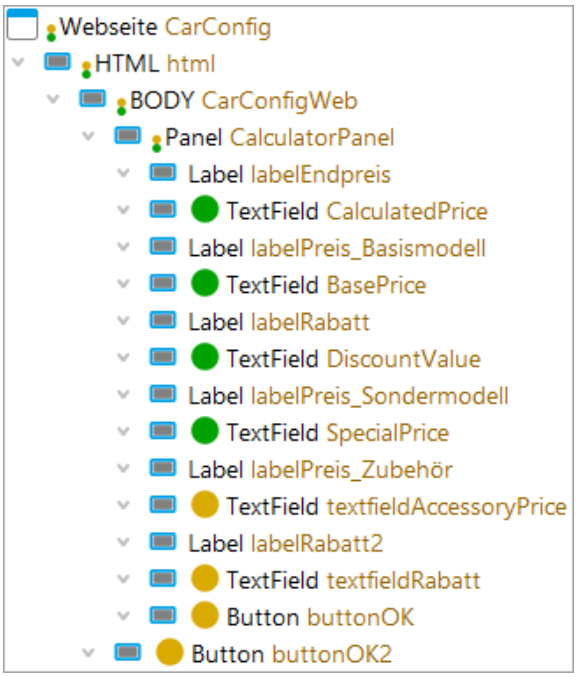


Abbildung 5.6: Stabile Komponentenerkennung - Beispiel 1

Positiv: Generische Klassen[1329] werden erkannt: `MenuBar`, `TabPanel`, `Panel`, `Label` und `TextField`.

Positiv: Für die grün markierten TextFields konnten Namen ermittelt werden, erkennbar daran, dass die QF-Test ID (brauner Text) nicht mit der Klasse beginnt, zum Beispiel `BasePrice`.

Positiv: Für die gelb markierten TextFields und Buttons konnten zwar keine Namen ermittelt werden, erkennbar daran, dass die QF-Test ID (brauner Text) mit der Klasse (`textfield`, `button`) beginnt. Aber der zweite Teil der QF-Test ID zeigt, dass zumindest eine Beschriftung gefunden wurde.

Unwichtig: Die Label haben keine Namen. Sie sind aber selten testrelevant.

Unwichtig: die Container 'HTML' und 'BODY' haben keine generische Klasse. Man könnte sie auf Panel mappen. In diesem Fall würde das aber weder die Wiedererkennung verbessern noch zusätzliche Funktionalität seitens QF-Test (z.B. zusätzliche Checks für die Check-Aufnahme) bringen.

Positiv: Außer `BODY` keine überflüssigen Container.

Beispiel 2

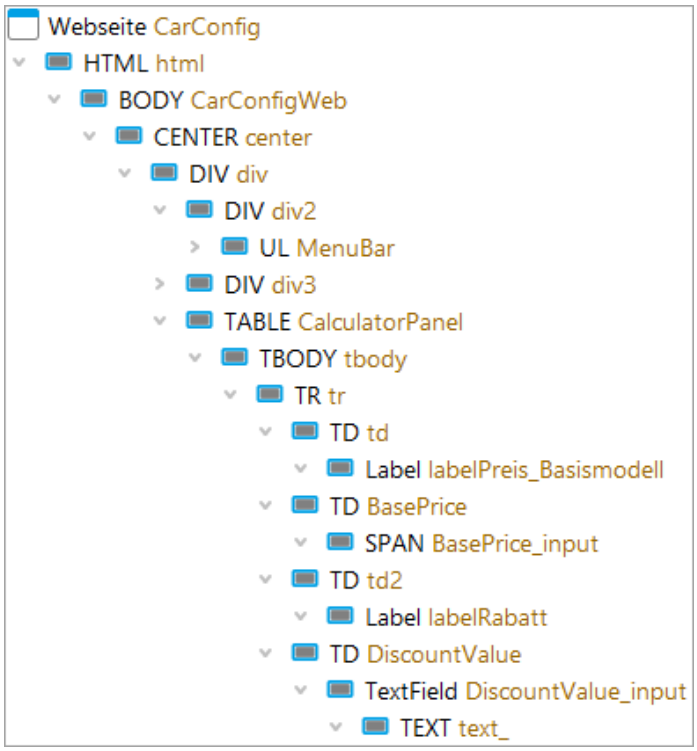


Abbildung 5.7: Stabile Komponentenerkennung - Beispiel 2

Positiv: Für die die testrelevanten Komponenten konnte Name oder Beschriftung ermittelt werden, erkennbar daran, dass die QF-Test ID (brauner Text) nicht mit der Klasse beginnt, zum Beispiel `BasePrice` und `DiscountValue_input`.

Negativ: es werden nur für wenige Komponenten generischen Klassen erkannt. Hier fehlt ein Komponenten-Mapping mit dem CustomWebResolver, siehe Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077).

Negativ: Überflüssige Hierarchieebenen. Die `DIV`, `TR` `TD`, `CENTER` und `TABLE` Komponenten sollten als `Panel` gemappt (siehe Der CustomWebResolver installieren Knoten(1082)) oder ignoriert werden (siehe Der CustomWebResolver installieren Knoten(1082), Konfigurationskategorie `ignoreTags`).

5.3.2 Welche Optimierungsmöglichkeiten gibt es?

Wenn für die relevanten Komponenten generische Klassen und Namen vorhanden sind, können Sie in den allermeisten Fällen davon ausgehen, dass die Komponentenerken-

nung robust ist, und den Rest dieses Abschnitts überspringen.

Wenn es Probleme bei der Wiedererkennung gibt, gilt es zwei grundsätzlich unterschiedliche Fälle zu betrachten:

Wird die Komponente (rechtzeitig) angezeigt?

Dieser Fall hat mit der Komponentenerkennung selbst gar nichts zu tun. Er tritt auf, wenn QF-Test sozusagen zu schnell für die Anwendung ist. In diesem Fall sollten Sie im Testfall explizit auf das Erscheinen der Komponente warten. Weitere Informationen finden Sie in Zeitliche Synchronisierung(101).

Wird die angezeigte Komponente erkannt?

Hier gibt es mehrere Möglichkeiten:

Web: Zuweisung von generischen Klassen zu GUI-Element-Klassen

Bei Web-Anwendungen bitte zuerst die Komponentenzuweisung, wie in Der CustomWebResolver installieren Knoten(1082) beschrieben, durchführen. Wenn dies nicht zu ausreichender Stabilität führt, dann in diesem Abschnitt weiterlesen.

Instabile Komponentenbezeichner

Es wurden Komponentenbezeichner(66) vergeben, diese sind aber über die Anwendungsversionen hinweg nicht stabil. In diesem Fall ist es besser, die Bezeichner mittels Resolver zu entfernen und mit den übrigen Erkennungskriterien zu arbeiten, wenn durch die Entwickler keine stabilen Bezeichner gesetzt werden können.

Bei Web-Anwendungen kann eine entsprechende Einstellung in der Konfigurationskategorie 'customIdAttributes' (siehe Der CustomWebResolver installieren Knoten(1082)) weiterhelfen.

Keine Komponentenbezeichner

Es wurden keine Komponentenbezeichner(66) vergeben und die übrigen Kriterien sind nicht ausreichend stabil. Hier lohnt es sich auch oft, mit der Entwicklung Kontakt aufzunehmen und ihnen die Relevanz von Komponentenbezeichnern für Regressionstests zu erläutern - beziehungsweise denjenigen, der budgetmäßig für Entwicklung und Testung verantwortlich ist, zu überzeugen, dass ein geringer Mehraufwand in der Entwicklung für das Einpflegen der Bezeichner eine deutliche Aufwandsreduzierung in der Testabteilung bedeuten kann.

Wenn dies nicht möglich ist, gibt es manchmal stabile Wiedererkennungskriterien, die QF-Test aber nicht standardmäßig nutzt, die man über einen Namenresolver (siehe Abschnitt 54.1.7(1163)) bekannt geben kann.

Die Komponentenbezeichner enthalten stabile Teile

Nur Teile der Komponentenbezeichner(66) sind stabil. Wenn ein Schema vorliegt, das programmtechnisch ausgewertet werden kann, ist dies auch ein

Fall für einen Namenresolver (siehe Abschnitt 54.1.7(1163)). Bei Web kann dies auch über die Konfigurationskategorie 'autoIdPatterns' (siehe Der CustomWebResolver installieren Knoten(1082)) eingestellt werden.

Für die Komponenten gibt es Beschriftungen, die QF-Test nicht standardmäßig erkennt

Es gibt keinen Namen und der Standardalgorithmus von QF-Test erkennt kein Merkmal und keine `qfs:label*`-Varianten(74), obwohl Kandidaten dafür vorhanden wären. Hier können Sie die Beschriftungen mittels FeatureResolver (siehe Abschnitt 54.1.10(1167)) bzw. ExtraFeaturesResolver (siehe Abschnitt 54.1.11(1168)) bekannt geben.

Web-Komponenten verfügen manchmal über ein Attribut, das als Beschriftung genutzt werden kann. Dies kann im CustomWebResolver in der Konfigurationskategorie `attributesToQftFeature` bekannt gegeben werden (siehe Der CustomWebResolver installieren Knoten(1082)).

Teile des Merkmals oder der `qfs:label*`-Varianten(74) sind stabil

Hier können entweder reguläre Ausdrücke direkt im 'Komponente'-Knoten bzw. in der SmartID genutzt werden. Die Lösung kann aber auch ein FeatureResolver (siehe Abschnitt 54.1.10(1167)) bzw. ExtraFeaturesResolver (siehe Abschnitt 54.1.11(1168))sein.

Übergeordnete Komponenten werden nicht stabil erkannt

Die Komponente an sich ist stabil, aber einer der Container, in denen sie liegt, ist nicht stabil. Hier können reguläre Ausdrücke oder Resolver für die betroffenen Containern Abhilfe schaffen. Wenn alle testrelevanten Komponenten Namen besitzen, kann auch die Option Gewichtung von Namen (Aufnahme)(520) in der Rubrik Aufnahme→Komponenten→Gewichtung von Namen auf "Name übertrifft alles" gesetzt werden.

Hier bietet sich auch der Einsatz von SmartIDs an.

Zusätzliche oder fehlende übergeordnete Komponenten

Die Komponente an sich ist stabil, aber die Hierarchie, in der sie liegt, ist nicht stabil, weil Container verschwinden oder hinzukommen. Wenn alle Komponenten, die testrelevant sind, Namen besitzen, kann auch hier die Option Gewichtung von Namen (Aufnahme)(520) in der Rubrik Aufnahme→Komponenten→Gewichtung von Namen auf "Name übertrifft alles" gesetzt werden.

Alternativ kann im Komponentenbaum die Komponente in der Hierarchie weiter nach oben gebracht werden, so dass die instabilen Container keinen Einfluss mehr haben.

Auch hier bietet sich der Einsatz von SmartIDs an.

Komponentenstruktur/Index

Das Attribut Index(935) spielt eine untergeordnete Rolle, kommt aber zum

Tragen, wenn die Komponentenerkennung ohne Name, Merkmal oder qfs:label*-Varianten(74) auskommen muss. Wenn dann auch noch der Index instabil ist, kann man diesen auch löschen, so dass die Geometrie zum Tragen kommt. In diesem Fall sollte nach dem Start die Größe der zu testenden Applikation immer auf den gleichen Wert (siehe Komponentenevent(790)) gesetzt werden.

5.4 Wiedererkennungskriterien

5.4.1 Klasse

Die Klasse einer Komponente steht für den Typ der jeweiligen Komponente und ist daher sehr wichtig. Vereinfacht gesagt: Wenn QF-Test einen Button aufnimmt, sucht es bei der Wiedergabe auch nur nach einem Button und nicht nach einer Tabelle oder einem Baum. Dies verbessert Performanz und Stabilität der Wiedererkennung und hilft Ihnen außerdem, die von QF-Test aufgezeichnete Information zu einer Komponente mit der tatsächlichen Komponente im GUI zu korrelieren.

Neben ihrer Rolle bei der Identifikation von Komponenten ist die Klasse auch wichtig für die Registrierung verschiedener Arten von Resolvern, die großen Einfluss darauf nehmen können, wie QF-Test Komponenten behandelt. Resolver werden in Abschnitt 54.1.7(1163) detailliert erläutert.

Der Name wird für die Generierung der QF-Test ID der Komponente verwendet. Beispiele hierzu finden Sie in Woran erkennt man eine robuste Komponentenerkennung?(55).

Komponenten können in einer SmartID(81) auch direkt über den Namen adressiert werden, ohne Aufnahme eines Komponente-Knoten(78)s.

Der Einfluss der Klasse auf die QF-Test ID(931) der Komponente ist in Generierung der QF-Test ID der Komponente(1017) beschrieben. Die Verwendung als SmartID in Abschnitt 5.6(81).

Generische Klassen

Normalerweise gibt es in jeder UI-Technologie systemspezifische Klassen, die einen Button, eine Tabelle oder anderen Typ kennzeichnen. Für einen Button kann diese Repräsentation z.B. ein `javax.swing.JButton` in Java Swing, ein `org.eclipse.swt.widgets.Button` in Java SWT, ein `javafx.scene.control.ButtonBase` in JavaFX oder ein `INPUT:SUBMIT` in Web-Anwendungen sein. Um ein technologieübergreifendes Testen zu ermöglichen, vereinheitlicht QF-Test nun diese spezifischen Klassen und arbeitet mit sogenannten

generischen Klassen, z.B. `Button` für alle zuvor genannten Button-Beispiele.

Sie finden eine Beschreibung der generischen Klassen in Kapitel 61(1329). Zusätzlich zu der generischen Klasse werden systemspezifische Klassen als Weitere Merkmale(933) aufgenommen, allerdings mit dem Status ”Ignorieren”. Im Fall von Erkennungsproblemen aufgrund zu vieler ähnlicher Komponenten können diese aktiviert werden, um die Erkennung zu schärfen, wenn auch auf Kosten der Flexibilität.

Swing
JavaFX

Sogar dann, wenn von der Klasse abgeleitet wurde, wird die generische Klasse aufgenommen. Zusätzlich sei erwähnt, dass dieses Konzept QF-Test erlaubt, problemlos Tests mit obfuskierten Klassen zu erstellen, ohne dass Sie die Standardeinstellungen ändern müssen. Während der Wiedergabe vergleicht QF-Test das aufgezeichnete Klasse Attribut der Komponente mit jeder Klasse des Objektes im SUT. Deshalb kann QF-Test auch mit Änderungen von Klassennamen umgehen, solange der Basistyp der selbe bleibt.

Web

HTML ist eine sehr flexible Sprache, um den Inhalt und Aufbau einer Webseite zu beschreiben. Es gibt nur ein Minimum an Quasi-Standards wie zum Beispiel ”INPUT:SUBMIT”, bei denen immer die selbe Funktionalität zu erwarten ist und die somit einer QF-Test Klasse standardmäßig zugewiesen werden können. Die Entwicklung von Web-Applikationen erfolgt im Normalfall mit Toolkits, die ihre eigenen Standards haben. In QF-Test wurden für eine Reihe gängiger Toolkits die Klassenzuordnungen vorgenommen, siehe Besondere Unterstützung für verschiedene Web-Komponentenbibliotheken(1122). Wenn die Anwendung mit einem erweiterten Toolkit erstellt wurde oder ganz mit einem eigenen gearbeitet wurde, wird es notwendig sein, QF-Test die Klassenzuordnungen bekanntzugeben. Dies ist in Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077) beschrieben.

Wenn QF-Test einer Komponente eine generische Klasse zuordnen kann, bietet das für die Testerstellung und Ausführung folgende Vorteile:

Unabhängigkeit von den konkreten technischen Klassen

Mit den generischen Klassen wird eine gewisse Unabhängigkeit von den konkreten technischen Klassen hergestellt. Dieses Konzept ermöglicht es Ihnen, Tests unabhängig von der konkreten Technologie zu erstellen.

Verbesserte Komponentenerkennung

Wenn die Funktionalität der Komponente bekannt ist, können die für die Erkennung am besten geeigneten Kriterien abgespeichert werden.

Beispiel Button: Hier ist die Beschriftung des Buttons erste Wahl für das ’Merkmal’ und das weitere Merkmal ’qfs:labelText’.

Beispiel Textfeld: Bei einem Textfeld hingegen macht es keinen Sinn, den eigenen Text für die Wiedererkennung zu verwenden. Hier sucht QF-Test nach einer Beschriftung in der Nähe und speichert diese im weiteren Merkmal in einer der

qfs:label*-Varianten(74), zum Beispiel in 'qfs:labelLeft', wenn sich die Beschriftung links des Textfelds befindet.

Auch die generische Klasse an sich ist ein Unterscheidungskriterium. Dies wird besonders bei Web-Anwendungen deutlich, bei denen es vorkommen kann, dass die meisten Komponenten mit der Klasse DIV, entsprechend ihres HTML-Tags, aufgenommen werden.

Optimale Mausposition bei der Wiedergabe

Die generische Klasse hat auch Einfluss auf die optimale Mausposition beim Abspielen der Events.

Beispiel Button: Ein Mausklick wird am besten mittig auf den Button abgespielt.

Beispiel Textfeld: Der Mausklick sollte die gleiche Stelle abgespielt werden, auf die der Tester bei der Aufnahme geklickt hat, damit bei Bedarf anschließend Text genau dort eingefügt werden kann.

Klassenspezifische Checks

Außerdem bietet QF-Test bei der Checkaufnahme klassenspezifische Checks an. Bei Textfeldern kann zum Beispiel zusätzlich geprüft werden, ob sie editierbar sind. Check Elemente(818) ist hingegen nur bei Listen, Tabellen oder Bäume sinnvoll.

5.4.2 Name

Falls die Entwickler für eine Komponente einen Komponentenbezeichner(66) vergeben haben, erkennt QF-Test dies und verwendet diesen, wenn geeignet, für das Attribut Name(932).

Wenn ein Wert für Name(932) gefunden wurde, wird dieser auch für die Generierung der QF-Test ID(931) der Komponente verwendet. Weitere Informationen finden Sie in Abschnitt 48.2(1017), Beispiele hierzu in Wie erreicht man eine robuste Komponentenerkennung?(54).

Auch bei der Aufnahme von SmartID(81)s ist der Name im Standard die erste Wahl.

Der Grund für den gewaltigen Einfluss eines guten Komponentenbezeichner(66)s ist die Tatsache, dass dieser die Wiedererkennung von Komponenten über lange Zeit und viele Änderungen hinweg zuverlässig ermöglicht. Eine Komponente zu finden, die einen eindeutigen Bezeichner besitzt, ist offensichtlich trivial. Ohne diese Hilfe verwendet QF-Test viele verschiedene Arten von Informationen, um eine Komponente zu lokalisieren. Der Algorithmus ist fehlertolerant und wurde fein abgestimmt, so dass er ausgezeichnete Ergebnisse erzielt. Und dennoch: Mit Ausnahme des Bezeichners kann sich jede Information zu einer Komponente ändern, wenn sich das SUT weiterentwickelt. Irgendwann, wenn größere Änderungen vorgenommen wurden oder sich mehrere kleinere Änderungen summiert haben, wird eine Komponente nicht mehr erkannt werden und ein manueller Eingriff vonnöten sein, um die Testsuite anzupassen.

Ein weiterer wichtiger Aspekt von Bezeichnern ist, dass sie das Testen von Applikationen, deren Oberfläche in verschiedene Sprachen übersetzt wird, von der aktuell eingestellten Sprache unabhängig macht, da der Bezeichner nur intern vergeben wird und nie übersetzt werden muss.

Die Automatisierung von Tests kann deutlich verbessert werden, wenn die Entwickler des SUT vorausgeplant haben oder bereit sind, durch Vergabe von Bezeichnern für zumindest einen Teil der Komponenten des SUT mitzuhelfen. Außerdem haben Bezeichner in der Testsuite eine sehr hohe Sichtbarkeit, da sie als Basis für die QF-Test ID Attribute der Komponenten dienen. Letzteres sollte nicht unterschätzt werden, speziell für Komponenten ohne besondere Merkmale. Knoten, die Text in Felder namens ”textName”, ”textAddress” oder ”textAccount” einfügen, sind wesentlich leichter zu verstehen und zu warten als solche für die Felder ”text”, ”text2” oder ”text3”. Der koordinierte Einsatz von Bezeichnern ist einer der entscheidenden Faktoren für die Effizienz der Automatisierung und für die mit QF-Test erzielbaren Einsparungen. Sollte sich die Entwicklungsabteilung oder die Projektleitung trotz des minimalen Aufwands nicht mit der Vergabe von Bezeichnern anfreunden können, geben Sie ihnen einfach dieses Kapitel des Handbuchs zu lesen.

Falls Entwickler ein anderes, konsistentes Schema zur Vergabe von Bezeichnern eingesetzt haben, das QF-Test jedoch standardmäßig nicht erkennt, werfen Sie bitte einen Blick in Beeinflussen des Name-Attributs mittels NameResolver(69).

Wenn eindeutige Namen ermittelt werden, können die Optionen Gewichtung von Namen (Wiedergabe)(548) und Gewichtung von Namen (Aufnahme)(520) auf ”Name übertrifft alles” gesetzt werden, wodurch die Komponentenerkennung von der Komponentenhierarchie unabhängig und auf Grund des Name-Cachings maximale Performanz erreicht wird.

Um die Vergabe von Bezeichnern zu vereinfachen, bietet QF-Test eine Funktion, um Bezeichner für die Komponenten vorzuschlagen, für die ein Bezeichner das Testen verbessert. Näheres dazu finden Sie bei der Option Hotkey für Komponenten(516).

Hinweis

Änderungen von Bezeichnern in der zu testenden Applikation sollten möglichst vermieden werden, da dies die Komponentenerkennung aushebelt und eine Menge Nacharbeit in den Tests bedeuten kann. Bitte beachten Sie, wenn dennoch Änderungen vorkommen werden, dass diese im Name(932) Attribut der Komponente nachgezogen werden und nicht im QF-Test ID(931) Attribut, das nur der Referenzierung der Komponente in den Tests dient! Eine weitere mögliche Schwierigkeit kann sein, dass die Namensänderung direkt im Test in der Referenz auf die Komponente erfolgt, zum Beispiel bei einem Mausklick im Attribut QF-Test ID der Komponente(776). Der Test schlägt dann mit einer `UnresolvedComponentIdException`(966) fehl.

Komponentenbezeichner

Komponentenbezeichner werden in den unterschiedlichen UI-Technologien unterschiedlich genannt. Im Handbuch wird für sie auch der Begriff 'Name' verwendet. Außerdem sind die Kriterien, ob und wie die Bezeichner in das Attribut 'Name' übernommen werden, je nach Technologie leicht unterschiedlich.

Die folgenden Ausführungen gelten für die Standardeinstellungen der Optionen, insbesondere von Gewichtung von Namen (Wiedergabe)[548] und Gewichtung von Namen (Aufnahme)[520] (Standardwert: "Hierarchie von Namen"). Auch die Nutzung von Resolvern kann das beschriebene Verhalten verändern.

Java Swing/AWT

Der Komponentenbezeichner heißt hier ebenfalls 'Name'. Falls er gesetzt wurde, wird er in das Attribut 'Name' übernommen. Wenn innerhalb eines Containers Komponenten gleiche Bezeichner haben, legt QF-Test das Weiteres Merkmal[933] `qfs:matchindex` mit einem entsprechenden Index für die Duplikate an.

Alle AWT und Swing Komponenten sind von der AWT Klasse `Component` abgeleitet. Deren `setName` Methode ist daher der Standard für Swing SUTs. Dank dieses Standards machen viele Entwickler auch unabhängig von Plänen für eine Testautomatisierung davon Gebrauch, was eine große Hilfe ist.

JavaFX

Der Komponentenbezeichner heißt hier 'ID'. Falls er gesetzt wurde, wird er in das Name Attribut übernommen. Wenn innerhalb eines Containers Komponenten gleiche IDs haben, legt QF-Test das Weiteres Merkmal[933] `qfs:matchindex` mit einem entsprechenden Index für die Duplikate an.

Für JavaFx wird `setId` verwendet, um Namen für Komponenten (hier "Node" genannt) zu vergeben. Alternativ können IDs mittels FXML über das Attribut `fx:id` gesetzt werden. Obwohl IDs von Knoten eindeutig sein sollen, wird dies nicht erzwungen.

Java SWT

Der Komponentenbezeichner heißt hier ebenfalls 'Name'. Falls er gesetzt wurde, wird er in das Name Attribut übernommen. Wenn innerhalb eines Containers Komponenten gleiche Bezeichner haben, legt QF-Test das Weiteres Merkmal[933] `qfs:matchindex` mit einem entsprechenden Index für die Duplikate an.

Leider hat SWT kein eigenes Konzept für die Vergabe von Namen. Eine akzeptierte Konvention ist der Gebrauch der Methode `setData(String key, Object value)` mit dem String `"name"` als Schlüssel und dem gewünschten Namen als Wert. QF-Test wertet diese Daten - falls vorhanden - aus und verwendet sie als Namen für die Komponenten. Mangels eines echten Standards für die Vergabe von Namen kommen diese in SWT-Anwendungen inklusive

Eclipse nur in Ausnahmefällen zum Einsatz.
Glücklicherweise kann QF-Test Namen für die Hauptkomponenten von Eclipse/RCP-Anwendungen aus den zugrunde liegenden Modellen ableiten, was zu guten Ergebnissen führt, vorausgesetzt es wurden IDs für diese Modelle vergeben. Weitere Details finden Sie bei der Beschreibung der Option Automatische Namen für Komponenten in Eclipse/RCP-Anwendungen[521].

Web

Der naheliegende Kandidat für den Bezeichner eines DOM-Knotens ist sein 'id' Attribut, nicht zu verwechseln mit dem QF-Test ID Attribut von QF-Test's Komponente Knoten. Leider erzwingt der HTML-Standard keine Eindeutigkeit von IDs. Außerdem sind 'id' Attribute ein zweischneidiges Schwert, da sie eine wichtige Rolle bei den internen JavaScript Operationen einer Web-Anwendung spielen können. Daher ist es zwar nicht unwahrscheinlich, dass 'id'-Attribute vergeben wurden, sie können aber nicht so frei definiert werden wie Bezeichner in anderen Technologien. Zu allem Übel müssen viele DHTML und Ajax Frameworks IDs automatisch generieren, was diese als Namen ungeeignet macht.

Glücklicherweise können Komponentenbezeichner über unterschiedliche Attribute des GUI-Elements realisiert werden. Meist ist es das Attribut 'id', manchmal auch 'name' - aber auch andere Attribute können genutzt werden.

Die Option ID-Attribut als Name verwenden[567] bestimmt, ob QF-Test 'id'-Attribute als Namen verwendet oder nicht. Bitte beachten Sie, dass die Option Alle Ziffern aus 'ID'-Attributen eliminieren[568] bewirken kann, dass ursprünglich eindeutige Bezeichner nach Löschung der Ziffern nicht mehr eindeutig sind. Bei der Prüfung, ob der ermittelte Name eindeutig genug ist, wird die Hierarchie, in der die Komponente liegt, für die Bestimmung der Eindeutigkeit mit herangezogen, wenn für die Optionen Gewichtung von Namen (Wiedergabe)[548] und Gewichtung von Namen (Aufnahme)[520] der Standardwert "Hierarchie von Namen" gesetzt.

Die automatisch generierten 'id'-Attribute enthalten manchmal einen statischen Teil, der als Bezeichner genutzt werden kann. Dies kann über die Konfigurationskategorie `autoIdPatterns`, siehe Der CustomWebResolver installieren Knoten[1082], konfiguriert werden. Außerdem kann in dieser Prozedur über die Kategorie `customIdAttributes` auch jedes andere HTML-Attribut als Komponentenbezeichner genutzt werden.

Im Falle einer Webapplikation, welche ein von QF-Test unterstütztes Komponentenbibliothek verwendet, können Sie einen Blick in Abschnitt 51.2.2[1125] werfen, um mehr über das Setzen eindeutiger Bezeichner für das jeweilige Toolkit zu erfahren.

Win

Der Komponentenbezeichner heißt hier 'AutomationId'. Falls er gesetzt wurde, wird er in das Name Attribut übernommen. Wenn innerhalb eines Containers

Komponenten gleiche Bezeichner haben, legt QF-Test ein Weiteres Merkmal(933) mit dem Namen `qfs:matchindex` und einem entsprechenden Index für die Duplikate an.

Android

Der Komponentenbezeichner heißt hier 'ID'. Er wird nur dann in das Name Attribut übernommen, wenn es sich nicht um einen trivialen Klassennamen (vgl. Android - Liste der trivialen Komponentenbezeichner(1019)) handelt. Wenn innerhalb eines Containers Komponenten gleiche Bezeichner haben, legt QF-Test ein Weiteres Merkmal(933) mit dem Namen `qfs:matchindex` und einem entsprechenden Wert für die Duplikate an.

Über die Vergabe von Bezeichnern

Es gibt eine kritische Anforderung für Bezeichner: Sie dürfen sich nicht im Lauf der Zeit ändern, nicht von einer Version der SUT zur nächsten, nicht von einem Aufruf zum anderen und nicht während der Ausführung des SUT, z.B. wenn eine Komponente zerstört und später neu erstellt wird. Ist ein Bezeichner einmal vergeben, muss er Bestand haben. Leider gibt es kein automatisches Schema zur Vergabe von Bezeichnern, das diese Anforderungen erfüllt. Solche Systeme erstellen Bezeichnern meist aus dem Bezeichner der jeweiligen Klasse und einem laufenden Index, was unweigerlich fehlschlägt, da der Bezeichner dann von der Reihenfolge der Erstellung der Komponenten abhängt. Da Bezeichner eine so zentrale Rolle bei der Erkennung von Komponenten spielen, können unzuverlässige Bezeichner, insbesondere automatisch generierte, sehr negative Effekte haben. Falls die Entwickler nicht überzeugt werden können, derartige Bezeichner durch eindeutige, stabile zu ersetzen, oder sie zumindest wegzulassen, können solche Bezeichner mit Hilfe eines `NameResolver` unterdrückt werden. Weitere Informationen hierzu finden Sie in Abschnitt 54.1.7(1163).

Es ist für die Arbeit mit QF-Test nicht nötig, Bezeichner flächendeckend zu vergeben. Allzu ausgiebiger Gebrauch kann sogar kontraproduktiv sein, da QF-Test ein eigenes Konzept dafür hat, ob eine Komponente "interessant" ist oder nicht. Uninteressante Komponenten werden wegabstrahiert und können so bei einer Änderung keine Probleme verursachen. Typische Beispiele für solche Komponenten sind Panels, die nur für Layout-Zwecke eingebaut werden. Eine Komponente mit nicht trivialem Bezeichner gilt für QF-Test immer als interessant, so dass die Benamung von unwichtigen Komponenten zu Problemen führen kann, wenn diese in einer späteren Version aus der Hierarchie entfernt werden.

Auch müssen Bezeichner nicht global eindeutig vergeben werden. Jede Klasse von Komponenten bildet einen eigenen Namensraum, so dass es keinen Konflikt gibt, wenn ein Textfeld und ein Button den selben Bezeichner haben. Außerdem genügt es, wenn die Bezeichner innerhalb einer Anzeige eindeutig sein - sie brauchen also nicht applikationsweit eindeutig zu sein (außer bei Web-Applikationen). Andererseits ist der Opti-

malfall hinsichtlich stabiler Komponentenerkennung in den Regressionstests, wenn die Bezeichner applikationsweit eindeutig sind. Dann können Sie die Optionen Gewichtung von Namen (Wiedergabe)[548] und Gewichtung von Namen (Aufnahme)[520] auf "Name übertrifft alles" setzen, die flexibelste Einstellung zur Wiedererkennung von Komponenten.

Zwei Fragen sind noch offen: Welche Komponenten sollten Bezeichner erhalten und was sind eigentlich gute Bezeichner? Eine gute Daumenregel ist es, allen Komponenten, mit denen ein Anwender direkt interagieren kann, einen Bezeichner zu geben, also Knöpfen, Textfeldern, etc. Komponenten, die nicht direkt erzeugt werden, sondern automatisch als Elemente von komplexen Komponenten, brauchen keinen Bezeichner. Dies sind z.B. die Listenelemente einer `ComboBox`. Die komplexe Komponente selbst (im Beispiel also die `ComboBox`) sollte dagegen sehr wohl einen Bezeichner haben.

Falls Bezeichner für Komponenten nicht von Anfang an vergeben wurden und die Entwickler nur den geringstmöglichen Aufwand spendieren wollen, diese zur Verbesserung der Testautomatisierung nachzuziehen, hat sich folgende Strategie als guter Kompromiss erwiesen: Fenster, komplexe Komponenten wie Bäume und Tabellen, sowie Panels, die eine Anzahl von Komponenten zu einer Art Formular zusammenfassen, sollten einen Bezeichner erhalten. So lange Struktur und Geometrie der Komponenten in solchen Formularen einigermaßen konstant sind, führt dies zu einer guten Wiedererkennung und brauchbaren QF-Test ID Attributen. Probleme durch individuelle Komponenten mit wechselnden Eigenschaften können von Fall zu Fall behandelt werden, entweder indem entwicklungsseitig nachträglich ein Bezeichner vergeben wird, oder mit Hilfe eines `NameResolver`.

Beeinflussen des Name-Attributs mittels NameResolver

In unterschiedlichen Applikationen kommen unterschiedlichste Namenskonzepte für Komponenten zum Einsatz. Nicht in allen Fällen liefern sie stabile Bezeichner, die für Regressionstests sinnvoll genutzt werden können: beispielsweise, wenn die Komponenten keine Bezeichner haben oder Bezeichner existieren, die sich aber von Zeit zu Zeit ändern, z.B. zeichnen Sie einen 'button1' beim ersten Mal auf und beim nächsten Mal heißt dieser Button plötzlich 'button2'. Eine andere Variante könnte sein, dass die aktuelle Versionsnummer des SUT im Bezeichner des Hauptfensters steckt.

Manchmal kennen die Tester jedoch einen Algorithmus, um eindeutige Namen zu setzen. In solchen Fällen sollten Sie sich Das `NameResolver` Interface[1163] im Kapitel Das `resolvers` Modul[1154] genauer anschauen.

Ein NameResolver kann verwendet werden, um Bezeichner aus der QF-Test Sicht zu ändern oder zu löschen. Diese Änderungen treffen dann nur für die QF-Test Sicht zu und ändern nichts am aktuellen Sourcecode.

Sie sollten über den Einsatz von NameResolver nachdenken, wenn:

- das SUT dynamische Bezeichner besitzt.
- Sie einen Algorithmus kennen, um eindeutige Bezeichner zu vergeben.
- Sie entwicklungsseitige Änderungen von Bezeichnern ”neutralisieren” wollen, z.B. bei einer neuen Version.
- Sie die Namensgebung der Komponenten optimieren wollen, z.B. einige Teile ausschneiden, um lesbarere QF-Test IDs für Komponenten in QF-Test zu erhalten.

Wenn Sie eine Eindeutigkeit pro Fenster erreichen (bei Web für alle Seiten), können Sie darüber nachdenken die Optionen Gewichtung von Namen (Wiedergabe)[548] und Gewichtung von Namen (Aufnahme)[520] auf ”Name übertrifft alles” zu setzen.

Hinweis

Wenn es möglich ist, dass die Entwickler eindeutige und stabile Bezeichner direkt vergeben, dann sollten es diese im Quellcode tun, da die Entwickler am besten wissen, welche Komponenten angelegt werden. Das Implementieren eines NameResolvers kann eine sehr aufwendige und mühsame Aufgabe sein, besonders wenn sich das GUI stark ändert.

NameResolver sind in Abschnitt 54.1.7[1163] beschrieben.

5.4.3 Merkmal

Im Merkmal Attribut wird, grob gesagt, ein Text abgelegt, der für die Wiedererkennung nützlich ist und direkt mit der Komponente selbst in Verbindung steht. Dies kann entweder der Text der Komponente sein (z.B. Beschriftung auf einem Button), eine programmtechnisch zugeordnete Beschriftung/Label der Komponente (z.B. CheckBox, RadioButton, TextField) oder ein Titel (Fenster[919], Dialog, TitledPanel). Bei einer Webseite[925] die URL. Die konkreten Werte, die in Frage kommen, sind klassen- und technologiespezifisch. Für alle Engines außer Web siehe Generische Klassen[1329]. Für Web gelten die in Merkmal bei Web-Komponenten[72] beschriebenen Zuordnungen.

Häufig ist der Wert des Merkmal mit dem in Weiteres Merkmal[933] `qfs:labelBest` identisch. Dies liegt daran, dass die Beste Beschriftung[76] für die Komponente in einer der `qfs:label*`-Varianten[74]abgespeichert wird und dies meist der Text ist, der direkt mit der Komponente in Verbindung steht. Die Redundanz macht dennoch Sinn, da für das weitere Merkmal der Status gesetzt werden kann: ’Ignorieren’, ’Sollte übereinstimmen’ oder ’Muss übereinstimmen’. Für das Merkmal gilt implizit immer der Status ’Sollte übereinstimmen’. Es kann aber aus Gründen der Rückwärtskompatibilität nicht von den `qfs:label*`-Varianten[74] abgelöst werden.

Der Einfluss des Merkmal bei der Generierung der QF-Test ID[931] ist in Generierung der QF-Test ID der Komponente[1017] beschrieben. Beispiele hierzu finden Sie in Woran erkennt man eine robuste Komponentenerkennung?[55].

Komponenten können in einer SmartID[81] auch direkt über das Merkmal adressiert werden, ohne Aufnahme eines Komponente-Knoten[78].

Verwendung von regulären Ausdrücken oder Arbeiten mit dynamischen Fenstertiteln

Video

Im Video

'Komponentenerkennung'
https://www.qftest.com/de/yt/komponentenerkennung.html

wird die Verwendung von regulären Ausdrücken in Fenstertiteln ab Minute 13:07 erläutert.

In vielen Applikationen werden Sie auf die Situation treffen, dass keine eindeutigen Namen seitens der Entwicklung vergeben wurden und QF-Test dieselben Komponenten immer wieder an unterschiedlichen Stellen aufzeichnet. Die Wiedergabe dieser aufgezeichneten Komponenten funktioniert dann meistens solange sich die Geometrie des Fensters nicht drastisch verändert.

In diesem Fall ist es wahrscheinlich, dass der Titel des Hauptfensters sich immer wieder ändert, z.B. kann der Titel die aktuelle Versionsnummer, den gerade eingeloggten Benutzer oder einen Dateinamen beinhalten. Wenn Sie Ihre Tests stabil halten wollen und die unterschiedlich aufgezeichneten Fenster als einen Fenster Knoten unter dem Fenster und Komponenten Knoten behandeln wollen, dann selektieren Sie den Fenster Knoten und editieren Sie dessen Merkmal Attribut. Dort ersetzen Sie den dynamischen Teil durch einen regulären Ausdruck und haken dabei die Checkbox 'Als Regexp' an. Nun sollten Ihre Tests wieder funktionieren.

Hier sehen Sie ein Beispiel für einen regulären Ausdruck für eine Komponente des CarConfigurators, deren Merkmal mit dem Wort 'Fahrzeuge' beginnt, aber danach einen beliebigen dynamischen Teil enthalten kann:

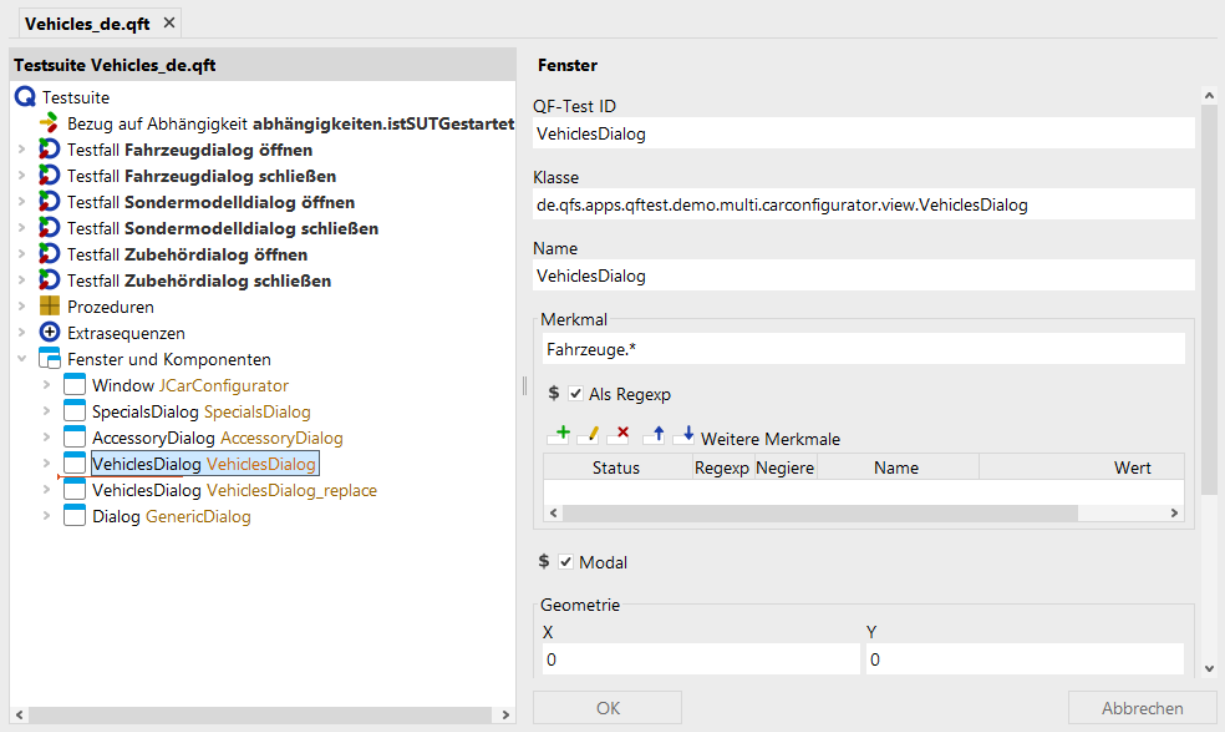


Abbildung 5.8: Ein regulärer Ausdruck im Merkmal Attribut

Reguläre Ausdrücke finden bei QF-Test an vielen Stellen Verwendung. Im Abschnitt 49.3(1023) finden Sie detaillierte Informationen zu ihrer Verwendung.

Merkmal bei Web-Komponenten

Das Merkmal für Web-Komponenten wird gemäß der folgenden Logik ermittelt:

- Bei HTML-Elementen, die ein `Frame` oder `Document` sind, wird die URL in das Merkmal Attribut eingetragen.
- Falls keine der speziellen nachfolgenden Zuordnung zutrifft, wird das `id`-Attribut des HTML-Elements verwendet, soweit vorhanden und eindeutig genug. Falls nicht, wird der Text des HTML-Elements, bei Bedarf gekürzt, im Merkmal Attribut eingetragen.
- Spezielle Zuordnungen:

Tag-Name des HTML-Elements	Wert des Merkmal Attributs
TEXT	Text des HTML-Elements, bei Bedarf gekürzt
A	Text des HTML-Elements (bei Bedarf gekürzt), sonst Fenstertitel, sonst URL
FIELDSET	Text eines darin enthaltenen HTML-Elements mit dem Tag ”LEGEND”
FORM	Wert des Attributs ”name”
IMG	Wert des Attributs ”alt”, sonst ”src”, sonst Teil der URL
INPUT	Wert des Attributs ”name” Bei RadioButtons mit identischen Namen -> Wert des Attributs ”name” mit angehängtem Wert des Attributs ”value”
BUTTON	Wert des Attributs ”name”, sonst Text des HTML-Elements, bei Bedarf gekürzt
LABEL	Text des HTML-Elements, bei Bedarf gekürzt
SELECT	Wert des Attributs ”name”
OPTION	Wenn die Option `OPT_WEB_USE_OPTION_LABEL` gesetzt ist, Wert des Attributs ”label”, ansonsten der Text des HTML-Elements, bei Bedarf gekürzt
OPTGROUP	Wert des Attributs ”label”
IFRAME	Wert des Attributs ”id”, sonst ”name”, sonst ”src”

Tabelle 5.1: Merkmal Attribut für Web-Komponenten

Bei sehr speziellen Konstellation kann das Merkmal Attribut auch einen Wert erhalten, der von obiger Logik abweicht.

5.4.4 Weitere Merkmale

In der Tabelle Weitere Merkmale[933] werden unterschiedliche Informationen, die für die Wiedererkennung der Komponente nützlich sein können, abgespeichert. Unter Weitere Merkmale[933] finden Sie eine Aufstellung der standardmäßig erzeugten Einträge. Sie können aber auch eigene über einen `ExtraFeatureResolver` (siehe Abschnitt 54.1.11[1168]) hinzufügen.

Einige der weiteren Merkmale werden prophylaktisch aufgezeichnet und im Normalfall gar nicht für die Komponentenerkennung genutzt. Dies betrifft hauptsächlich Informationen zur Komponentenklasse, die QF-Test für die Ableitung der Generische Klassen[62] nutzt. Sie erhalten standardmäßig den Status ’Ignorieren’. Dieser kann umgestellt werden, wenn in Spezialfällen der Originalwert von Interesse ist.

Für die Wiedererkennung ist die Beschriftung einer Komponente relevant, für die unter-

schiedliche `qfs:label*`-Varianten zur Verfügung stehen.

`qfs:label*`-Varianten

Beschriftungen sind nach dem Komponentenbezeichner(66), der von den Entwicklern vergeben und von QF-Test im Attribut Name(64) gespeichert wird, das nächstbeste Kriterium für die Komponentenerkennung.

Beschriftungen können über unterschiedliche Arten dargestellt werden. Bei einem Button ist dies meist der eigene Text, bei einem Textfeld hingegen ein Text, der häufig links oder oberhalb des Eingabefeldes steht. Es gibt programmtechnisch zugeordnete Beschriftungen. Auch der Tooltip, der Name eines Images oder der Titel der Komponente können als Beschriftung genutzt werden.

QF-Test speichert Informationen zur Beschriftung in den weiteren Merkmalen ab. Bis QF-Test Version 6 wurde nach einem bestimmten Algorithmus die am besten passende Beschriftung ermittelt und im weiteren Merkmal `qfs:label` abgespeichert.

7.0+ Ab QF-Test Version 7.0 werden in Frage kommende Beschriftungen in weiteren Merkmalen abgespeichert, die mit `qfs:label` beginnen und einen spezifischen Zusatz für den Fundort der Beschriftung erhalten, zum Beispiel `qfs:labelText` für den Text des Buttons oder `qfs:labelLeft` für die Beschriftung links des Eingabefeldes. Der Vorteil einer spezifischen Beschriftung ist einerseits die Performanz bei der Wiedergabe, da direkt nach der entsprechenden Beschriftung gesucht werden kann, andererseits die Flexibilität.

Hinweis Wenn Sie in vorhandenen Komponentenknoten das weitere Merkmal `qfs:label` auf den neuen Algorithmus umstellen wollen, ändern Sie bitte den Namen des weiteren Merkmals auf `qfs:labelBest`. In diesem Fall wird, wie bisher, aus allen in Frage kommenden Beschriftungen die beste Beschriftung ermittelt. Weitere Informationen zur Umstellung finden Sie im Kapitel Das `ExtraFeatureResolver` Interface(1168).

Für Beschriftungen, die über die Position gefunden werden, gibt die folgende Grafik Auskunft:

qfs:labelTopleft	qfs:labelTop	-
qfs:labelLeft	betrachtete Komponente	qfs:labelRight
-	qfs:labelBottom	-

Tabelle 5.2: `qfs:label*`-Varianten mit Position

Nachfolgend finden Sie eine Auflistung aller zur Verfügung stehenden `qfs:label*`-Varianten, im Anschluss daran die Beschreibungen. Außerdem wird die Syntax für die direkte Adressierung in einer SmartID(81) aufgelistet.

Die Angabe unter ”Kategorie” entspricht den Begriffen, die bei der Beschreibung für Generische Klassen(1329) in der Rubrik ”qfs:label*” verwendet werden.

qfs:label*-Varianten	SmartID-Kennzeichner	Kategorie
qfs:labelTitle	#title=	Titel
qfs:labelFor	#for=	Zugeordnetes Label
qfs:labelText	#text=	Eigener Text
qfs:labelLeft	#left=	Label in der Nähe
qfs:labelTop	#top=	Label in der Nähe
qfs:labelRight	#right=	Label in der Nähe
qfs:labelBottom	#bottom=	Label in der Nähe
qfs:labelTopleft	#topleft=	Label in der Nähe
qfs:labelInherited	#inherited=	Label in der Nähe
qfs:labelTooltip	#tooltip=	Tooltip
qfs:labelImage	#image=	Icon-Beschreibung
qfs:labelPlaceholder	#placeholder=	Prompt

Tabelle 5.3: qfs:label*-Varianten

qfs:labelTitle
Die Überschrift einer Komponente, zum Beispiel eines Fensters (Komponentenklasse ”Window”) oder einer Kachel mit Titel, also der Komponentenklasse ”Panel:titledPanel”.

qfs:labelFor
Text, der im Programmcode der Komponente als Beschriftung zugewiesen wird. Bei Web-Anwendungen zum Beispiel über das HTML-Attribut labelFor.

qfs:labelText
Der Text der Komponente selbst.

qfs:labelLeft
Der Text einer Komponente der Klasse ’Label’ links der betrachteten Komponente.

qfs:labelTop
Der Text einer Komponente der Klasse ’Label’ über der betrachteten Komponente.

qfs:labelRight
Der Text einer Komponente der Klasse ’Label’ rechts der betrachteten Komponente.

qfs:labelBottom
Der Text einer Komponente der Klasse 'Label' unter der betrachteten Komponente.

qfs:labelTopleft
Der Text einer Komponente der Klasse 'Label' oben links der betrachteten Komponente.

qfs:labelInherited
Die Beschriftung einer anderen Komponente. Beispiel: "Straße: Hauptstraße 11", wobei der Straßenname und die Hausnummer in eigenen Eingabefeldern stehen. Das Eingabefelder für die Hausnummer erhält hier "qfs:labelInherited" mit dem Wert "Straße:".

qfs:labelTooltip
Der Tooltip der Komponente selbst.

qfs:labelImage
Der Name eines der Komponente zugeordneten Images.

qfs:labelPlaceholder
Nur bei Web-Anwendungen. Der Prompt, der für die Komponente angezeigt wird, wenn sie keine Eingabe des Benutzers enthält.

Der Einfluss auf die QF-Test ID der Komponente der `qfs:label*`-Variante, die die Beste Beschriftung[76] darstellt, ist in Generierung der QF-Test ID der Komponente[1017] beschrieben.

Informationen zum Übergang vom alten auf den neuen Algorithmus finden Sie in Das `ExtraFeatureResolver` Interface[1168].

Beste Beschriftung

Bei der Komponentenanalyse werden die in Frage kommenden Beschriftungen nach bestimmten Kriterien bewertet und als in den Weiteren Merkmalen abgespeichert, wobei die Namen der Einträge `qfs:label` anfangen (siehe Tabelle 5.3[75]). Die `qfs:label*`-Variante mit der besten Bewertung (beste Beschriftung) erhält den Status "Sollte übereinstimmen", die anderen "Ignorieren". Die Reihenfolge der Einträge in obiger Tabelle entspricht grob der Gewichtung. Sie ist aber auch von der Komponentenklasse abhängig. Dokumentiert ist dies in der Rubrik "qfs:label*" in den Eigenschaften der einzelnen Generische Klassen[1329]. Bei den `qfs:label*`-Varianten der Kategorie "Label in der Nähe" spielen auch die Entfernung des Labels und der Überlappungsgrad eine Rolle.

Die beste Beschriftung wird zusätzlich in dem weiteren Merkmal mit dem Namen `qfs:labelBest` abgespeichert. In einer SmartID kann dieses Merkmal über den

Kennzeichner `qlabel` angesprochen werden. Siehe auch SmartID-Syntax für Weitere Merkmale[86].

qfs:text

`qfs:text` enthält den Textinhalt der Komponente. Dies dient als Zusatzinformation für Textfelder oder PDF-Komponenten. Es wird nicht aufgenommen, kann aber seit QF-Test Version 5.3 ohne zusätzlichen Resolver für die Komponentenerkennung genutzt werden.

value

Web

`value` enthält bei bestimmten HTML-Komponenten wie Checkboxen und Radiobuttons den Wert des gleichnamigen HTML-Attributs, sofern dieser Aussagekräftig ist. Dabei spiegelt `value` nicht den aktuell ausgewählten Wert oder gar den Selektionsstatus des Elements wider, sondern den statischen Wert, der übertragen wird, wenn das Element ausgewählt ist.

5.4.5 Index

Der Index einer Komponente kann ebenfalls zur Wiedererkennung verwendet werden. Allerdings ist zwischen dem Index[935] eines Komponente-Knoten[78] und dem in einer SmartID[81] verwendeten Index zu unterscheiden: Ersterer bezieht sich immer auf GUI-Elemente dieser Klasse bezogen auf die Parent-Komponente. Bei der SmartID bezieht sich der Index auf die in Frage kommenden Komponenten mit der angegeben SmartID (siehe SmartID mit Index[88]).

5.4.6 Geometrie

Die Geometrie trägt nur einen geringen Teil zur Komponentenerkennung bei, wenn andere Kriterien vorhanden sind. Es kann aber auch vorkommen, dass Komponenten weder einen Namen noch eine Beschriftung noch brauchbare weitere Merkmale noch einen brauchbaren Index vorweisen. Wenn dann auch über anwendungsspezifische Resolver (siehe Abschnitt 54.1[1154]) keine Wiedererkennungsmerkmale bereitgestellt werden können, stützt sich Wiedererkennung auf die Komponentenklasse, die immer vorhanden ist, die Komponentenhierarchie sowie Position und Größe der Komponente.

Wenn in diesem Fall darauf geachtet wird, dass die Größe der Fenster bei der Wiedergabe die gleiche Größe wie bei der Aufnahme haben (siehe Komponentenevent[790]), sollte die Komponentenerkennung stabil sein. Der Änderungsaufwand bei Versionswechseln

der Applikation kann aber etwas höher sein, da Positionsänderungen von Komponenten explizit nachgezogen werden müssen.

5.4.7 Komponentenhierarchie

Die Verschachtelung wird bei Komponentenevent(790) ebenfalls für das Wiederauffinden genutzt.

Bei Komponente-Knoten werden die Container, in denen eine Komponente liegt, bei der Aufnahme aufgezeichnet. Ob sie bei vorhandenem Name Attribut für die Wiedererkennung genutzt werden sollen, kann über die Optionen Gewichtung von Namen (Wiedergabe)(548) und Gewichtung von Namen (Aufnahme)(520) gesteuert werden. Die jeweils günstige Einstellung wird in Über die Vergabe von Bezeichnern(68) beschrieben.

Komponente-Knoten können im Komponentenbaum aus tiefen Verschachtelungen in übergeordnete Knoten verschoben werden, solange diese Verflachung der Komponentenhierarchie die Erkennung nicht beeinträchtigt. Manchmal kann diese Vorgehensweise sogar zu besserer Stabilität der Erkennung führen, insbesondere, wenn die Wiedererkennungskriterien für die übergeordneten Knoten instabil sind. Oder wenn übergeordnete Komponenten nicht immer existieren, zum Beispiel Scroll-Panels, die nur nach Bedarf in die GUI-Hierarchie eingefügt werden.

Bei einer SmartID(81) kann die Komponentenhierarchie ebenfalls für die Wiedererkennung genutzt werden. Hier wird die zu verwendende übergeordnete Komponente (oder auch mehrere) explizit in der SmartID angegeben. Nähere Informationen finden Sie in SmartID-Syntax für Komponentenhierarchien(88).

Über Geltungsbereich (Scope)(90) kann der Suchbereich für Komponenten bei der Wiedergabe auf eine bestimmte Hierarchieebene eingeschränkt werden.

5.5 Komponente-Knoten

Wenn Komponente(930) Knoten an Stelle von SmartID(81)s verwendet werden, speichert QF-Test die Wiedererkennungskriterien der aufgezeichneten Komponenten in Fenster(919) und Komponente(930) Knoten ab, deren hierarchische Anordnung der Struktur des GUI im SUT entspricht. Diese Knoten befinden sich unterhalb des Fenster und Komponenten(942) Knotens. Das folgende Bild zeigt einen Ausschnitt der Komponente-Knoten, die zum QF-Test Hauptfenster gehören:

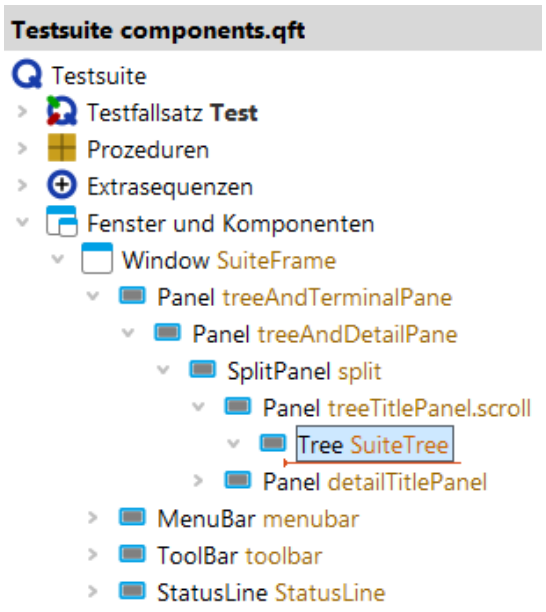


Abbildung 5.9: Komponentenhierarchie eines Swing SUT

Im Detailbereich eines Komponente(930) Knotens werden die Wiedererkennungskriterien(62) abgespeichert. Zusätzlich befindet sich darin das QF-Test ID Attribut. Dieses ist die Referenz-ID für alle Knoten in den Tests, die sich auf eine Komponente beziehen.

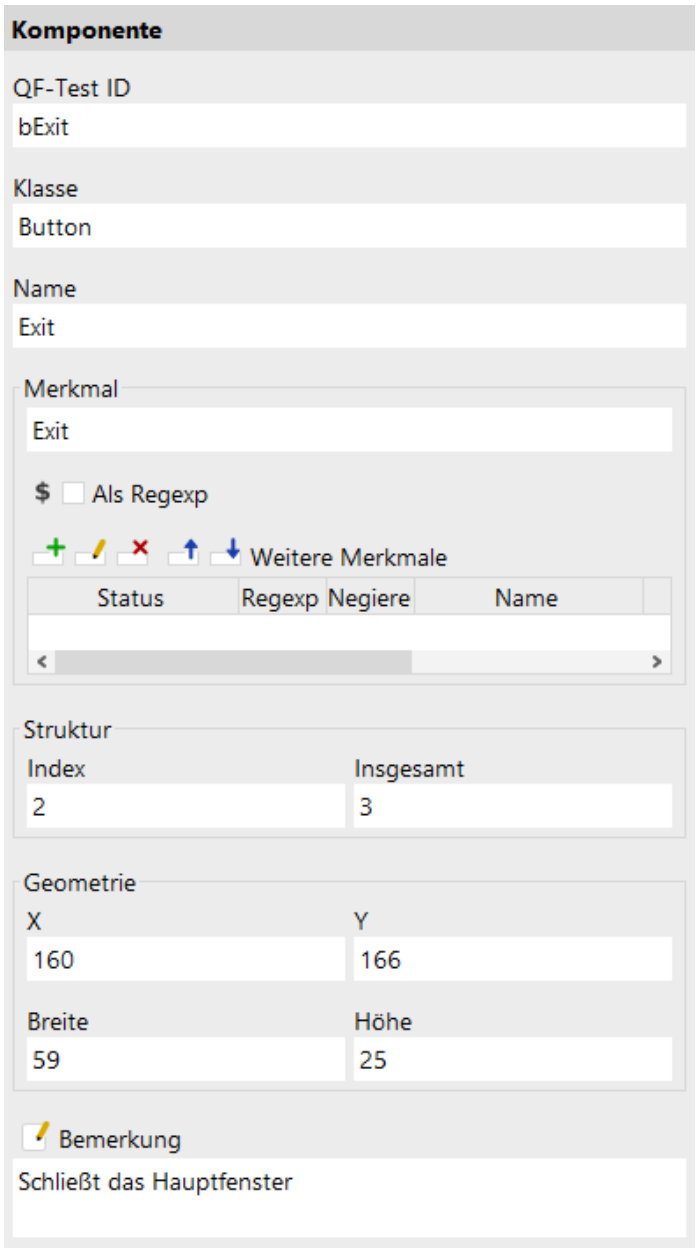


Abbildung 5.10: Komponentenknoten

Jeder Knoten einer Testsuite besitzt ein QF-Test ID[931] Attribut, welches für die meisten Knoten keine besondere Bedeutung hat und automatisch verwaltet wird. Für Komponenten Knoten hat die QF-Test ID dagegen eine wichtige Funktion. Andere Knoten mit einer Zielkomponente, wie Events oder Checks, haben das Attribut QF-Test ID der Komponente, welches sich auf die QF-Test ID des Komponente-Knotens bezieht. Diese indirekte Referenzierung der GUI-Elemente ist sehr nützlich: wenn sich die Oberfläche des SUT auf eine Weise ändert, die QF-Test nicht automatisch kompensieren kann, müssen le-

diglich die Komponenten Knoten der nicht erkannten Komponenten angepasst werden und der Test läuft wieder.

Es ist sehr wichtig sich klarzumachen, dass die QF-Test ID einer Komponente nur ein künstliches Konzept für den internen Gebrauch innerhalb von QF-Test ist, nicht zu verwechseln mit dem Attribut Name welches zur Identifikation der Komponente im SUT dient, worauf wir im folgenden Abschnitt genau eingehen werden. Der eigentliche Wert der QF-Test ID ist vollkommen irrelevant und steht in keinem Bezug zum GUI des SUT. Entscheidend ist nur, dass die QF-Test ID eindeutig ist und dass andere Knoten sich korrekt darauf beziehen. Andererseits wird die QF-Test ID des Komponente-Knotens in der Baumansicht angezeigt, und zwar nicht nur für die Komponente selbst, sondern auch für Events und andere Knoten, die sich darauf beziehen. Folglich sollten Komponenten aussagekräftige QF-Test IDs besitzen, die auf die eigentliche Komponente im GUI hinweisen.

Wenn QF-Test einen Komponente-Knoten anlegt, muss es ihm automatisch eine QF-Test ID zuweisen. Es tut sein Bestes, einen aussagekräftigen Bezeichner aus den verfügbaren Informationen zu konstruieren. Details hierzu finden Sie in Generierung der QF-Test ID der Komponente(1017). Sollte Ihnen eine generierte QF-Test ID nicht zusagen, können Sie diese ändern. Wenn Sie dabei einen Wert wählen, der schon vergeben ist, gibt QF-Test eine Warnung aus. Haben Sie bereits Events aufgezeichnet, die sich auf diese Komponente beziehen, bietet QF-Test an, deren QF-Test ID der Komponente Attribute automatisch anzupassen. Diese Automatik funktioniert nicht bei Referenzen mit Variablen im QF-Test ID der Komponente Attribut.

Hinweis

Ein häufig gemachter Fehler ist es, das Attribut QF-Test ID der Komponente eines Events anstelle der QF-Test ID selbst zu ändern. Dadurch wird die Verbindung zwischen dem Event und seiner Zielkomponente zerstört, was zu einer `UnresolvedComponentIdException`(966) führt. Sie sollten das also nur dann tun, wenn Sie tatsächlich die Zielkomponente ändern wollen.

Häufig werden Tests aus vorhandenen Prozeduren zusammengestellt. Dann ist es oft hilfreich, dass in Abschnitt Komponenten aufnehmen(44) beschriebene Verfahren zum direkten Aufzeichnen von Komponenten zu verwenden. Die QF-Test ID der so aufgenommenen Komponente wird in der Zwischenablage des Betriebssystems gespeichert und kann somit einfach im entsprechenden Prozedurparameter eingefügt werden.

Man kann mittels Komponenten aufnehmen(44) auch zunächst die gesamte Struktur der Komponenten des SUT erstellen, um einen Überblick zu erhalten und sinnvolle QF-Test IDs zu vergeben. Bei Aufnahmen werden dann diese QF-Test IDs verwendet.

5.6 SmartID

6.0+

SmartIDs ermöglichen eine einfache und flexible Wiedererkennung von Komponenten

direkt auf Basis der ID, ohne dass die Wiedererkennungskriterien an anderer Stelle abgespeichert werden. Dies verschlankt den aufgenommenen Komponentenbaum in ”Fenster und Komponenten” deutlich. Bei ausschließlicher Verwendung von SmartIDs wird der Komponentenbaum überhaupt nicht mehr benötigt. Es ist allerdings zu berücksichtigen, dass diese Flexibilität und Vereinfachung ihren Preis hat und - je nach Situation - die Performanz und Wartbarkeit beeinträchtigen kann.

Video

Im Februar 2024 fand ein Spezialwebinar zum Thema statt. Hier geht es zum

Videomitschnitt des Spezialwebinars
https://qftest.com/de/yt/smartid-spezialwebinar.html

auf unserem QF-Test YouTube-Kanal.

SmartIDs verwenden die gleichen Wiedererkennungsmerkmale, die bei der klassischen Komponentenerkennung in den Komponente-Knoten(78) abgespeichert werden. Der Unterschied ist, dass aus den möglichen Wiedererkennungsmerkmalen eines oder mehrere explizit ausgewählt und an Stelle der Referenz zur aufgenommenen Komponente eingetragen werden. Zum Beispiel direkt im Attribut QF-Test ID der Komponente(776) eines Mausklick-Knotens.

Ziel der SmartIDs ist eine Verschlankung des Komponentenbaums - so weit sinnvoll, aber nicht um jeden Preis. Einfaches soll möglichst einfach genutzt werden können, aber wenn es schwierig ist, eine Komponente anzusprechen, sind Komponente(930) Knoten besser geeignet. Die Themen ”Eindeutigkeit” und ”Performanz” können alternativ über das Scope-Konzept adressiert werden, siehe Geltungsbereich (Scope)(90).

Die SmartID wird durch ein führendes `#` gekennzeichnet. Die einfachste Version der SmartID ist entweder der Name oder die Beschriftung der Komponente mit einem vorangestellten `#`. Zum Beispiel `#username`, um eine Komponente mit dem Namen `username` anzusprechen oder `#Anwendername`, wenn `Anwendername` die Beschriftung des Feldes ist.

Typischerweise besteht die SmartID jedoch aus `#`, gefolgt von der Klasse der Komponente, die mit einem Doppelpunkt abgeschlossen wird. Bei Beschriftungen folgt nun ein optionaler Kennzeichner mit Gleichheitszeichen. Dahinter steht der Wert der SmartID, zum Beispiel `#TextField:left=Anwendername`.

Der Kennzeichner gibt an, worauf sich der SmartID-Wert bezieht. Bei der Wiedergabe ohne Kennzeichner greift die Option Priorität bei Aufnahme von SmartIDs mit Kennzeichner(560). Es stehen folgende Kennzeichner zur Verfügung:

- `name`: Name(64), siehe auch SmartID-Syntax für Name(85). Dieser Kennzeichner wird normalerweise nicht aufgenommen.
- `feature`: Merkmal(70), siehe auch SmartID-Syntax für Merkmal(86). Dieser Kennzeichner wird normalerweise nicht aufgenommen.

- `label`: Merkmal[70] ist eine Sonderform, die `feature` oder eine der `qfs:label*`-Varianten für die Übereinstimmung akzeptiert. Dieser Kennzeichner wird normalerweise nicht aufgenommen.
- Die Namen der Weitere Merkmale[73], beziehungsweise Kurzformen davon, zum Beispiel `qlabel` für die Beste Beschriftung[76]. Siehe auch SmartID-Syntax für Weitere Merkmale[86]. Diese Kennzeichner werden immer aufgenommen.

Die Angabe der Klasse und des Kennzeichners dienen der besseren Lesbarkeit und der Performanz. Ob die Klasse aufgenommen wird, kann über die Option Für SmartID immer die Klasse aufnehmen[559] gesteuert werden. Dies gilt jedoch nur für Komponenten, deren Klasse zu den Generische Klassen[1329] gehört. Für andere Klassen muss der Klassenname auf jeden Fall spezifiziert werden, zum Beispiel `#DIV:compid`.

Die SmartID wird im Attribut QF-Test ID der Komponente eingetragen, zum Beispiel im Attribut QF-Test ID der Komponente[776] von Event- oder Check-Knoten. Sie kann genauso wie die QF-Test ID der Komponente in Variablen gespeichert, in Parametern übergeben oder in Skripten genutzt werden. Bei komplexen Komponenten wie Tabellen, Listen oder Bäumen kann die SmartID ebenfalls die QF-Test ID der Komponente ersetzen. Der Index, der das Unterelement bezeichnet, bleibt unverändert. Im Anschluss an eine SmartID kann eine darin enthaltene Komponente entweder über eine weitere SmartID oder mittels XPath (Adressierung mit XPath und/oder CSS-Selektoren[97]) adressiert werden.

SmartID-spezifische Sonderzeichen und Trenner für Unterelemente (":", "@", "&" und "%", siehe auch Abschnitt 48.4[1019]) müssen mit einem vorangestellten \ geschützt werden, wenn sie im Wert der SmartID oder in der Klasse vorkommen.

SmartIDs können für alle Client-Technologien genutzt werden.

Wie bei SmartIDs ist zu berücksichtigen, dass die Aktualisierung nicht so komfortabel ist wie bei Komponente-Knoten[78] Knoten. Allerdings steht in QF-Test eine mächtige "Suchen und Ersetzen" Funktionalität zur Verfügung, um SmartIDs suiteübergreifend anzupassen.

Hinweis Für eine kleine Einführung in SmartID lesen Sie auch unseren Blogartikel SmartID - Die nächste Generation der Komponentenerkennung .

5.6.1 Anwendungsbereiche für SmartIDs

Die Anwendungsbereiche sind die gleichen, in denen bisher Generische Komponenten[91] zum Einsatz kamen. SmartIDs ersetzen generische Komponenten weitgehend und sind einfacher zu nutzen als diese.

Lesbarkeit

Bei der direkten Aufnahme von Testfällen kann der Einsatz von SmartIDs die

aufgenommenen Event- und Checkknoten lesbarer machen. Insbesondere, wenn die aufgenommenen Namen der Komponenten kryptisch sind und stabile Beschriftungen vorhanden sind. In diesem Fall macht es Sinn, die Reihenfolge für die Aufnahme der Wiedererkennungskriterien auf ”zuerst Beschriftung, dann Name” zu ändern. Stellen Sie hierzu die Option Priorität bei Aufnahme von SmartIDs mit Kennzeichner[560] auf `label,name`.

Ignorieren der Komponentenhierarchie

Manche Anwendungen haben tief verschachtelte Komponentenhierarchien. SmartIDs machen es einfach, den Komponentenbaum zu reduzieren, was insbesondere dann hilfreich ist, wenn die Komponentenhierarchie über die Versionen hinweg nicht stabil bleibt. (Bisher wurden zu diesem Zweck Generische Komponenten[91] eingesetzt. Dies ist auch weiterhin möglich, auch parallel zu SmartIDs.)

Testgetriebene Entwicklung

Bei testgesteuerter Entwicklung bieten SmartIDs den großen Vorteil, dass keine Komponente-Knoten[78] Knoten angelegt werden müssen. Außerdem werden bei testgesteuerter Entwicklung häufig die Komponentenbezeichner[66] im technischen Design festgelegt, die dann für die Testerstellung genutzt werden können.

Schlüsselwort-basierende Tests

Schlüsselwort-basierende Tests werden technisch über Prozeduraufrufe und Parameter implementiert. Der Testersteller nimmt somit keine Komponenten auf und ist für die Identifikation der Komponenten auf visuelle Informationen aus dem GUI angewiesen. Dies kann die Beschriftung der Komponente oder deren Funktion (Klasse) sein. Weitere Informationen finden Sie in Schlüsselwortgetriebenes bzw. Keyword-Driven Testing mit QF-Test[413].

Integration mit anderen Test-Tools

Bei der Steuerung der Testausführung seitens QF-Test über andere Test-Tools wie Robot Framework können die Wiedererkennungskriterien direkt über SmartIDs spezifiziert werden.

5.6.2 SmartID-Syntax für Klasse

Die Klasse[62] wird in der SmartID direkt nach dem `#` angegeben und mit einem `:` abgeschlossen, zum Beispiel `#Button:`.

Für Komponentenklassen, mit denen der Anwender typischerweise interagiert, muss die Klasse nicht explizit in der SmartID angegeben werden. Bei diesen Klassen kann

die SmartID auch einfach nur den Komponentenbezeichner(66) oder die Komponentenbeschriftung (entweder das Merkmal(70) oder eine der qfs:label*-Varianten(74)) umfassen, zum Beispiel #btnOK, wenn der Name des Buttons "btnOK" ist, oder #Speichern, wenn die Beschriftung des Buttons "Speichern" lautet. Dies macht eine SmartID einfacher in der Handhabung, allerdings in gewissem Maß auf Kosten der Performance, da ohne Angabe der Klasse mehr Kandidaten auf Übereinstimmung mit der SmartID geprüft werden müssen.

Auf Grund der besseren Performance wird bei der Aufnahme einer SmartID standardmäßig die Klasse mit aufgenommen. Falls dies nicht erwünscht ist, können Sie dies über die Option Für SmartID immer die Klasse aufnehmen(559) deaktivieren.

Im Kapitel 61(1329) ist in den Klasseneigenschaften vermerkt, wenn die Klasse in der SmartID angegeben werden muss. Alle Klassen, die nicht in dem Kapitel genannt werden, müssen immer in der SmartID spezifiziert werden. Beispiel: #DIV:addresses wobei "addresses" hier der Name des DIV-Elements in einer Web-Anwendungen ist.

Panels sind insofern ein Sonderfall, da sich Panels mit Beschriftung für geschachtelte SmartIDs (siehe Abschnitt 5.6.7(88)) oder Scopes (siehe Abschnitt 5.7(90)) anbieten. Daher gehört der Klassentyp Panel:TitledPanel zu den SmartID-Klassen und muss nicht separat angegeben werden.

Wenn Sie zusätzlich zur generischen Klasse einen vordefinierten Klassentyp verwenden, können Sie diese Kombination wie gewohnt schreiben, zum Beispiel #Button:ComboBoxButton:. Die vordefinierten Klassentypen finden Sie in Generische Klassen(1329). Bei eigenen Klassentypen muss der innere Doppelpunkt mittels \ geschützt werden, zum Beispiel #Panel\:myPanel:. Auch die Indextypkennzeichner "@", "&" und "%" müssen mit einem vorangestellten \ geschützt werden (siehe Abschnitt 48.4(1019)).

Informationen zu Kombinationsmöglichkeiten finden Sie in Abschnitt 48.3(1018).

5.6.3 SmartID-Syntax für Name

Ein Name(64) kann in der SmartID direkt nach dem # angegeben werden, z.B. #txtUsername. Wenn die Klasse der Komponente zu den Generische Klassen(1329) gehört, reicht die einfache Angabe des Namens. Ansonsten muss die Klasse(62) vorangestellt werden, zum Beispiel #DIV:txtUsername.

Der verwendete Name kann SmartID-spezifische Sonderzeichen enthalten ":", "@", "&" und "%" (siehe auch Abschnitt 48.4(1019)), diese müssen jedoch mit einem vorangestellten \ geschützt werden.

Um zu erzwingen, dass der Name(64) für die Komponentenerkennung verwendet wird, kann in der SmartID Name= vorangestellt werden, zum Beispiel #Name=txtUsername. Die Groß-/Kleinschreibung muss beim Präfix Name= nicht beachtet werden.

Weitere Informationen zu Kombinationsmöglichkeiten finden Sie in Abschnitt 48.3[1018].

5.6.4 SmartID-Syntax für Merkmal

Das Wiedererkennungskriterium Merkmal[70] kann in der SmartID direkt nach dem # angegeben werden, zum Beispiel `#Anwendername`. Wenn die Klasse der Komponente zu den Generische Klassen[1329] gehört, reicht die einfache Angabe des Merkmals. Ansonsten muss die Klasse[62] vorangestellt werden, zum Beispiel `#DIV:Anwendername`.

Das verwendete Merkmal kann SmartID-spezifische Sonderzeichen enthalten, diese müssen jedoch mit einem vorangestellten \ geschützt werden.

Um zu erzwingen, dass das Merkmal für die Komponentenerkennung verwendet wird, kann dem Merkmal `Feature=` vorangestellt werden, zum Beispiel `#Feature=Anwendername`. Wenn es unwichtig ist, ob das Merkmal oder das weitere Merkmal `qfs:label*`-Varianten[74] verwendet wird, kann `Label=` vorangestellt werden, zum Beispiel `#Label=Anwendername`. Die Groß-/Kleinschreibung muss bei den Präfixen `Feature=` und `Label=` nicht beachtet werden.

Weitere Informationen zu Kombinationsmöglichkeiten finden Sie in Abschnitt 48.3[1018].

5.6.5 SmartID-Syntax für Weitere Merkmale

Wiedererkennungskriterien aus der Gruppe der Weitere Merkmale[73] stehen ebenfalls für die SmartID zur Verfügung. Diese können über den Kennzeichner, der vor den SmartID-Wert gesetzt wird, angesprochen werden. Zwischen Kennzeichner und SmartID steht ein =. Für alle Weiteren Merkmals entspricht der Kennzeichner dem Namen des weiteren Merkmals. Die Groß- und Kleinschreibung muss berücksichtigt werden. Der SmartID-Wert entspricht dem Wert des weiteren Merkmals. Auch hier ist die Groß- und Kleinschreibung relevant.

Beispiele:

- Die SmartID `#module=Modul1` referenziert eine Komponente mit einem weiteren Merkmal mit dem Namen `module` und dem Wert `Modul1`.
- Die SmartID `#my\:foo=Irgend\&etwas` referenziert eine Komponente mit einem weiteren Merkmal mit dem Namen `my:foo` und dem Wert `Irgend&etwas`.

Für die `qfs:label*`-Varianten gibt es Kurzformen für den Kennzeichner. Diese werden weiter unten in diesem Kapitel erläutert.

SmartID-spezifische Sonderzeichen ":", "@", "&" und "%" (siehe auch Abschnitt 48.4[1019]) müssen mit einem vorangestellten \ geschützt werden, wenn sie im Wert der SmartID, Bezeichner oder in der Klasse vorkommen.

Informationen zu Kombinationsmöglichkeiten finden Sie in Abschnitt 48.3[1018].

`qfs:label*`-Varianten

Eine herausragende Stellung haben die `qfs:label*`-Varianten[74], die Beschriftungen für eine Komponente enthalten. Wenn für eine Komponente Beschriftungen existieren, kann in der SmartID entweder die Beste Beschriftung[76] oder eine spezifische Beschriftung genutzt werden. Der Vorteil einer spezifischen Beschriftung ist neben der präziseren Angabe auch die Performanz bei der Wiedergabe, da direkt nach der entsprechenden Beschriftung gesucht werden kann. Andernfalls wird aus allen in Frage kommenden Beschriftungen die beste Beschriftung ermittelt, dieser Vorgang ist zeitintensiver. Um eine spezifische Beschriftung zu nutzen, starten Sie mit `#` dann ein Bezeichner, das heißt, der Kurzform des weiteren Merkmalnamens (siehe `qfs:label*`-Varianten[75]), gefolgt von "=" und dem SmartID-Wert an, zum Beispiel `#left=Vorname`. Die beste Beschriftung können Sie direkt nach dem `#` angeben oder mit `#qlabel=` spezifizieren. Wenn Sie `#label=` verwenden, kann sich der Wert auf das Merkmal oder eine der `qfs:label*`-Varianten beziehen. Die Groß- und Kleinschreibung muss bei diesen Kennzeichnerkurzformen nicht beachtet werden. Die Beschriftung kann in der SmartID auch ohne Kennzeichner verwendet werden, zum Beispiel `#Anwendername`. Dann gilt die in der Option Priorität bei Aufnahme von SmartIDs mit Kennzeichner[560] eingestellte Reihenfolge. Standardmäßig ist dies: Name - beste Beschriftung - Merkmal.

Beispiele:

- `#left=Vorname` - Die Beschriftung links der Komponente muss "Vorname" lauten.
- `#qlabel=Vorname` - Die Beste Beschriftung[76] für die Komponente muss "Vorname" lauten.
- `#label=Vorname` - Entweder muss das Merkmal oder die beste Beschriftung für die Komponente "Vorname" lauten.
- `#Vorname` - Entweder muss der Name der Komponente oder das Merkmal oder die beste Beschriftung für die Komponente "Vorname" lauten.

Weitere Merkmale `qfs:text` und `text`

Auch die weiteren Merkmale `qfs:text` und `text` haben eine Sonderstellung. Beide können über den Kennzeichner `text=` vor dem eigentlichen Wert angesprochen werden. Wenn dediziert `qfs:text` verwendet werden soll, können Sie `qtext=` verwenden.

Beispiele: `#text=Anna`, `#qtext=Benno`

Hinweis

Genaugenommen wird bei vorangestelltem `#text=` zuerst nach dem weiteren Merkmal `#qfs:labelText=` gesucht und dann nach `qfs:text` und `text`. Da

jedoch die beiden letzteren speziell für Textkomponenten gedacht sind, für die `#qfs:labelText=` nicht aufgenommen wird, ergibt sich hier kaum Konfliktpotenzial.

Die weiteren Merkmale `qfs:text` und `text` können auch ohne vorangestelltem `#text=` genutzt werden. Hierzu können Sie die Option Priorität bei Aufnahme von SmartIDs mit Kennzeichner[560] entsprechend ergänzen, zum Beispiel "name,feature,qlabel,text".

Weiteres Merkmal `qfs:type`

Das weitere Merkmal `qfs:type` gibt den Typ einer Klasse an. Wenn nicht einer der von QF-Test vordefinierten Typen (siehe Generische Klassen[1329]) verwendet wird, müssen die darin enthaltenen Doppelpunkte mit \ geschützt werden.

5.6.6 SmartID mit Index

Alle SmartIDs können mit einem Index versehen werden, wenn mehrere Komponenten für die gleiche SmartID in Frage kommen. Dabei zählt die technische Reihenfolge der Komponenten in der Hierarchie. Diese muss nicht der visuellen Reihenfolge entsprechen. Die Zählung des Index beginnt bei 0. Der Index wird in spitzen Klammern angegeben. Wird kein Index angegeben, wird implizit der Index 0 verwendet.

Beispiele: `#Name<2>`, `#TextField:<2>`

Sonderfälle

Bei Komponenten der Klasse `Label` gilt diese Reihenfolge nicht. Da sie in den meisten Fällen als Beschriftung anderer Komponentenklassen genutzt werden und dort im Merkmal oder als `qfs:label*`-Varianten[74] abgespeichert werden, werden die Komponenten der Klasse `Label` nachrangig behandelt. Label Komponenten müssen daher explizit mit vorangestellter Klasse `Label:` angesprochen, zum Beispiel `#Label:Vorname` werden.

Informationen zur allgemeinen SmartID-Syntax finden Sie in Abschnitt 48.3[1018].

5.6.7 SmartID-Syntax für Komponentenhierarchien

Die Komponentenhierarchie[78] kann auch mit SmartIDs für die Wiedererkennung genutzt werden. Als Trennzeichen zwischen den Hierarchieebenen dient `@`.

Beispiele:

Komponente in Container

Die SmartID `#Kundeninformationen@#Name` referenziert eine Komponente mit der SmartID `#Name` in einer übergeordneten Komponente (zum Beispiel einem `TitledPanel`) mit der SmartID `#Kundeninformationen`.

Komponente in ”normaler” Komponente

Manchmal werden Komponenten wie zum Beispiel ein Button keine guten eigenen Wiedererkennungsmerkmale besitzen, über die Komponente, in der sie liegen, sehr gut angesprochen werden können. Ein typisches Beispiel ist hier der Button zum Aufklappen der Liste in einer ComboBox: `#ComboBoxSmartID@#Button:`

Komponente in Unterelement

Links oder Buttons in Listen- oder Tabellenelementen können mit verschachtelten SmartIDs adressiert werden: `#ListenSmartID&22@#Link:<1>` Hierbei adressiert der Teil vor ”@” ein Listenelement, `#Link:<1>` adressiert den zweiten Link darin.

5.6.8 Aufnehmen und Abspielen von SmartIDs

Wenn Sie SmartIDs aufnehmen wollen, aktivieren Sie die Option Aufnahme von SmartIDs[559] oder haken Sie einfach den Menüpunkt Aufnahme→Aufnahme von SmartIDs an.

Bei der Aufnahme von SmartIDs prüft QF-Test zunächst, ob ein Name[64] vorhanden ist. Falls ja, wird dieser für die SmartID verwendet. Wenn nicht, wird nach einer Beschriftung gesucht (Merkmal[70] oder Weitere Merkmale[73].) Wenn die ermittelte SmartID für mehrere Komponenten gültig ist, wird ein Index angefügt.

Hinweis

Normalerweise ist die Aufnahme von SmartIDs relativ einfach. Anhängig von der Zielkomponente und den vorhandenen Informationen kann es aber vorkommen, dass keine SmartID aufgenommenen werden kann, so dass ganz klassisch ein Komponente-Knoten[78] Knoten aufgenommen wird. Dies ist zum Beispiel der Fall, wenn dem GUI-Element keine generische Klasse zugewiesen werden kann oder wenn QF-Test für das GUI-Element weder Name[64], noch Merkmal[70] noch das weitere Merkmal `qfs:label*`-Varianten[74] ermitteln kann.

Standardmäßig wird die generische Klasse der SmartID vorangestellt. Dies erhöht nicht nur die Lesbarkeit sondern kann auch die Performanz bei der Wiedergabe deutlich verbessern. Über die Option Für SmartID immer die Klasse aufnehmen[559] kann das Voranstellen der Klasse deaktiviert werden. Dabei ist zu beachten, dass die Klassen ’Label’ und ’Panel’ trotzdem vorangestellt werden, um die korrekte Wiedergabe zu gewährleisten.

Das Abspielen von Knoten mit SmartIDs unterscheidet sich nicht von dem mit

aufgenommenen Komponenten. Es können beide Varianten innerhalb eines Testfalls verwendet werden. SmartIDs können auch bei aufgenommenen Komponenten zur Adressierung untergeordneter Komponenten verwendet werden. Das Beispiel `aufgenommeneListe&10@#Button:` zeigt Kombination der QF-Test ID einer aufgenommenen Liste mit Index und der SmartID des in dem Listenelement liegenden Buttons.

5.6.9 QF-Test ID der Komponente als SmartID

Es ist möglich, die QF-Test ID einer aufgenommenen Komponente auf eine SmartID inklusive vorangestelltem # Kennzeichen zu setzen. Dies kann genutzt werden, um die SmartID quasi umzuleiten und die Komponentenerkennung klassisch über die Wiedererkennungsmerkmale der aufgenommenen Komponente durchzuführen. Einzelne Komponenten aufzunehmen macht insbesondere dann Sinn, wenn die SmartID lang und umständlich wird, schlechte Performance hat oder schwer eindeutig zu machen ist. Das SmartID-Kennzeichen # kann dann der Einheitlichkeit halber genutzt werden, muss aber nicht.

5.7 Geltungsbereich (Scope)

6.0+ Mit einem Geltungsbereich kann der Suchbereich für Komponenten eingegrenzt werden. Das ist nützlich, um eine Eindeutigkeit von Komponentenreferenzen herzustellen oder die Lesbarkeit eines Tests zu verbessern. Beispiel: Es gibt drei Panels mit Adressdaten mit identisch beschrifteten Textfeldern. Der Geltungsbereich kann nun auf eines der Panels gesetzt werden. Nun beziehen sich die angegebenen SmartIDs ausschließlich auf die Felder in diesem Panel.

Der Geltungsbereich kann unter gewissen Umständen auch zur Beschleunigung der Komponentenerkennung genutzt werden, insbesondere bei Fenstern oder Webseiten, die sehr viele Komponenten enthalten. Ein Beispiel hierzu sind Web-Anwendungen, die von Anfang an alle GUI-Elemente mit dem Status ”unsichtbar” laden, und nur die jeweils relevanten sichtbar schalten. Hierbei kann es nützlich sein, über den Geltungsbereich die Komponentenerkennung zumindest auf das sichtbare Fenster einzugrenzen.

Der Geltungsbereich wird in der Bemerkung eines Knotens, über die SmartID oder auch die QF-Test ID der aufgenommenen Komponente mit vorangestelltem `@scope` gesetzt, zum Beispiel `@scope #myDialog`. Wenn der Scope für mehrere Event- oder Check-Knoten gelten soll, wird der Geltungsbereich in der Bemerkung eines Knotens (zum Beispiel Sequenz, Testschritt oder Testfall) gesetzt, der diese Knoten direkt oder indirekt über Prozeduraufrufe enthält.

Der aktive Geltungsbereich kann vereinfacht über eine SmartID referenziert werden, die

nur aus der Raute # besteht.

Wenn eine Komponente nicht innerhalb des Geltungsbereichs liegt, kommt es zu einer ComponentNotFoundException. Geltungsbereiche können bei Bedarf umgangen werden, indem das Doctag @noscope in der Bemerkung des entsprechenden Event- oder Checkknotens eingefügt wird oder noscope: an den Anfang der SmartID gesetzt wird. Zum Beispiel ist es über die SmartID #noscope:Speichern möglich, den Button "Speichern" anzuklicken, der eigentlich außerhalb des Geltungsbereich liegt, der für die Sequenz gesetzt ist, in der sich das Klick-Event befindet (siehe Abschnitt 48.3(1018)).

Geltungsbereiche können geschachtelt werden, wobei der innere Geltungsbereich im äußeren liegen muss und diesen weiter einschränkt. Auch hier kann mit dem Doctag @noscope eine Ausnahme erreicht werden. Hierzu werden in der Bemerkung des Knotens, dessen Komponenten in einem Geltungsbereich außerhalb des aktuell gültigen liegen, die Doctags @noscope und @scope NEWSCOPE angegeben (NEWSCOPE ist ein Platzhalter für den neuen Geltungsbereich). Die Reihenfolge der Doctags ist beliebig.

Der Geltungsbereich bezieht sich immer nur auf den jeweiligen Knoten und die darin ausgeführten Knoten. Die Komponenten einer im Geltungsbereich gerufenen Prozedur müssen somit im Geltungsbereich liegen oder mit #noscope:... oder Doctag @noscope) gekennzeichnet werden.

Geltungsbereiche können über SmartIDs, aber auch über die QF-Test ID von aufgenommenen Komponenten festgelegt werden. Trotzdem werden Sie nur bei der Referenzierung einer Komponente via SmartID berücksichtigt. Beim Referenzieren einer aufgenommenen Komponente wird der aktuelle Geltungsbereich immer ignoriert.

5.8 Generische Komponenten

Bevor mit QF-Test Version 6.0 SmartID(81)s zur Verfügung standen, waren generische Komponenten das Mittel der Wahl, um Komponentenaufnahmen zu vermeiden. Mit SmartIDs kann dieses Ziel einfacher und flexibler erreicht werden. Die bisherige Beschreibung der generischen Komponenten bleibt aber aus Gründen der Rückwärtskompatibilität hier erhalten.

Ein typischer Anwendungsfall hierfür ist das Testen von lokalisierten Anwendungen.

Eine andere Situation könnte der Einsatz eines GUI Frameworks bei der Entwicklung sein. Dieser Einsatz generiert jede Menge ähnlicher Dialoge, wo sich nur ein paar Komponenten unterscheiden. Aber Sie müssen jedes Mal jeden Bereich neu aufzeichnen, auch wenn Sie dies schon einmal gemacht haben, z.B. globale Navigationsbuttons, weil es sich immer um ein neues Fenster handelt.

Für generische Komponenten benutzen Sie Variablen in den Komponenteneigenschaf-

ten oder löschen einfach nicht-dynamische Teile daraus.

Ein allgemeiner Ansatz für das Generalisieren von Komponenten ist folgender:

1. Zeichnen Sie einige Komponenten, die Sie generalisieren wollen, auf und vergleichen Sie diese.
2. Erstellen Sie eine generische Komponente, die in der QF-Test ID 'generisch' enthält, damit Sie diese wiederfinden.
3. Entfernen Sie alle Attribute, die Sie nicht für die Wiedererkennung verwenden wollen, aus dieser generischen Komponente.
4. Legen Sie die Wiedererkennungseigenschaft fest, z.B: 'Name', 'Merkmal' oder 'Index'.
5. Setzen Sie in diesem Attribut eine Variable, z.B. `$(name)`.
6. Um falsche Treffer zu vermeiden, deaktivieren Sie die Geometrieerkennung, indem Sie in den 'X'- und 'Y'-Attributen ein '-' setzen.
7. Spezifizieren Sie '@generic' im Attribut Bemerkung, damit diese Komponenten nicht unbeabsichtigt von der 'Ungenutzte Komponenten entfernen' Aktion gelöscht wird.
8. Erstellen Sie eine Prozedur, um auf diese generische Komponente zuzugreifen und verwenden Sie die Variable von vorhin als Parameter der Prozedur.

Hinweis Generische Komponenten sind sehr nützlich für das Abspielen von Tests, aber QF-Test verwendet diese nicht für die Aufzeichnung. Sie werden immer die konkreten Komponenten aufgezeichnet bekommen. Sie müssen daher nachträglich die konkreten Komponenten durch die generische ersetzen.

5.9 Unterelemente: Adressierung relativ zur übergeordneten Komponente

In QF-Test ist es möglich, Komponenten relativ zu einer übergeordneten Komponente zu adressieren. Dies ist besonders dann interessant, wenn die untergeordnete Komponente nur im Zusammenspiel mit der übergeordneten eindeutig angesprochen werden kann. Hier gibt es verschiedenste Anwendungsfälle und auch unterschiedliche Möglichkeiten der Implementierung.

Adressierung über Index

Bei Tabellen, Listen und Bäumen macht es Sinn, für die Unterelemente einen Index zu verwenden. Die Hauptkomponente wird über die QF-Test ID der Komponente oder eine SmartID spezifiziert. Der Index für das Unterelement wird daran angehängt. Beispiele: `listid@Eintrag`, `#Table@Spaltenüberschrift&5`. Siehe Adressierung mittels Index(95).

Wenn die Hauptkomponente über eine SmartID adressiert wird, können Reiter in `TabPanels` oder Listeneinträge einer `ComboBox` vereinfacht referenziert werden, zum Beispiel `#Tab:Tab1` oder `#Item:EintragX`. Siehe Adressierung mittels Index(95).

Adressierung des Unterelements mittels SmartID

SmartIDs können an die QF-Test ID der Komponente oder die SmartID, die eine übergeordnete Komponente identifiziert, angehängt werden. Als Trennzeichen zwischen über- und untergeordneter Komponente wird `@` verwendet. Die Verschachtelung kann auch mehrstufig sein. Die einzelnen Komponenten können auch mit einem Index versehen werden.

Beispiele: `#Dialog:@#OK`, `comboboxid@#Button:`, `#Table:&0&0@#CheckBox:`.

Adressierung des Unterelements mittels QPath

Der QPath kann ähnlich wie die angehängte SmartID verwendet werden, ist aber lang nicht so mächtig wie diese. Ein QPath kann an eine QF-Test ID der Komponente angehängt werden. Als Trennzeichen dient `@:`. Die QF-Test ID der Komponente kann auch mit einem Index versehen sein. Beispiele: `buttonid@:Icon`, `tableid&0&0@:CheckBox` Weitere Informationen finden Sie in Adressierung mit QPath(97).

Adressierung des Unterelements mittels XPath und CSS-Selektoren

Web

Bei Web-Anwendungen kann auch ein XPath und/oder ein CSS-Selektor an eine QF-Test ID der Komponente oder eine SmartID angehängt werden. Als Trennzeichen dient `@:xpath=` beziehungsweise `@:css=`. Die QF-Test ID der Komponente kann auch mit einem Index versehen sein. Beispiele: `genericDocument@:xpath=${quoteitem:$(xpath)}`. Weitere Informationen finden Sie in Adressierung mit XPath und/oder CSS-Selektoren(97).

Geltungsbereich

Auch über einen Geltungsbereich kann die übergeordnete Komponente spezifiziert werden. Siehe Geltungsbereich (Scope)(90).

Aufnahme von Unterelementen als Knoten

Bei Tabellen, Listen und Bäumen kann es auch Sinn machen, das Unterelement als Element(936) aufzunehmen. Es hängt von der jeweiligen Situation ab, ob ein Unterelement über Index angesprochen wird oder es sinnvoller ist, es

aufzunehmen. Sie können beide Methoden nach Belieben verwenden und auch mischen. Als Faustregel gilt, dass Element Knoten besser für Komponenten mit wenigen, konstanten Elementen geeignet sind, wie z.B. die Spalten einer Tabelle oder die Reiter eines `TabPanel`. Die Syntax ist vorzuziehen, wenn QF-Test Variablen in Indizes verwendet werden oder wenn die Namen von Elementen variieren oder editierbar sind. Die Option Art der Unterelemente(524) legt fest, ob QF-Test beim Aufzeichnen Element Knoten anlegt oder die QF-Test ID-Syntax verwendet. Mit der Standardeinstellung "Intelligent" folgt QF-Test obigen Regeln. Weitere Informationen zur Aufnahme von Unterelementen finden Sie in Adressierung mit Elemente Knoten(99).

Mögliche Kombinationen

Hinweis: In der nachfolgenden Aufstellung kann die SmartID der übergeordneten Komponente bereits aus geschachtelten Referenzierungen besteht.

Referenzierung der übergeordneten Komponente	Referenzierung der Unterkomponente	Beispiel
QF-Test ID der Komponente	Index	Listenelement mit Textindex: `listid@Eintrag`
SmartID	Index	Tabellenzelle mit numerischen Indizes: `#Table:&0&2`
QF-Test ID der Komponente	SmartID	Icon in Button: `buttonid@#Icon:`
SmartID	SmartID	Textfeld in Dialog: `#Dialog:@#TextField:`
SmartID mit Index	SmartID	Button in Tabellenzelle: `#Table:&0&2@#Button:`
QF-Test ID der Komponente mit Index	SmartID	Button in Tabellenzelle: `tableID&0&2@#Button:`
QF-Test ID der Komponente mit Index	QPath	Button in Tabellenzelle: `tableID&0&2@:Button`
QF-Test ID der Komponente mit oder ohne Index	XPath und/oder CSS-Selektor	`genericHtml@:css=${quoteitem:$(css)}`
SmartID mit oder ohne Index	XPath und/oder CSS-Selektor	`#genericDocument@:xpath=${quoteitem:$(xpath)}`
Geltungsbereich	SmartID	Geltungsbereich als Doctag in einem Testschritt, SmartID im Check-Knoten

Tabelle 5.4: Adressierung von Unterelementen

5.9.1 Adressierung mittels Index

Das Unterelement wird über eine entsprechende Zeichenkettenfolge mithilfe einer speziellen Syntax beschrieben. Die QF-Test ID der Komponente, die im Test verwendet wird, setzt sich hierbei aus der QF-Test ID oder der SmartID der komplexen Komponente (Baum, Tabelle etc.), gefolgt von einem speziellen Trennzeichen, und dem Index des Unterelements zusammen. Das Trennzeichen legt dabei fest, ob es sich um einen numerischen Index, einen Textindex oder einen regulären Ausdruck (vgl. Abschnitt 49.3(1023)) handelt:

Trennzeichen	Index Format
@	Textindex
&	Numerischer Index
%	Regulärer Ausdruck

Tabelle 5.5: Trennzeichen und Indexformat für den Zugriff auf Unterelemente

Um auf eine Zelle in einer Tabelle mit Primärindex und Sekundärindex zuzugreifen, hängen Sie einfach ein weiteres Trennzeichen an, gefolgt vom Sekundärindex. Die beiden Indizes dürfen dabei durchaus verschiedenen Formaten angehören.

Bei Bäumen setzt sich der Index aus den einzelnen Baumknoten zusammen, die den Pfad zum adressierten Knoten bilden. Die einzelnen Knoten werden im Pfad durch das für den Knoten gültige Trennzeichen, gefolgt von einem ”/”, getrennt. Falls ein Trennzeichen für mehrere Knoten hintereinander gilt, muss es nicht vor jedem ”/” stehen.

Hinweis

Die besondere Bedeutung der Trennzeichen ’@’, ’&’ und ’%’ macht diese zu Sonderzeichen, die geschützt werden müssen, wenn sie selbst in einem Namen auftauchen. Näheres zu diesem Thema finden Sie in Schützen von Sonderzeichen (*quoting*)(1025).

Negativer Index

Es ist fast immer möglich mit negativem Index die Zählung von hinten beginnen zu lassen.

SmartIDs: Einfache Indizes für TabPanels und Listen

Reiter in TabPanels müssten bei SmartIDs gemäß der obigen Syntax zum Beispiel über `#TabPanel:@Tab1` angesprochen werden, wobei `Tab1` der Name des Reiters ist. Alternativ kann hier die Abkürzung `#Tab:Tab1` verwendet werden. Wenn keine andere Komponente die SmartID `#Tab1` hat, kann der Reiter sogar einfach über `#Tab1` adressiert werden.

In Listen können die Listeneinträge gemäß obiger Syntax mit `#List:@EintragX` angesprochen werden. Als Abkürzung ist auch `#Item:EintragX` möglich. Dies

gilt auch für Dropdown-Listen von ComboBoxen. Wenn keine andere Komponente die SmartID `#EintragX` hat, kann der Listeneintrag wie bei den Tab-Reitern einfach über `#EintragX` adressiert werden.

Beide Abkürzungen sind komfortabel, die hohe Flexibilität hat aber ihren Preis bei der Performance. Wie hoch dieser ausfällt, hängt von vielen Faktoren ab, so dass die Abwägung Komfort/Performance im Einzelfall getroffen werden muss.

Beispiele

Komponente	Index	Beschreibung
Tabelle	`@Name&5`	Tabellenzelle in der sechsten Zeile und in der Spalte mit der Überschrift "Name". Komplette QF-Test ID der Komponente: `tableid@Name&5`
Liste	`&0`	Numerischer Index: erster Eintrag in einer Liste. Komplette SmartID: `#List:&0`
Liste	`@Europa`	Textindex: Listeneintrag mit dem Text "Europa". Komplette SmartID: Standardsyntax: `#List:@Europa` Verkürzte Syntax 1 (alternativ): `#Item:Europa` Verkürzte Syntax 2 (alternativ): wenn keine andere Komponente die SmartID `Europa` hat: `#Europa`
Baum	`@/Wurzel/A1/A1-2/B`	Textindex: Baumpfad, der alle Knoten über ihre Texte spezifiziert. Komplette QF-Test ID der Komponente: `treeid@/Wurzel/A1/A1-2/B`
Baum	`&/0/5/1/3/`	Numerischer Index: Baumpfad, der alle Knoten über ihnen numerischen Index spezifiziert. Komplette SmartID: `#Tree:&/0/5/1/3/`
Baum	`%/W.*/A.*`	Reguläre Ausdrücke für die Baumknoten.
Baum	`&/0@/Ast1%/B.*`	Gemischte Indizes: numerischer Index für den ersten, Textindex für den zweiten und regulärer Ausdruck für den dritten Knoten.
Tabelle	`&-1&-1`	Negative Indizes: unterste Zelle in der rechten Tabellenspalte.
TabPanel	`@Tab1`	Textindex: Adressierung über Reiterbeschriftung. Komplette SmartID: Standardsyntax: `#TabPanel:@Tab1` Verkürzte Syntax 1 (alternativ): `#Tab:@Tab1` Verkürzte Syntax 2 (alternativ), wenn keine andere Komponente die SmartID: `Tab1` hat: `#Tab1`

Tabelle 5.6: Indizes von Unterelementen

5.9.2 Adressierung mit QPath

Jedem QF-Test ID der Komponente Attribut (mit oder ohne Unterelement) in einem Event oder Check Knoten können ein oder mehrere Indizes der Form `@:ClassName<idx>` angehängt werden, wobei `<idx>` optional ist. Dies weist QF-Test an, zunächst die Zielkomponente (und ggf. das Unterelement) für den Teil des QF-Test ID der Komponente Attributs vor dem `@:` zu ermitteln und anschließend darin nach sichtbaren Komponenten der Klasse ClassName zu suchen. Ist `<idx>` angegeben, wird dies als 0-basierter Index in die Liste der sichtbaren Kandidaten interpretiert. Kein `<idx>` ist äquivalent zu `<0>`.

Die QPath-Syntax erwartet nach dem `@:` eine generische Klasse. Eine Übersicht der generischen Klassen finden Sie im Kapitel 61(1329). Falls sich die Komponente nicht mit einer generischen Klasse aufzeichnen lässt, muss man im QPath die vollständige Klassname angeben. Für JavaFX lauten einige davon zum Beispiel `ImageView`, `VBox`, `GridPane` oder `BorderPane`.

Das folgende Beispiel bezeichnet den zweiten `ImageView` auf der dritten Position einer Liste. `panelSecond.list&3@:javafx.scene.image.ImageView<1>`

5.9.3 Adressierung mit XPath und/oder CSS-Selektoren

Web
4.1+

XPath und CSS-Selektoren sind standardisierte Beschreibungen, um Komponenten in Webbrowsern anzusprechen. (Offizielle Spezifikationen: https://www.w3.org/TR/xpath/ und https://www.w3.org/TR/css3-selectors/).

QF-Test unterstützt die Komponentenbeschreibung via XPaths und CSS-Selektoren für HTML-Elemente, um eine leichtere Migration bestehender Web-Tests anderer Tools nach QF-Test zu ermöglichen.

Das Internet bietet bereits eine Vielzahl von Tutorials an, die die Komponentenerkennung mithilfe von CSS-Selektoren (z.B.: https://www.w3schools.com/cssref/css_selectors.asp) sowie die Komponentenerkennung mithilfe von XPaths (z.B.: https://www.w3schools.com/xml/xpath_syntax.asp) beschreiben. Insofern wird auf die Eigenheiten dieser Komponentenerkennung hier nicht mehr explizit eingegangen.

Verwendung in der QF-Test ID

Angenommen in QF-Test soll eine Web-Komponente anhand des XPath "$(xpath)" oder eines CSS-Selektors "$(css)" erkannt werden, so kann dies prinzipiell über mehrere Wege geschehen. Am einfachsten/schnellsten ist es meist, den XPath bzw. den CSS-Selektor im QF-Test ID der Komponente(776) Attribut eines beliebigen Event-Knotens zu spezifizieren. Hierzu wird die folgende Syntax benutzt:

```
genericHtml@:xpath=${quoteitem:$(xpath)}
```

```
genericHtml@:css=${quoteitem:$(css)}
```

bzw. gleichwertig:

```
 genericDocument@:xpath=${quoteitem:$(xpath)}
genericDocument@:css=${quoteitem:$(css)}
```

Die Syntax kann hierbei beliebig verschachtelt werden. Zum Beispiel kann man mithilfe von:

```
genericDocument@:xpath=${quoteitem:$(xpath)}@:css=${quoteitem:$(css)}
```

QF-Test dazu anweisen zuerst mithilfe eines XPath nach einer Komponente zu suchen und dann anschließend mithilfe eines CSS-Selektors nach einer Unterkomponente.

Hinweis

Bitte beachten Sie, dass die `@:xpath`/`@:css` aus nachvollziehbaren Gründen erwartet, dass die verwendete XPath-/CSS-Anweisung eine einzelne Komponente zurückliefert. Die Verwendung einer XPath-Anweisung die keine einzelne Komponente, sondern eine Zahl (Beispiel: `count(.//input[@id!='Google'])`) oder einen Boolean (Beispiel: `nilled($in-xml//child[1])`) zurückliefert, kann deshalb unter Umständen zu unerwartetem Verhalten führen.

Verwendung in Skripten

Das `rc`-Modul erlaubt in SUT-Skripten ebenfalls Web-Komponenten via XPath bzw. CSS-Selektoren zu finden.

```
com = rc.getComponent("genericHtml") # or rc.getComponent("genericDocument")
res = com.getByXPath(rc.getStr("xpath"))     # find subcomponent via xpath
res = com.getByCSS(rc.getStr("css"))         # find subcomponent via css
res = com.getAllByXPath(rc.getStr("xpath")) # find all subcomponent via xpath
res = com.getAllByCSS(rc.getStr("css"))      # find all subcomponent via css
```

Beispiel 5.1: Finden von Komponenten anhand von XPath / CSS-Selektoren in Skripten

Und um einen XPath zu benutzen, der keine Komponente(n) zurückliefert benutzen Sie bitte die `callJS`-Funktion:

```
node = rc.getComponent('genericDocument')
print node.callJS("""return document.evaluate("count(.//input[@id='Google'])",
document, null, 0, null).numberValue;""")
```

Beispiel 5.2: Beispiel für den Aufruf eines XPath-Selektors, der keine Komponente zurückliefert

Verwendung in Komponentenknoten

Innerhalb eines Komponente-Knotens kann QF-Test des Weiteren ebenfalls angewiesen werden, dass für die Komponentenerkennung ein XPath bzw. CSS-Selektor verwendet werden soll. Hierzu spezifiziert man ein wie folgt aussehendes Erkennungsmerkmal unter ”Weitere Merkmale”:

Status	Muss übereinstimmen
Regexp	Nein
Negieren	Nein
Name	qfs:item
Wert	@:xpath=${quoteitem:$(xpath)} oder @:css=${quoteitem:$(css)}

Abbildung 5.11: ”Weitere Merkmale”-Attribute für die Komponentenerkennung anhand von XPath oder CSS-Selektoren.

5.9.4 Adressierung mit Elemente Knoten

Ein Element wird durch zwei Dinge definiert: die Komponente, zu der es gehört, sowie einen Index innerhalb dieser Komponente. Der Parentknoten des Elements legt die Komponente fest. Der Index kann entweder eine Zahl sein oder ein Text. Numerische Indizes starten mit 0. So entspricht z.B. in einer `JList` Komponente das Element mit Index 1 dem zweiten Listeneintrag. Für Bäume sind einfache numerische Indizes nahezu unbrauchbar, da durch das Ein- und Ausklappen von Ästen die Indizes aller darunter liegenden Knoten verändert werden.

Ein Textindex definiert ein Element durch den Text, den es in der Oberfläche anzeigt. Ein Listenelement namens ”Eintrag1” in einer `JList` Komponente würde z.B. mit dem Textindex ”Eintrag1” aufgezeichnet. Die Textform ist flexibler als die numerische, kann aber Probleme bereiten, wenn die angezeigten Texte der Elemente in einer Komponente nicht eindeutig sind. In diesem Fall wird das erste passende Element angesprochen. Ein Textindex kann auch als regulärer Ausdruck (vgl. Abschnitt 49.3(1023)) angegeben werden. Auch in diesem Fall ist das Ziel das erste Element, das auf den regulären Ausdruck passt.

Die Option Format für Unterelemente(524) legt fest, welches Format beim Aufzeichnen von Elementen verwendet wird.

Beinahe alle Arten von Element haben nur einen Index. Dies reicht für die Zelle einer `JTable` Komponente nicht, da Tabellen zweidimensionale Strukturen sind. Entsprechend sind zwei Indizes nötig, um eine Zelle exakt zu beschreiben. Der erste, der Primärindex(937), legt die Tabellenspalte fest, der Sekundärindex(938) die Zeile.

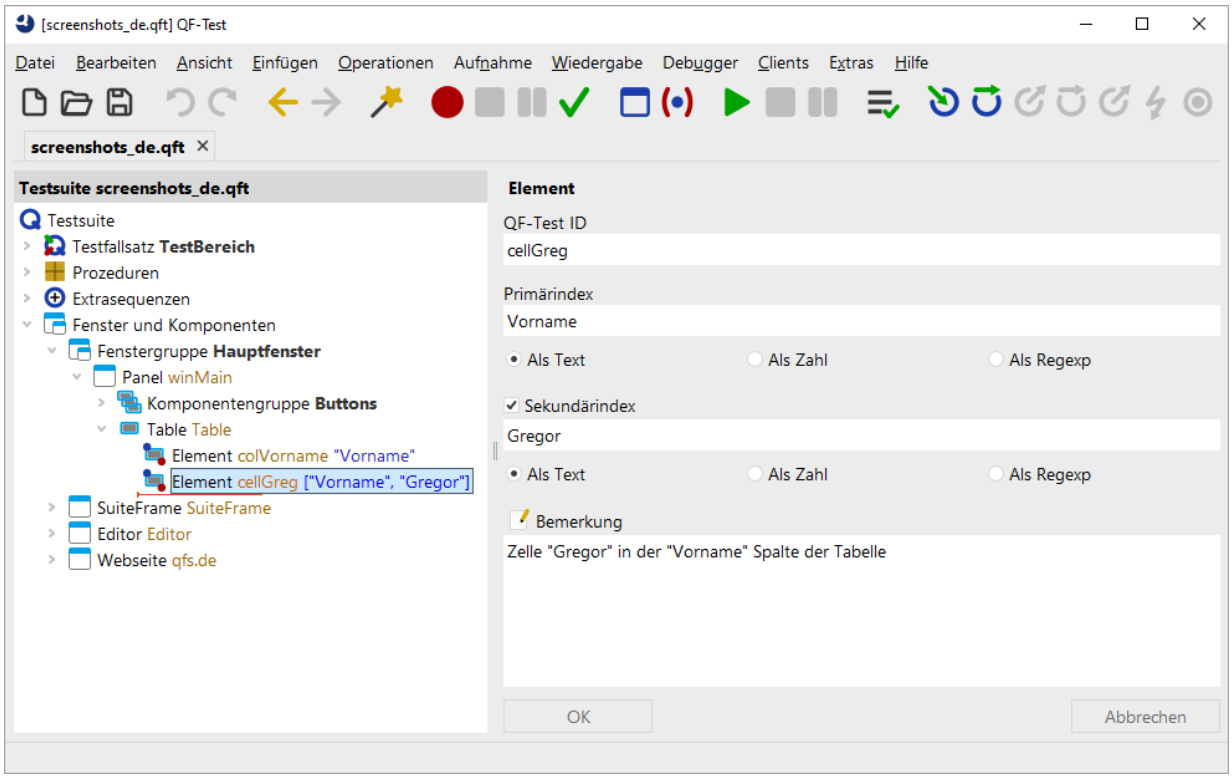


Abbildung 5.12: Ein Element für ein Tabellenfeld

Auch Baumknoten nehmen eine Sonderstellung ein. Wie oben beschrieben lässt sich die hierarchische Struktur nicht so leicht auf eine lineare Struktur abbilden. Außerdem treten in Bäumen oft Baumknoten mit den gleichen Namen auf. Wenn man dagegen die direkten und indirekten Parentknoten mit in den Namen einbezieht, lässt sich meistens Eindeutigkeit erreichen.

QF-Test verwendet eine besondere Syntax um Baumknoten darzustellen. Ein Index, der mit einem '/'-Zeichen beginnt, wird als Pfadindex interpretiert. Denken Sie dabei einfach an ein Dateisystem: Die Datei namens "/tmp/file1" kann entweder flach als "file1" dargestellt werden, was zu Konflikten mit anderen Dateien namens "file1" in anderen Verzeichnissen führen kann. Alternativ kann der volle und eindeutige Pfad "/tmp/file1" verwendet werden. QF-Test unterstützt auch numerische Indizes mit dieser Syntax: Ein numerischer Index der Form "/2/3" bezeichnet den vierten Kindknoten des dritten Kindknotens des Wurzelknotens. Eine kombinierte Form um z.B. mittels "/tmp/2" den dritten Knoten im "tmp" Knoten anzusprechen, ist derzeit nicht möglich.

Hinweis

Diese spezielle Syntax macht das '/'-Zeichen zu einem Sonderzeichen für Elemente in einer Baumkomponente. Taucht dieses Zeichen selbst in einem Namen auf, muss es daher geschützt werden. Näheres zu diesem Thema finden Sie in Abschnitt 49.5(1025).

Alles was in Abschnitt 5.5(78) zum QF-Test ID Attribut von Komponenten gesagt wurde, trifft

auch auf das QF-Test ID(937) Attribut eines Elements zu. Dieses Attribut muss eindeutig sein und wird von Events und Checks referenziert.

Wenn QF-Test die QF-Test ID eines Elements automatisch zuweist, erzeugt es diese, indem es die QF-Test ID der Komponente des Parentknotens nimmt und den Index (oder die Indizes) anhängt. Diese Art von QF-Test ID ist normalerweise gut zu lesen und zu verstehen. Leider ist sie aber auch die Quelle eines häufigen Missverständnisses: Wenn Sie den Index eines aufgezeichneten Elements ändern wollen, um auf ein anderes Element zu verweisen, dürfen Sie **nicht** das Attribut QF-Test ID der Komponente des Knotens ändern, der darauf verweist. Stattdessen müssen Sie den Primärindex des Element Knotens ändern.

5.10 Lösung von Problemen bei der Wiedererkennung

5.10.1 Zeitliche Synchronisierung

Wenn Sie Exceptions erhalten, weil eine Komponente nicht gefunden wurde, kann einer der Gründe sein, dass nicht lang genug auf die Komponente gewartet wurde. Ein Mausklick hat zwar eine gewisse Standardwartezeit, diese ist aber nicht immer ausreichend. Daher sollten Sie prüfen, ob es genügend Synchronisationspunkte gibt, wie Warten auf Komponente oder Check Knoten mit Wartezeiten, um die Testschritte nur dann auszuführen, wenn das SUT wirklich bereit dazu ist.

- Ein Warten auf Komponente Knoten kann genutzt werden, wenn eine Komponente neu auftaucht. Die maximale Wartezeit (in Millisekunden) wird im Attribut Wartezeit angegeben.
- Mittels Check Knoten kann auf eine Zustandsänderung von Komponenten gewartet werden. Geben Sie hier ebenfalls die maximale Wartezeit (in Millisekunden) im Attribut Wartezeit an.
- Manchmal ist es auch eine Schleife notwendig, in der auf die Zustandsänderung gewartet wird und, wenn noch nicht erfolgt, eine Aktion, zum Beispiel der Klick auf einen ”Aktualisieren” Button, ausgeführt wird.
- Auch auf die Änderung der Anzahl der Zeilen in einer Tabelle kann in einer Schleife gewartet werden.
- Viele Anwendungen verwenden Anzeigen, die Wartezeiten symbolisieren, zum Beispiel Fortschrittsbalken oder ”Eieruhren”. Hier kann zunächst auf das Erscheinen und dann auf das Verschwinden der Komponente gewartet werden.

Die Wartezeit Attribute geben eine maximale Wartezeit an. Sobald der gewünschte Zustand in der Anwendung erreicht ist, fährt QF-Test mit der Ausführung fort. Diese Wartezeiten können daher großzügig gewählt werden.

Eine Änderung der Optionen für Standardwartezeiten (Abschnitt 41.3.6(553)) sollte nur erfolgen, wenn generell längere Wartezeiten anwendungsweit Sinn machen.

Hinweis Als letzte Möglichkeit können Sie auch mit einer festen Verzögerung arbeiten. Bei Angaben im Attribut Verzögerung vorher/nachher wartet QF-Test die komplette angegebene Zeit. Verzögerung vorher/nachher sollten daher nur genutzt werden, wenn es gar keine für QF-Test erkennbare Zustandsänderung in der zu testenden Anwendung gibt, auf die gewartet werden kann.

5.10.2 Wiedererkennung

Wenn sich Ihr SUT in einer Weise verändert, die es QF-Test unmöglich macht, eine Komponente wiederzufinden, schlägt Ihr Test mit einer `ComponentNotFoundException`(958) fehl. Diese sollte nicht mit einer `UnresolvedComponentIdException`(966) verwechselt werden, welche durch Entfernen eines Komponente-Knotens aus der Testsuite oder dem Ändern des Attributs QF-Test ID der Komponente eines Event Knotens zu einer nicht vorhandenen QF-Test ID ausgelöst werden kann.

Video Es gibt zwei Videos, die die Behandlung einer `ComponentNotFoundException` ausführlich erklären:

- 'ComponentNotFoundException - einfacher Fall'
  https://www.qftest.com/de/yt/componentnotfoundexception-einfach-40.html
- 'ComponentNotFoundException - komplexer Fall'
  https://www.qftest.com/de/yt/componentnotfoundexception-komplex-40.html

Wenn Sie eine `ComponentNotFoundException` erhalten, führen Sie den Test erneut mit aktiviertem Test-Debugger aus, so dass der Test angehalten wird und Sie den Knoten, der das Problem verursacht hat, untersuchen können. Hier zahlt es sich aus, wenn QF-Test ID Attribute aussagekräftig sind, da Sie verstehen müssen, welche Komponente der Test anzusprechen versucht hat. Falls Sie sich gar keinen Reim darauf machen können, was der Sinn des betreffenden Knotens sein soll, deaktivieren Sie ihn und versuchen Sie, ob der Test ohne diesen Knoten durchläuft. Es könnte sich um einen Störeffekt handeln, der bei der Aufnahme nicht gefiltert wurde, und der gar nichts zum eigentlichen Test beiträgt. Grundsätzlich sollten Ihre Tests immer auf das Minimum an Knoten reduziert werden, mit denen sich der gewünschte Effekt erzielen lässt.

Falls der Knoten erhalten bleiben muss, werfen Sie als nächstes einen Blick auf das SUT, um zu sehen, ob die Zielkomponente aktuell sichtbar ist. Wenn nicht, müssen Sie Ihren Test entsprechend anpassen, um diese Situation zu behandeln. Ist die Komponente sichtbar, überprüfen Sie anhand des Bildschirmabbilds im Protokoll, ob das auch zum Zeitpunkt des Fehlers der Fall war, und versuchen Sie, den fehlgeschlagenen Knoten noch einmal als Einzelschritt auszuführen. Wenn die Ausführung nun klappt, haben Sie ein Problem mit dem Timing, das Sie durch den Einbau eines Warten auf Komponente[876] Knotens, eines Check Knotens mit Wartezeit oder einer anderen Warteaktion (siehe Zeitliche Synchronisierung[101]) lösen können.

Falls die Komponente sichtbar ist und die Wiedergabe kontinuierlich fehlschlägt, ist die Ursache eine Änderung an der Komponente oder einer ihrer Parent-Komponenten. Nun gilt es festzustellen, was sich geändert hat und wo. Nehmen Sie hierzu einen neuen Klick auf die Komponente auf und vergleichen Sie dann den alten und neuen Komponente-Knoten in der Hierarchie unterhalb von Fenster und Komponenten[942].

Hinweis Sie können vom Event Knoten direkt zum zugehörigen Komponente Knoten springen, indem Sie Strg-W drücken oder rechts-klicken und im Popup-Menü Komponente finden wählen. Sie können mittels Strg-Backspace oder Bearbeiten→Vorherigen Knoten anwählen wieder zurückspringen. Ein schlauer Trick ist, die zu vergleichenden Komponente Knoten durch Setzen von Marken zu kennzeichnen Bearbeiten→Marken, um sie leichter wiederzufinden.

Der Knackpunkt ist die Stelle, an der die Hierarchie der beiden Knoten verzweigt. Wenn sie unter verschiedenen Fenster Knoten angesiedelt sind, liegt der Unterschied in den betreffenden Fenstern selbst. Andernfalls gibt es einen gemeinsamen Vorgänger direkt oberhalb der Verzweigung. Der entscheidende Unterschied ist dann in den jeweiligen Knoten direkt unterhalb dieses gemeinsamen Vorgängers zu finden. Wenn Sie die Stelle mit der Unterscheidung gefunden haben, vergleichen Sie die Attribute der betreffenden Knoten von oben nach unten und suchen Sie nach Abweichungen.

Hinweis Sie können mittels Ansicht→Neues Fenster... ein weiteres QF-Test Fenster öffnen und damit die Detailansichten der beiden Knoten nebeneinander platzieren.

Die einzigen Unterschiede, die immer zu einem Fehler bei der Wiedererkennung führen, sind Änderungen der Attribute Klasse oder Name. Abweichungen bei Merkmal, Struktur oder Geometrie können üblicherweise kompensiert werden, sofern sie sich nicht häufen.

Eine Änderung der Klasse sollte bei der Verwendung von Generische Klassen[1329] kaum vorkommen. Die Verwendung von generischen Klassen bietet einige Vorteile, wird bei Web-Anwendungen aber manchmal erst nach der Erstellung erster Tests eingeführt (siehe Verbesserte Komponentenerkennung mittels `CustomWebResolver`[1077]). In diesem Fall müssen Sie das Klasse Attribut der bereits vorhandenen Komponente-Knoten an diese Änderung anpassen.

Auch der Komponentenbezeichner[66] kann sich ändern. Falls die Änderung beabsichtigt

zu sein scheint, z.B. die Korrektur eines Rechtschreibfehlers, können Sie das Name Attribut entsprechend anpassen. Wahrscheinlicher ist, dass es sich um einen automatisch generierten Komponentenbezeichner(66) handelt, der sich jederzeit wieder ändern kann. Auch hier kann es Sinn machen, das Problem mit den Entwicklern diskutieren und entwicklungsseitig eine Lösung zu finden. Ansonsten kann bei Web-Anwendungen der Name über den Der CustomWebResolver installieren Knoten(1082) beeinflusst werden, speziell über die Kategorien `autoIdPatterns` und `customIdAttributes`. Bei allen Technologien kann der Name mit Hilfe eines `NameResolvers`, wie in Abschnitt 54.1.7(1163) beschrieben, beeinflusst werden. Er kann ganz unterdrückt oder auf relevante Teile reduziert werden.

Änderungen am Attribut Merkmal sind insbesondere für Fenster Knoten nicht ungewöhnlich. Dort entspricht das Merkmal dem Titel des Fensters. Kombiniert mit einer signifikanten Änderung der Geometrie kann das zum Scheitern der Wiedererkennung führen. Dies kann durch Anpassen des Merkmal Attributs an die neuen Gegebenheiten, oder - bevorzugt - durch Verwendung eines regulären Ausdrucks (vgl. Abschnitt 49.3(1023)), der alle Varianten abdeckt, behoben werden.

Abhängig von Art und Umfang der Änderungen gibt es zwei grundsätzliche Möglichkeiten zur Korrektur:

- Passen Sie die Attribute des alten Knoten an und entfernen Sie die neu aufgenommenen Knoten. Wenn die Änderungen im SUT nicht zu groß waren und die Komponentenerkennung noch funktioniert, können Änderungen auch über die QF-Test Funktion Komponenten aktualisieren(105) automatisiert durchgeführt werden.

- Behalten Sie die neuen Knoten und entfernen Sie den alten. Hierzu müssen Sie zunächst sicherstellen, dass alle Knoten, die auf die alte Komponente verweisen, auf die neue QF-Test ID geändert werden. Dies lässt sich durch einen kleinen Trick erreichen: Ändern Sie die QF-Test ID des alten Komponente-Knotens auf die QF-Test ID des neuen. QF-Test beschwert sich zunächst, dass die QF-Test ID nicht eindeutig ist, was Sie ignorieren können, und bietet dann an, alle Verweise zu aktualisieren, was Sie mit ”Ja” bestätigen müssen. Anschließend können Sie den alten Knoten entfernen.

Hinweis

Die automatische Anpassung der Verweise in anderen Testsuiten funktioniert nur, wenn diese zum selben Projekt gehören oder das Attribut Abhängige Dateien (umgekehrte Includes)(597) des Testsuite Knotens korrekt gesetzt ist.

5.11 Bereinigung und Wartung des Komponentenbaums

Im Laufe der Testerstellung können sich zum einen ungenutzte Komponenten im Komponentenbaum ansammeln. Hier können Sie ab und zu den Komponentenbaum bereinigen(105). Zum anderen können sich die Wiedererkennungsmerkmale durch Änderungen in der Applikationsoberfläche ändern. Bevor die Änderungen nach mehreren Oberflächenänderungen so stark werden, dass die Wiedererkennung bricht, macht es Sinn, in den betroffenen Fenstern und Dialogen Komponenten aktualisieren(105) durchzuführen.

5.11.1 Komponentenbaum bereinigen

Immer wenn eine Sequenz aufgezeichnet wird, werden für die Komponenten, die noch nicht in der Testsuite vorhanden sind, neue Knoten angelegt. Wird die Sequenz später gelöscht, bleiben die Komponenten erhalten, daher haben Komponenten eine gewisse Tendenz sich anzusammeln.

Das Kontextmenü für Fenster und Komponente-Knoten hat zwei Einträge namens Ungenutzte Komponenten markieren... und Ungenutzte Komponenten entfernen, die jene Komponenten markieren oder ganz entfernen, auf die sich kein anderer Knoten in dieser Testsuite mehr bezieht.

Vorsicht ist geboten, falls Sie Variablen in QF-Test ID der Komponente Attributen verwenden, da die Automatik diese nicht erkennt.

Werden Komponenten aus anderen Testsuiten referenziert, sollte diese zum selben Projekt gehören oder das Attribut Abhängige Dateien (umgekehrte Includes)(597) des Testsuite Wurzelknotens korrekt gesetzt sein.

5.11.2 Komponenten aktualisieren

Es ist kaum zu vermeiden, dass sich die Komponenten des SUT im Lauf der Zeit verändern. Wie beschrieben stellt dies kein großes Problem dar, sofern Bezeichner konsequent eingesetzt werden, da QF-Test dann mit fast jeder Art von Veränderung zurechtkommt.

Ohne Bezeichner summieren sich Änderungen mit der Zeit und können einen Punkt erreichen, an dem die Wiedererkennung fehlschlägt. Um dieses Problem zu umgehen, sollten Sie die Komponenten in QF-Test von Zeit zu Zeit an den aktuellen Stand des SUT anpassen. Dies kann mit Hilfe des Menüeintrags Komponente(n) aktualisieren in dem Kontextmenü geschehen, das Sie nach einem Klick mit der rechten Maustaste auf einen

beliebigen Knoten unterhalb des Fenster und Komponenten(942) Knotens erhalten.

Hinweis Diese Funktion kann sehr viel Information auf einmal ändern, daher ist es schwierig zu beurteilen, ob alles wie erwartet funktioniert hat oder ob eine Komponente falsch erkannt wurde. Erstellen Sie daher immer eine Sicherheitskopie, bevor Sie viele Komponenten aktualisieren. Außerdem sollten Sie Fenster für Fenster vorgehen und darauf achten, dass die Komponenten, die Sie aktualisieren wollen, im SUT sichtbar sind (mit Ausnahme der Menüeinträge). Stellen Sie nach jedem Schritt sicher, dass die Tests immer noch sauber laufen.

Vorausgesetzt, dass eine Verbindung zum SUT besteht, erscheint bei Aufruf dieser Funktion folgender Dialog:

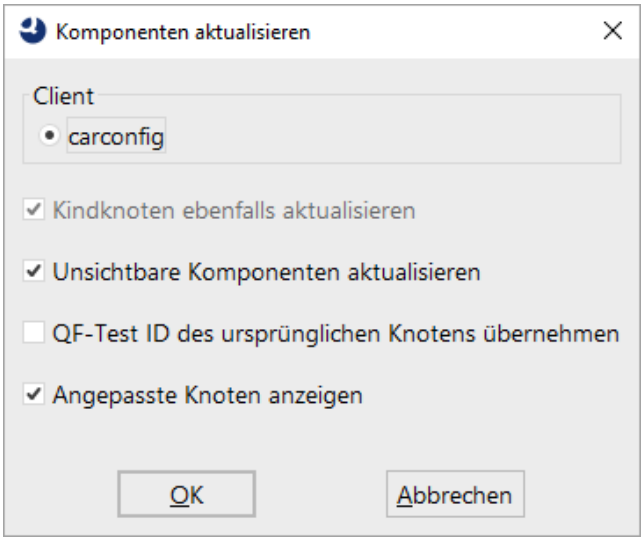


Abbildung 5.13: Komponenten aktualisieren Dialog

Sind Sie mit mehreren SUT Clients verbunden, müssen Sie zunächst einen für die Aktualisierung auswählen.

Legen Sie fest, ob Sie nur den/die selektierten Komponente Knoten selbst oder auch dessen/deren Kindknoten aktualisieren wollen.

Sie können auch Komponenten einbeziehen, die im SUT momentan nicht sichtbar sind. Dies ist vor allem für Menüeinträge hilfreich.

Die QF-Test ID(931) eines aktualisierten Knotens wird beibehalten, wenn "QF-Test ID des ursprünglichen Knotens übernehmen" gewählt ist. Andernfalls erhält der Knoten eine von QF-Test generierte QF-Test ID, sofern sinnvolle Informationen zur Verfügung stehen. Andere Knoten, die sich auf diese QF-Test ID beziehen, werden automatisch angepasst. QF-Test prüft dabei auch Abhängigkeiten in den Testsuiten, die zum selben Projekt gehören oder die im Attribut Abhängige Dateien (umgekehrte Includes)(597) des Testsuite(595)

Knotens aufgeführt sind. Diese Testsuiten werden automatisch geladen und indirekte Abhängigkeiten ebenfalls aufgelöst.

Hinweis In diesem Fall werden die geänderten Testsuiten automatisch geöffnet, damit die Änderungen gespeichert oder rückgängig gemacht werden können.

Nach der Bestätigung mit ”OK” wird QF-Test versuchen, die betreffenden Komponenten im SUT zu lokalisieren und aktuelle Informationen einzuholen. Komponenten, die nicht gefunden werden können, werden übersprungen. Anschließend werden die Komponente-Knoten an die aktuelle Struktur des GUI im SUT angepasst, was auch zur Folge haben kann, dass Knoten verschoben werden.

Hinweis Für große Hierarchien von Komponenten kann diese äußerst komplexe Operation einige Zeit in Anspruch nehmen, in extremen Fällen sogar einige Minuten.

Diese Funktion ist besonders nützlich, wenn Bezeichner zum ersten Mal im SUT verwendet werden. Wenn Sie bereits einige Tests erstellt haben, bevor Sie die Entwickler überzeugen konnten, Bezeichner zu vergeben, können Sie hiermit die Bezeichner in Ihre Komponenten einbinden und gleichzeitig die QF-Test IDs anpassen. Dies funktioniert am besten, wenn Sie eine SUT Version bekommen können, die bis auf die Bezeichner mit der vorherigen Version übereinstimmt.

Hinweis Sehr wichtiger Hinweis: Das Aktualisieren von ganzen Fenstern oder Hierarchien von Komponenten ab einer gewissen Größe führt oft zu dem Versuch, Komponenten zu aktualisieren, die im Moment nicht vorhanden oder unsichtbar sind. In so einem Fall ist es sehr wichtig, falsch-positive Treffer für diese Komponenten zu verhindern. Sie können dies erreichen, indem Sie vorübergehend die ’Bonus’ und ’Herabsetzung...’ Optionen zur Wiedererkennung (Abschnitt 41.3.4(547)) verändern. Setzen Sie insbesondere die ’Herabsetzung für Merkmal’ auf einen Wert unterhalb des Wertes der ’Mindestwahrscheinlichkeit’, also z.B. auf 49, sofern Sie ansonsten die Standardwerte verwenden. Vergessen Sie nicht, die ursprünglichen Werte hinterher wiederherzustellen.

Wenn Sie die Einstellung der Optionen Gewichtung von Namen (Wiedergabe)(548) und Gewichtung von Namen (Aufnahme)(520) ändern müssen, zum Beispiel weil sich Bezeichner von Komponenten wider Erwarten als nicht eindeutig herausgestellt haben, ändern Sie zunächst nur die Option für die Aufnahme. Wenn die Aktualisierung beendet ist, ziehen Sie die Option für die Wiedergabe entsprechend nach.

5.12 Komponenten untersuchen

3.1+

In bestimmten Fällen ist es hilfreich, zusätzliche Informationen über Komponenten zu erhalten, neben den in der Sektion Fenster und Komponenten(942) abgespeicherten, oder die dort abgespeicherten Informationen direkt im Zusammenspiel mit der Applikation zu betrachten.

Besonders wichtig ist dies während des ”Einlernens von Komponenten” bei Web-Anwendungen. Dieser Vorgang sollte vor der Erstellung von Tests erfolgen. Er ist in Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077) beschrieben. Hierzu steht der UI-Inspektor zur Verfügung.

Bei der Arbeit mit Skripten ist es manchmal hilfreich, wenn man sich eine Liste der Methoden eines GUI-Elements anzeigen lassen kann.

5.12.1 Methoden anzeigen

Jedes GUI-Objekt besitzt bestimmte (öffentliche) Methoden und Felder die man im einem SUT-Skript(720) verwenden kann, sobald man Zugriff auf ein Objekt hat (siehe Abschnitt 11.3.4(195)). Um sie anzuzeigen wählt man entweder den Eintrag Methoden für Komponente anzeigen... aus dem Kontextmenü eines Knotens unterhalb des Fenster und Komponenten(942) Zweigs oder klickt im Komponentenaufnahme-Modus mit der rechten Maustaste auf die Komponente (siehe auch Abschnitt 4.5(44)).

Web Die Methoden und Felder, die für HTML-Elemente in einem Browser angezeigt werden, können nicht direkt mit dem von `rc.getComponent()` zurückgelieferten Objekt verwendet werden. Es sind JavaScript-Methoden und -Eigenschaften, die in `callJS` eingebettet werden müssen (vgl. Abschnitt 54.10(1253)).

5.12.2 UI-Inspektor

7.0+

Der UI-Inspektor zeigt die Hierarchie der Komponenten im Client und deren Eigenschaften an. Neben der Nachverfolgung von Problemen bei der Komponentenaufnahme oder der Wiedererkennung erleichtert er durch die in der Detailansicht dargestellten Informationen auch die Erstellung von Resolvern.

Video Im April 2024 fand ein Spezialwebinar zum Thema statt. Hier geht es zum

Videomitschnitt des Spezialwebinars
https://qftest.com/de/yt/uiinspektor-spezialwebinar.html

auf unserem QF-Test YouTube-Kanal.

Der UI-Inspektor steht für Android und Web zur Verfügung. Ab QF-Test Version 7.1 wird zusätzlich noch Windows und Swing/AWT unterstützt, und ab Version 7.1.3 wird ebenfalls FX unterstützt.

Die Darstellung der Knoten im Komponentenbaum bietet bereits einen Überblick über die wichtigsten Informationen. Wird die Klasse(62) in blauer Schrift dargestellt, so wird diese Komponente als interessant betrachtet. Dies wiederum legt fest, ob für diese

Komponente ein Komponente[930] Knoten angelegt wird. Kann eine generische Klasse (siehe Generische Klassen[1329]) bestimmt werden, so wird zuerst die generische Klasse zusätzlich fett visualisiert und dann die ursprüngliche Klasse dahinter in Klammern gesetzt. Standardmäßig werden alle generischen Klassen als interessant betrachtet. Ist eine Komponente unsichtbar, so wird diese in einem Grauton dargestellt.

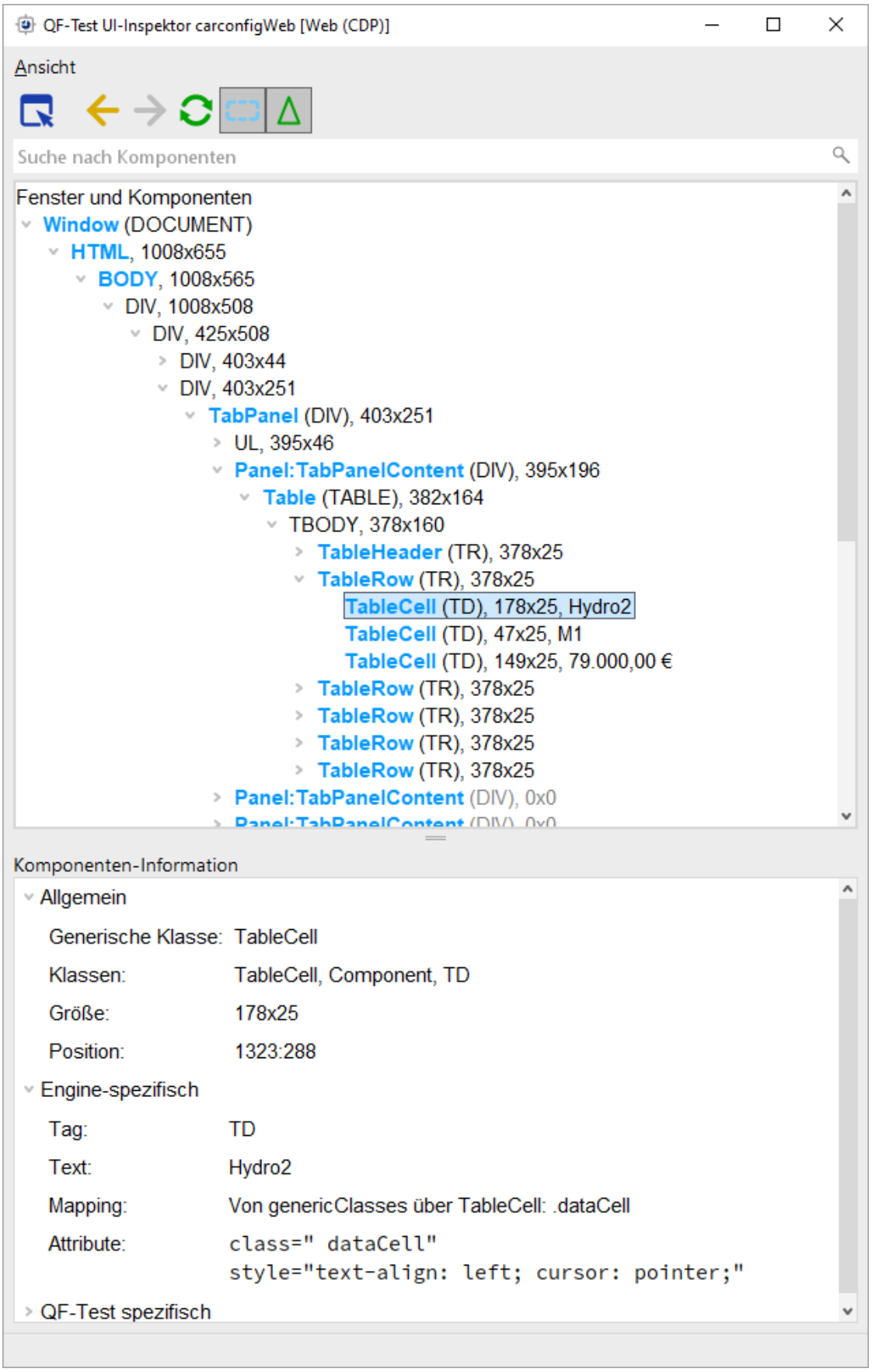


Abbildung 5.14: Beispiel für Allgemeine Informationen

UI-Inspektor öffnen

Um den UI-Inspektor zu öffnen, haben Sie folgende Möglichkeiten:

- Über den Menüeintrag Clients→Inspektor anzeigen.
- Über den Menüeintrag Im Inspektor anzeigen aus dem Kontextmenü bei der Komponentenaufnahme.
- Über das Kontextmenü Im Inspektor anzeigen eines Komponente-Knotens oder eines Knotens, welcher eine QF-Test ID der Komponente oder eine SmartID[81] enthält.
- Direkt vom SUT aus über (konfigurierbare) Tastaturkürzel. Standardmäßig Umschalt-Strg-F11 für Windows/Linux bzw. ^⇧-F11 für Mac. Siehe Abschnitt C.2[1439].

Web

- Über den Button Inspektor im CustomWebResolver installieren[902] Knoten.

Android

- Im Aufnahmefenster mithilfe des Fadenkreuz-Buttons in der Toolbar, siehe QF-Test Android-Aufnahmefenster[266].

Toolbar

Die Buttons in der Toolbar haben folgende Bedeutung:

Komponente auswählen, um sie zu untersuchen. Während der Komponentenauswahl sind Aufnahme- und Check-Funktion deaktiviert, Aktionen werden nicht an das SUT weitergegeben. So können Komponenten im SUT per Mausklick mithilfe der im UI-Inspektor dargestellten Informationen untersucht werden.

Navigieren im Komponentenbaum. Zusammen mit dem Inspektor-Modus wird auch die Historienfunktion aktiviert. Diese speichert die Selektionen in UI-Inspektor und Client und man kann rückwärts und vorwärts in diesen navigieren.

Komponentenbaum aktualisieren. Ein Aktualisieren ist erforderlich, wenn sich Komponenten geändert haben oder Resolver installiert wurden.

Unsichtbare Komponenten im Komponentenbaum anzeigen. Komponenten mit sichtbaren Kindern werden immer angezeigt.

Geometrie-Informationen im Komponentenbaum anzeigen.

UI-Inspektor-Detailansicht

Die Detailansicht gibt einen Überblick über die wichtigsten Eigenschaften einer Komponente. Die Ansicht ist in drei Teilbereiche untergliedert:

Allgemein

Dieser Bereich umfasst die grundlegenden Eigenschaften einer Komponente, etwa ihre Klasse.

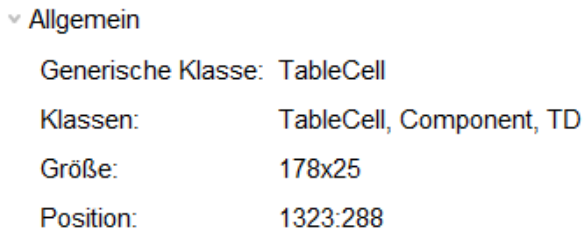


Abbildung 5.15: Allgemeine Informationen

Engine-spezifisch

Die Engine-spezifischen Details umfassen eher technische Informationen zu einer Komponente. Die zur Verfügung stehenden Information unterscheiden sich je nach verwendeter Technologie.

Web

Bei Web-Anwendungen sind es Informationen über das DOM-Element, etwa dessen HTML-Tag, die HTML-Attribute und der auslesbare Text.

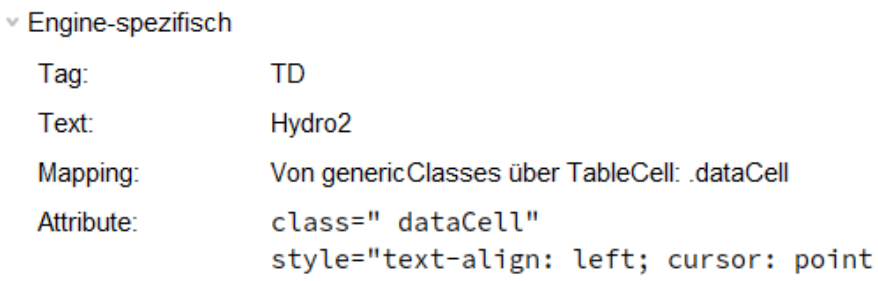


Abbildung 5.16: Web-spezifische Informationen

Android

In Android sind es Informationen über die Android-Komponente und beinhalten die inhaltliche Beschreibung, Ressourcen-ID, Paketname sowie Informationen über den Fenster-Typ und dessen Anordnung.

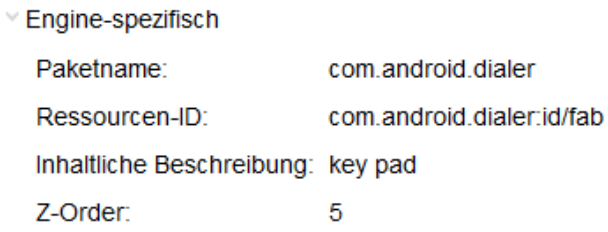


Abbildung 5.17: Android-spezifische Informationen

Windows-Tests

In Windows-Anwendungen sind es Informationen über das Automation Element und beinhalten die wichtigsten UI Automation-Eigenschaften. Detaillierte Beschreibungen dieser Informationen finden Sie in Abschnitt 54.12.1(1271).

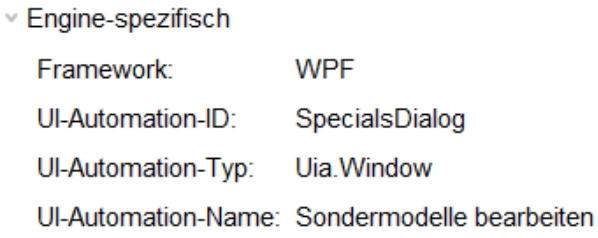


Abbildung 5.18: Windows-spezifische Informationen

Swing

In Swing-Anwendungen enthält der UI-Inspektor Informationen über die Swing-Komponente und die wichtigsten *Accessible Properties* sowie den Namen, Tooltip und die Clientproperties.

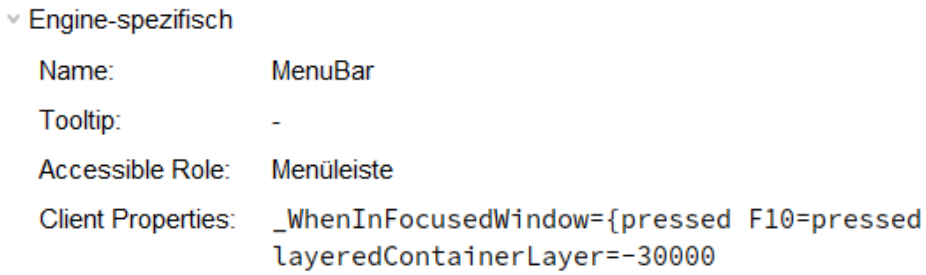


Abbildung 5.19: Swing-spezifische Informationen

JavaFX

In FX-Anwendungen enthält der UI-Inspektor Informationen über die FX-Komponente und die wichtigsten *Accessible Properties* sowie den Style, ID, Tooltip und die Clientproperties.

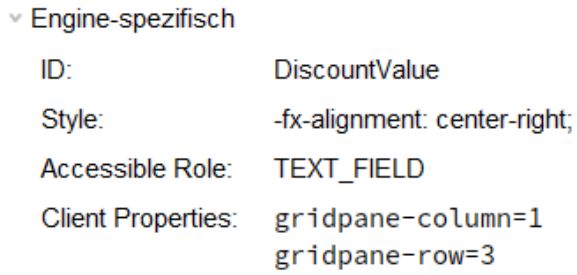


Abbildung 5.20: FX-spezifische Informationen

SWT

In SWT-Anwendungen enthält der UI-Inspektor Informationen über das SWT-Widget, wie benutzerdefinierte Daten, Tooltip, Schriftart, Sichtbarkeit und Aktivierungsstatus.

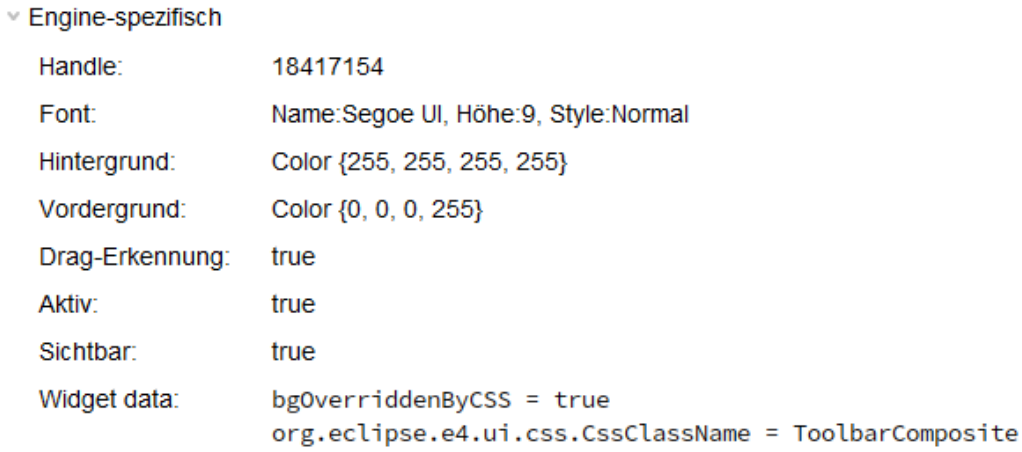


Abbildung 5.21: SWT-spezifische Informationen

QF-Test spezifisch

Diese Informationen sind an den bekannten Komponente-Knoten[78] Knoten angelehnt. Sie eignen sich gut, um zu überprüfen, ob ein erstellter Resolver wie gewünscht funktioniert.

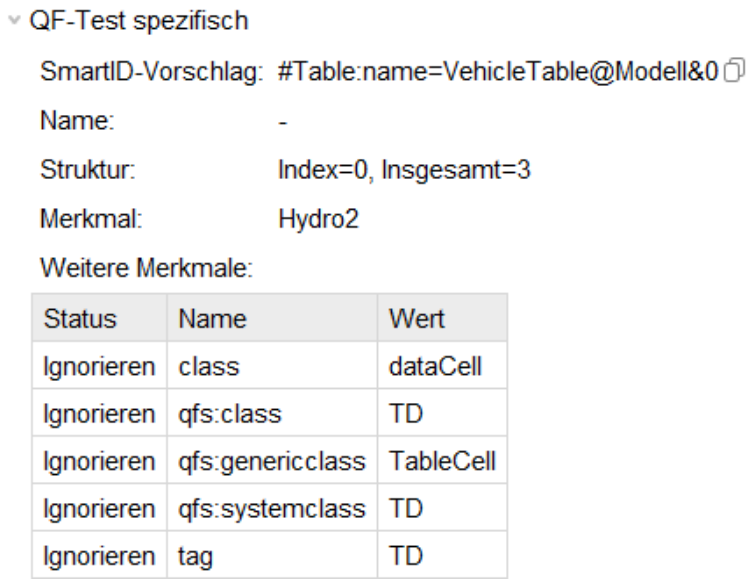


Abbildung 5.22: QF-Test spezifische Informationen

Die Detailansicht kann zusätzlich die Informationen zu zwei verschiedenen Komponenten nebeneinander darstellen, so dass man diese leicht vergleichen kann. Dazu klickt man mit der rechten Maustaste auf einen anderen Knoten im Komponentenbaum und wählt Vergleichen aus dem Kontextmenü. Über Vergleichsknoten zurücksetzen aus dem Kontextmenü oder mithilfe des Schließen-Buttons der Detailansicht wird man die Vergleichsansicht wieder los.

Kapitel 6

Variablen

Video

Es gibt ein

kurzes Überblicksvideo zu Variablen
https://www.qftest.com/de/yt/variablen.html

, welches die wichtigsten Punkte dieses Kapitels abdeckt.

Variablen sind von zentraler Bedeutung, wenn es darum geht, die Wiederverwendbarkeit einer Testsuite zu steigern. Zum Einsatz kommen sie vor allem beim Aufruf von Prozeduren(672). Variablen sind aber auch in vielen anderen Situationen hilfreich.

Variablen können in allen Attributen mit Eingabefeldern verwendet werden. Viele Checkboxen können über den links davor stehenden Button  in Eingabefelder umgewandelt werden, dann können dort Wahrheitswerte entweder direkt oder als Variablen eingetragen werden.

6.1 Variablenreferenzen

Es gibt mehrere Möglichkeiten Variablen zu referenzieren:

6.1.1 Referenzierung einfacher Variablen

`$(Variablenname)` gibt den Wert einer Variablen zurück.

Ist der Wert keine Zeichenkette, wird aber als Teil einer Zeichenkette ausgewertet oder das Ergebnis als Text genutzt, so wird automatisch die Text-Darstellung des Wertes verwendet.

6.1.2 Referenzierung von Gruppenvariablen

`${Gruppe:Name}` greift auf eine Variable in einer Variablengruppe zu. Damit können zum Beispiel Variablen in einer Gruppe angesprochen werden, in die Daten aus einer externen Quelle (siehe Externe Daten(126)) geladen wurden. Einige Gruppen, zum Beispiel `qftest`, `env` und `system`, sind immer definiert und haben besondere Bedeutungen (vgl. Spezielle Gruppen(127)).

Auch hier wird kontextabhängig automatisch die Text-Darstellung des Wertes zurückgegeben.

6.1.3 Referenzierung von Variablen in Skripten und Skriptausdrücken

Für Skripte und Skriptausdrücke(189) (zum Beispiel `$[Jython-Ausdruck]` oder die Bedingung eines If(693)-Knotens), ist der Zugriff auf Variablen, die über QF-Test Knoten angelegt wurden, in Variablen(191) beschrieben.

Die Runcontext-Methoden `rc.get*`

Es gibt viele Methoden im Runcontext-Modul, mit denen auf Variablen zugegriffen werden kann, zum Beispiel `rc.getStr`, `rc.getInt`, `rc.getNum`, `rc.getBool` oder `rc.getObj`. Eine detaillierte Beschreibung dieser Methoden finden Sie in Die API des Runcontexts(1030).

Das Runcontext-Feld `rc.vars`

Der Runcontext ermöglicht mit dem `Map`-änlichen Objekt `rc.vars` einen einfachen Zugriff auf die aktuellen Werte der QF-Test Variablen: Als Alternative für `rc.getObj('Variablenname')` kann im Script direkt `rc.vars.Variablenname` verwendet werden. Wird über diesen Ausdruck ein Wert zugewiesen, so entspricht dies dem Setzen einer lokalen Variable, analog zu `rc.setLocal`.

Das Runcontext-Feld `rc.groups`

Analog zu `rc.vars` ist es möglich, mit `rc.groups` auf Gruppenvariablen zuzugreifen: Anstelle von `rc.getObj('Gruppe','Name')` oder `rc.getObj('qftest','dir.version')` kann `rc.groups.Gruppe.Name` bzw. `rc.groups.qftest.dir.version` genutzt werden. Wird diesem Ausdruck ein Wert zugewiesen, so entspricht dies dem Setzen eines Wertes in einer Gruppe, analog zu `rc.setGroupObject`. Dabei ist zu beachten, dass Elementen in speziellen Gruppen unter Umständen kein Wert zugewiesen werden kann. In diesem Fall wird dann eine `ReadOnlyPropertyException`(962) geworfen.

`$(`*`Variablenname`*`)` und `${`*`Gruppe`*`:`*`Name`*`}` in Jython-Skripten

In Jython-Skripten und -Skriptausdrücken können QF-Test Variablen mit der gleichen Syntax wie in normalen Knoten referenziert werden. Da das Jython-Skript technisch eine Zeichenkette ist, wird bei der Expansion der Textwert der jeweiligen Variable in den Jython-Code eingefügt.

Diese diese Referenzierungsart wird nicht empfohlen. Wenn der Variablenwert Zeichen wie Rückstriche (\) oder Zeilenumbrüche enthält, kann dies zu unerwünschten Effekten führen.

6.2 Ermittlung des Wertes einer Variablen

Um zu erklären, wie und warum Variablen an verschiedenen Stellen definiert werden, müssen wir zunächst darauf eingehen, wie der Wert einer Variablen ermittelt wird, wenn diese bei einem Testlauf referenziert wird.

Jede Variablendefinition wird auf einem von zwei Stapeln abgelegt. Der primäre Stapel ist für direkte Zuordnungen bestimmt, während der sekundäre Stapel Definitionen von Rückfallwerten aufnimmt. Wenn der Wert einer Variable zum Beispiel mittels `$(...)` angefordert wird, durchsucht QF-Test zunächst den primären Stapel von oben nach unten nach einer passenden Definition. Wird keine solche gefunden, wird die Suche im sekundären Stapel fortgesetzt, ebenfalls von oben nach unten. Bleibt auch diese Suche erfolglos, wird eine `UnboundVariableException`(962) geworfen, sofern Sie nicht mittels der speziellen Syntax `${default:varname:defaultvalue}` bzw. im Skript über `rc.withDefault('defaultvalue').getStr('varname')` einen Defaultwert angegeben haben (vgl. Spezielle Gruppen(127)).

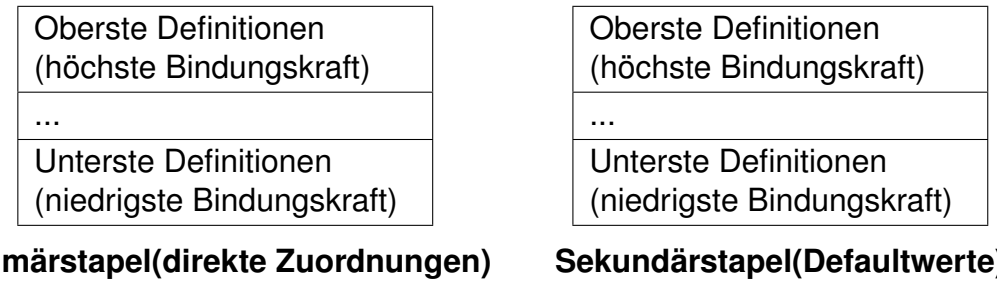


Abbildung 6.1: Direkte Zuordnungen und Defaultwerte

Der Mechanismus unterstützt rekursive bzw. selbstbezogene Variablendefinitionen. Zum Beispiel bewirkt das Setzen der Variable `classpath` auf den Wert `irgendein/pfad/archiv.jar:$(classpath)` eine Erweiterung des Wertes einer

Definition von `classpath` mit geringerer Bindungskraft. Existiert keine solche Definition, wird eine `RecursiveVariableException`(962) ausgelöst.

6.3 Definition von Variablen

Variablen können an verschiedenen Stellen definiert werden.

Variablendefinitionstabellen
Zweispaltige Tabellen(20) werden zum Beispiel im Prozeduraufruf(675) verwendet, um die zu übergebenden Parameternamen und -werte zu definieren oder in einem Prozedur(672)-Knoten, um die Rückfallwerte zu setzen. In jeder Zeile kann eine Variable mit Name und Wert angegeben werden. In vielen weiteren Knoten wie zum Beispiel Testsuite(595), Testfallsatz(606) und Testfall(599) können Variablen ebenfalls in Tabellen definiert werden.

Rückgabewert einer Prozedur
Bei Prozeduraufrufen kann die gerufene Prozedur einen Wert zurückliefern. Dieser wird dann der Variablen mit dem in Variable für Rückgabewert(676) eingetragenen Namen im Prozeduraufruf-Knoten zugewiesen. In der gerufenen Prozedur kann dabei der Typ des zurückgegebenen Objektes mit dem Attribut Expliziter Objekttyp(679) des Return(678)-Knotens festgelegt werden.

Ergebnis eines Checks
Eine der möglichen Ergebnisbehandlungen in einem Check-Knoten ist die Zuweisung des Ergebnisses an eine Variable, deren Namen im Attribut Variable für Ergebnis angegeben wurde, zum Beispiel bei einem Check Boolean(812)-Knoten.

Rückgabewert eines Auslesen-Knotens
Die Auslesen-Knoten, zum Beispiel Text auslesen(839), weisen den erhaltenen Wert einer Variablen zu, deren Name im Attribut Variablenname angegeben ist.

Variable setzen-Knoten
Über den Variable setzen(871)-Knoten können Variablen ebenfalls definiert werden. Mit dem Attribut Expliziter Objekttyp(874) kann dabei der Typ des zurückgegebenen Objektes festgelegt werden.

Skript-Knoten
In Skripten können über die Methoden `rc.setLocal`, `rc.setGlobal`, `rc.setLocalJson` etc. Variablen gesetzt werden, die dann in QF-Test Knoten verwendet werden können. `setGroupObject` kann genutzt werden, um Variablen in einer Variablengruppe zu setzen. Für weitere Informationen siehe Skripting(186) oder auch Die API des Runcontexts(1030).

Optionendialog

Im Optionendialog können Variablen in der Rubrik ”Variablen” gesetzt und geändert werden (vgl. Variablen[592]). Dies ist besonders für globale, System- und Kommandozeilen-Variablen interessant.

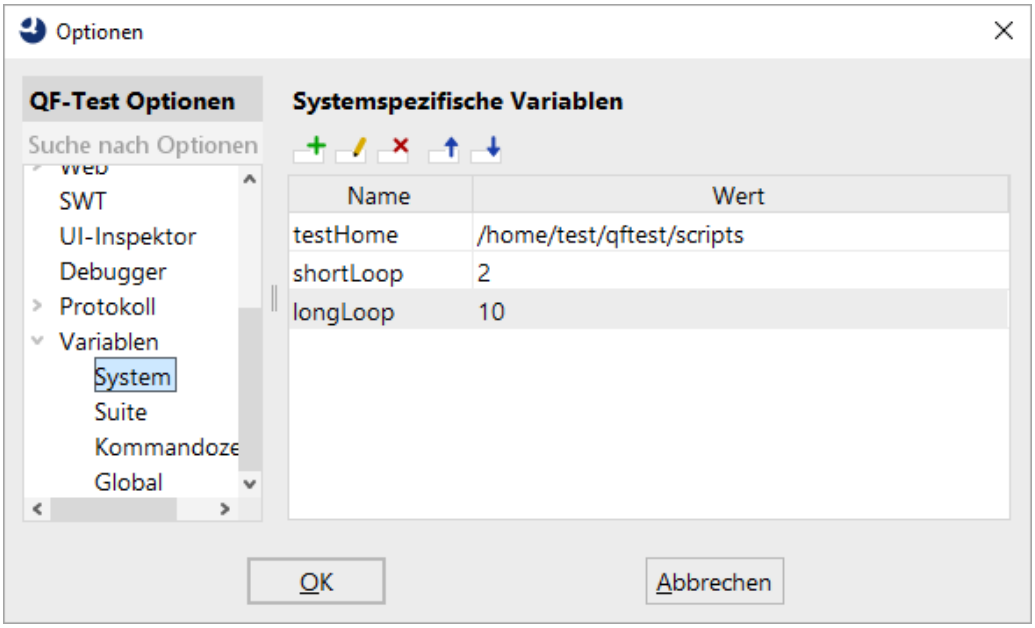


Abbildung 6.2: Definition von Systemvariablen im Optionendialog

6.4 Variablenebenen

Variablen können auf unterschiedlichen Ebenen definiert werden. Wie bereits beschrieben gibt es die grundlegende Unterscheidung bei der Auswertungsreihenfolge der Variablendefinitionen zwischen Primär- und Sekundärstapel. In beiden Stapeln gibt es weitere festgelegte Ebenen.

6.4.1 Primärstapel

Für den Primärstapel gilt folgende Reihenfolge:

Lokale Testfall-Variablen

Lokale Testfall-Variablen liegen im oberen Teil des Primärstapels. Während der Testausführung werden für jeden Knoten, der betreten wird, die definierten Variablen (aber nicht

die Rückfallwerte!) oben auf den Stapel gelegt und beim Verlassen desselben wieder entfernt.

Für jeden Knoten, der eine Variablendefinitionstabelle besitzt, wird auf dem Primärstapel eine eigene Ebene angelegt, in der die in der Tabelle definierten Variablen liegen. Variable aus lokalen (nicht globalen!) Variablenzuweisungen werden in einer der vorhandenen Ebenen hinzugefügt beziehungsweise aktualisiert, zum Beispiel der Rückgabewert einer Prozedur, die Ergebnisvariable eines Check- beziehungsweise Auslesen-Knotens oder Variablen, die über Variable setzen oder Skript-Knoten erstellt werden. Die Variable wird der Ebene des obersten Prozedurknotens, oder wenn nicht vorhanden, des Testfallknotens hinzugefügt beziehungsweise aktualisiert - falls die Variable nicht in einer darüber liegenden Ebene (zum Beispiel Sequenz-, Testschritt-, Schleife-, If-Knoten) bereits existiert. In diesem Fall wird der Wert der Variable in dieser höheren Ebene aktualisiert.

Um lokale Variablen anzulegen, muss bei den entsprechenden Knoten das Attribut Lokale Variable aktiviert sein. Dies kann in den Optionen voreingestellt werden (siehe Variablen(592)). Aus Skripten heraus werden lokale Variablen mit den Methoden `rc.setLocal` oder `rc.setLocalJson` angelegt (siehe Die API des Runcontexts(1030)).

Globale Variablen

Wenn in Knoten, die Variablen definieren können, das Attribut Lokale Variable nicht aktiviert ist, wird die Variable auf der Ebene der globalen Variablen angelegt. Aus Skripten heraus werden Variablen auf dieser Ebene mit den Methoden `rc.setGlobal` oder `rc.setGlobalJson` (siehe Die API des Runcontexts(1030)), angelegt.

Hinweis

Eine globale Variable kann auch trotz Lokale Variable erzeugt werden, wenn kein Kontext für lokale Variablen verfügbar ist. Dies ist zum Beispiel der Fall, wenn ein Knoten direkt aus den 'Extrasequenzen' ausgeführt wird.

Eine globale Variable bleibt unverändert bestehen, bis sie explizit aktualisiert, gelöscht oder QF-Test beendet wird. Die globalen Variablen "überleben" also einzelne Testläufe. Sie dienen dazu, Werte zwischen voneinander unabhängigen Testfällen oder Prozeduren auszutauschen. Sie sollten aber im Hinterkopf behalten, dass diese Variablen zunächst durch den Ablauf des Tests definiert werden müssen, bevor sie referenziert werden können.

Falls Sie globale Variablen bearbeiten wollen, ist dies entweder im Debug-Modus möglich (siehe Anzeige der Variablen im Debug-Modus – Beispiel(123)) oder im Optionendialog in der Rubrik "Variablen".

Um vor einem Testlauf die vorhandenen globalen Variablen zu löschen, nutzen Sie den Menüeintrag Wiedergabe→Globale Variablen löschen. Läuft QF-Test im Batchmodus

(vgl. Aufruf von QF-Test[13]), werden die globalen Variablen vor der Ausführung jedes mit dem Kommandozeilenargument `-test <Index>|<ID>`[992] angegebenen Tests gelöscht.

Kommandozeilen-Variablen

Beim Start von QF-Test können über Kommandozeilenargumente Variablen definiert werden. Diese rangieren über den Variablen, die im Testsuite[595]-Knoten spezifiziert sind. In der Kommandozeile werden die Variablen über das Argument `-variable <Name>=<Wert>`[994] gesetzt, siehe auch Kommandozeilenargumente und Rückgabewerte[971].

Variablen des Testsuite-Knotens

Auf dieser Ebene des Stapels liegen die Variablen, die im Testsuite-Knoten der aktuellen Testsuite definiert sind. Hier werden typischerweise Variablen angelegt, die für alle Tests der Testsuite gelten und bei Bedarf in einem Batch-Lauf per Kommandozeile überschrieben werden können sollen. Ein typisches Beispiel hierfür ist die Wahl des Browsers, in dem eine Webapplikation ausgeführt werden soll. Diese kann sich dann zwischen interaktiver Testentwicklung und Batch-Ausführung unterscheiden.

6.4.2 Sekundärstapel

Für den Sekundärstapel gilt folgende Reihenfolge:

Rückfallwerte

Beim Betreten eines Knotens, für den Rückfallwerte definiert wurden, werden diese oben auf den Sekundärstapel gelegt. Wenn ein Knoten aus einer anderen Testsuite aufgerufen wird, werden auch die Variablen des Testsuite-Knotens der verlassenen Testsuite vom Primärstapel genommen und oben auf den Sekundärstapel gelegt. Beim Verlassen des Knotens beziehungsweise der Testsuite werden die Variablen wieder vom Sekundärstapel entfernt und im Falle der Testsuite auf den Primärstapel in die entsprechende Ebene verschoben.

Auf dem Sekundärstapel werden nur dann Einträge angelegt, wenn für die entsprechenden Knoten Rückfallwerte definiert wurden oder rufende Testsuiten Variablendefinitionen im Testsuite-Knoten enthalten.

Systemspezifische Variablen

Hier können Pfadnamen, JDK- oder Betriebssystem-spezifische Werte etc. festgelegt werden. Dieser Satz von Definitionen befindet sich immer ganz unten im sekundären Stapel und hat damit die geringste Bindungskraft. Sie können im Optionendialog in der Rubrik ”Variablen” gesetzt werden. Die Variablen werden zusammen mit anderen Systemoptionen in der System-Konfigurationsdatei gespeichert.

6.5 Anzeige der Variablen im Debug-Modus – Beispiel

Betrachten wir folgendes Beispiel:

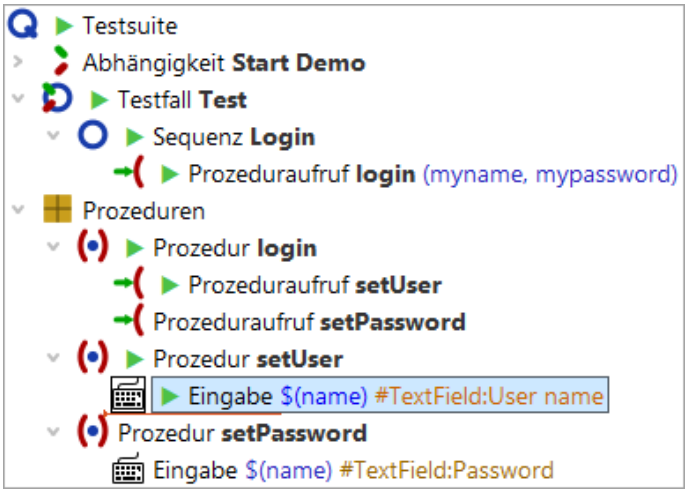


Abbildung 6.3: Variablen Beispiel

Die Sequenz ”Login” enthält einen Prozeduraufruf der Prozedur ”login”, die zwei Parameter erwartet: `user` und `password`. Die Standardwerte für Parameter der Prozedur sind `user=username` und `password=pwd`. Der Prozeduraufruf überschreibt diese mit `user=myname` und `password=mypassword`.

Die ”login” Prozedur enthält selbst Prozeduraufrufe von weiteren Prozeduren. Hier werden keine Parameter übergeben. Die Prozeduren ”setUser” und ”setPassword” haben in Standardwerte für Parameter jeweils einen Eintrag.

Die folgende Abbildung zeigt die Übersicht der Variablendefinitionen zum Zeitpunkt der Ausführung der Prozedur ”setUser”.

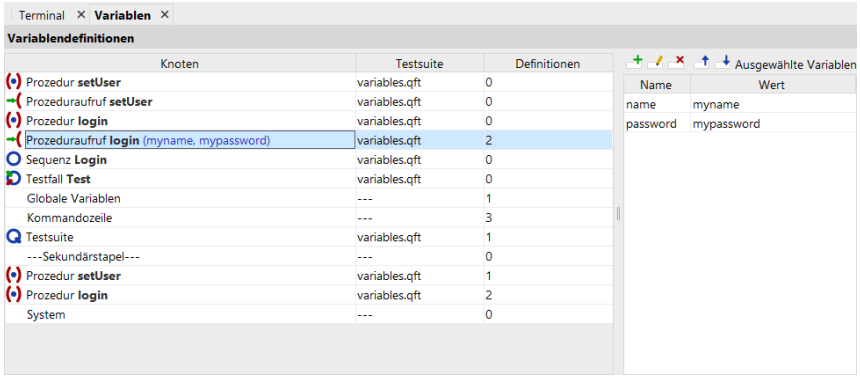


Abbildung 6.4: Variablendefinitionen

Lassen Sie uns einen genaueren Blick auf die einzelnen Zeilen der Tabelle werfen:

1. Prozedur setUser: Keine Variable definiert.
2. Prozeduraufruf setUser: Hier werden keine Variablen übergeben. Dies ist nicht notwendig (wie zum Beispiel bei Java). Bei der Auswertung der Variablendefinitionen sucht QF-Test die Tabelle Zeile für Zeile von oben nach unten durch - unabhängig von Prozedur- oder Testfallgrenzen. Sobald eine Variable mit dem passenden Namen gefunden wird, wird der zugehörige Wert verwendet.
3. Prozedur login: Keine Variable definiert.
4. Prozeduraufruf login: Hier werden zwei Variablen im Prozeduraufruf definiert. Die Zeile wurde selektiert. Somit sind auf der rechten Seite die definierten Variablen mit ihren Werten zu sehen. Beim aktuellen Ausführungsstand des Tests wird als nächstes die Variable ”name” verwendet. Da in dieser Zeile das erste Vorkommen einer Variablen mit diesem Namen ist, wird der zugehörige Wert, ”myName”, verwendet.
5. Sequenz Login: Keine Variable definiert.
6. Testfall Test: Keine Variable definiert.
7. Globale Variablen: Im Abhängigkeit-Knoten wurde die Variable ”client” definiert, da sie in allen Testfällen, die mit der zu testenden Applikation interagieren, benötigt wird. Globale Variablen bleiben unverändert bestehen, bis sie explizit geändert oder gelöscht werden.
8. Kommandozeile: Es wurden drei Variablen in der Kommandozeile definiert. Eine davon ist der Name des Browsers, der für den aktuellen Testlauf verwendet werden soll.

9. Testsuite: Hier ist der Name des Browsers hinterlegt, der verwendet wird, wenn kein anderer Browser in einer der Zeilen darüber definiert ist.

10. Sekundärstapel: Kennzeichnung, dass hier der Primärstapel endet und darunter der Sekundärstapel beginnt.

11. Prozedur setUser: Es ist ein Standardwert für die Variable ”name” hinterlegt, der verwendet würde, wenn es in keiner der Zeilen darüber eine Variable mit diesem Namen gäbe.

12. Prozedur login: Auch hier sind Standardwerte für die Variablen ”name” und ”password” hinterlegt, die verwendet würden, wenn es in keiner der Zeilen darüber eine Variable mit dem entsprechenden Namen gäbe.

13. System: Keine Variable definiert.

6.6 Datentypen von Variablen

Die Attributfelder der QF-Test Knoten interpretieren bis auf wenige Ausnahmen die eingegebenen Werte als Text. Bei den Ausnahmen handelt es sich um die Bedingungen von If(693)-, Testfall(599)- und Testfallsatz(606)-Knoten sowie Code-Attribute, die gültige Ausdrücke einer bestimmten Syntax erwarten.

Da die Attribute im Normalfall als Text interpretiert werden, ist eine spezielle Syntax für den Zugriff auf Variablen oder für die Durchführung von Berechnungen oder Textmanipulationen notwendig. Siehe Variablenreferenzen(116) beziehungsweise Skriptausdrücke(189).

In Skript-Knoten können alle Datentypen genutzt werden, die die jeweilige Skriptsprache zur Verfügung stellt. Innerhalb der Skriptinterpreter können die Datenobjekte von beliebigen Skripten genutzt werden (siehe Variablen(191)). Sie tauchen jedoch nicht auf dem Variablenstapel von QF-Test auf und werden daher weder im Debug-Modus in der Tabelle der Variablendefinitionen angezeigt noch im Protokoll dokumentiert.

Um Variablen aus Skripten auf den Variablenstapel von QF-Test zu legen, stehen die Runcontext-Methoden `rc.setLocal` und `rc.setGlobal` zur Verfügung. Hiermit können QF-Test Variablen Textwerte, aber auch Werte mit anderen Datentypen zugewiesen werden. Um in Variable setzen(871) Knoten Werte zu setzen, die keine Texte darstellen, können im Attribut Defaultwert Skriptausdrücke(189) genutzt werden; oder Sie können die Text-Darstellung des Wertes darin eintragen und über das Attribut Expliziter Objekttyp(874) den gewünschten Objekttyp angeben.

Für den Zugriff auf diese Variablen stehen je nach Bedarf unterschiedliche Methoden zur Verfügung. Für QF-Test Knoten sind sie in Variablenreferenzen(116) beschrieben, für

Skripte und Skriptausdrücke in Variablen[191], speziell für Jython-Skripte in Jython-Variablen[200].

Eine detaillierte Beschreibung der Methoden finden Sie in Die API des Runcontexts[1030].

6.6.1 JSON-Daten

Bei der Arbeit mit HTTP-Requests werden die Daten oft als JSON-Objekt bereitgestellt. Wenn Sie das Objekt serialisieren, also in eine JSON-Zeichenkette umwandeln und einer QF-Test Variablen zuweisen wollen, können Sie die Methoden `rc.setLocalJson()` und `rc.setGlobalJson()` des Runcontexts (siehe Die API des Runcontexts[1030]) in einem Skript-Knoten verwenden.

Wenn Sie eine JSON-Zeichenkette in ein JSON-Objekt umwandeln wollen, steht Ihnen in einem Skript-Knoten `rc.getJson()` im Runcontext (siehe Die API des Runcontexts[1030]) zur Verfügung.

JSON-Objekte können mit den in Das `JSON` Modul[1065] beschriebenen Methoden bearbeitet werden.

6.7 Externe Daten

Auf externe Daten kann mit Hilfe von Properties laden[893], Excel-Datei[659], Datenbank[653], CSV-Datei[664] und Ressourcen laden[890] Knoten zugegriffen werden. Diese weisen einem Satz von Definitionen einen Gruppennamen zu. Den Wert einer Ressource oder Property mit der Bezeichnung *Name* erhalten Sie mit der Syntax `${`*`Gruppe`*`:`*`Name`*`}`.

In einem Datentreiber[646] kann ebenfalls über Excel-Datei[659], Datenbank[653] und CSV-Datei[664] auf externe Daten zugegriffen werden. In diesem Fall wird jedoch keine Gruppe erstellt, sondern je Datenzeile eine Schleifeniteration generiert, in der die Werte des Datensatzes an einfache Variablen gebunden, deren Name der Titel der entsprechenden Datenspalte ist und auf die über die Syntax `$(`*`Spaltentitel`*`)` zugegriffen werden kann.

Wird ein Test im Batchmodus ausgeführt (vgl. Aufruf von QF-Test[13]), löscht QF-Test die Ressourcen und Properties vor der Ausführung jedes mit dem Kommandozeilenargument `-test <Index>|<ID>`[992] angegebenen Tests. Im interaktiven Modus werden diese aufgehoben, um das Erstellen einer Testsuite zu vereinfachen. Vor einem kompletten Testlauf sollten Sie allerdings mittels Wiedergabe→Ressourcen und Properties löschen für eine saubere Ausgangsbasis sorgen.

6.8 Spezielle Gruppen

Folgende Variablengruppen sind immer vorhanden. Auf die Werte kann über die Syntax `$(Gruppenname:Variablenname)` zugeriffen werden.

system

Über die Gruppe `system` haben Sie Zugriff auf die System `Properties` der laufenden Java-VM (für Programmierer: `System.getProperties()`). Es handelt sich dabei immer um die VM, mit der QF-Test gestartet wurde, da die Variablen-Expansion dort stattfindet.
So liefert etwa `${system:java.class.path}` den Klassenpfad, mit dem QF-Test gestartet wurde oder `${system:user.home}` das Heimatverzeichnis des Benutzers. Welche Namen in der `system` Gruppe definiert sind, hängt vom verwendeten JDK ab.

env

Falls das Betriebssystem Umgebungsvariablen wie `PATH`, `TMP` oder `JAVA_HOME` unterstützt (was auf praktisch allen Systemen der Fall ist, auf denen QF-Test läuft), kann über die Gruppe `env` auf diese Variablen zugegriffen werden.

decrypt

9.0+ Über die Gruppe `decrypt` können Sie eine Zeichenkette für die weitere Verarbeitung in QF-Test (zum Beispiel für eine Eingabe in Textfelder, ein Token für einen API-Zugriff oder ein Datenbankpasswort) temporär entschlüsseln. QF-Test ersetzt dabei im Protokoll die Expansion eines entschlüsselten Wertes durch den Platzhalter `***`. Zum Verschlüsseln eines Wertes markieren Sie diesen im Variable setzen(871) Knoten und wählen Sie nach einem Rechts-Klick Text verschlüsseln aus dem resultierenden Popupmenü.

Hinweis Hinweis: Bei einigen Werten in QF-Test Knoten wird grundsätzlich der endgültige Expansionswert im Protokoll ausgegeben. Begutachten Sie daher im Zweifelsfall die Werte im Protokoll, bevor Sie dieses weitergeben. Beachten Sie darüber hinaus die Hinweise zur Option Salt für Verschlüsselung von Kennwörtern(532).

default

3.4+ Sie können über die Gruppe `default` einen Defaultwert für eine Variable angeben. Die Syntax hierfür ist `${default:varname:defaultvalue}`, in Skripten `rc.withDefault('defaultvalue').getStr('varname')` oder auch `rc.getStr('default', 'varname:defaultvalue')`. Dies ist sehr nützlich für Dinge wie generische Komponenten und nahezu überall, wo es einen sinnvollen Defaultwert für eine Variable gibt, da der Defaultwert dann eng mit der Anwendung der Variablen verbunden ist und nicht auf Sequenz oder Testsuite-Ebene definiert werden muss. Natürlich sollten Sie diese Syntax nur verwenden, wenn die Variable nur an einer oder sehr wenigen Stellen benutzt

wird. Wenn Sie dieselbe Variable mit dem selben Defaultwert an verschiedenen Stellen verwenden, ist es besser, die normale `$(...)` Syntax zu verwenden und den Defaultwert explizit festzulegen, da dieser dann bei Bedarf an einer einzigen Stelle geändert werden kann.

as

9.0+ Über die Gruppe `as` lässt sich - ähnlich wie in einem Variable setzen(871) oder Return(678) Schritt - der Typ eines Objektes ändern. Die Syntax hierfür ist `${as:type:value}`, wobei in `value` über `$(...)` auch Werte in Variablen referenziert werden können. Gültige Werte für `type` sind: `string`, `str`, `boolean`, `number`, `object`, `pattern`, `integer`, `int`, `long`, `float`, `double`, `cmdline` und `json`.

id

3.1+ Die Gruppe `id` dient dazu, QF-Test IDs von Komponenten zu referenzieren. Die Werte in dieser Gruppe expandieren einfach zu sich selbst, d.h. `"${id:wasauchimmer}"` wird zu `"wasauchimmer"`code>. Man kann QF-Test IDs von Komponenten zwar auch ohne diese Gruppe ansprechen, allerdings verbessert die Referenzierung über diese Gruppe die Lesbarkeit der Tests. Vor allem aber werden diese QF-Test IDs auch beim Verschieben der referenzierten Komponente oder Änderungen an ihrer QF-Test ID angepasst.

idlocal

4.2.3+ Die Gruppe `idlocal` ist analog zur Gruppe `id`, enthält aber zusätzlich den Pfad der aktuellen Testsuite, d.h. aus `"${idlocal:x}"` wird `"pfad/zur/aktuellen/suite/suite.qft#x"`. Damit lässt sich erzwingen, dass eine Komponente nur aus der Testsuite genommen wird, die zum Zeitpunkt der Expansion aktuell ist, selbst wenn es eine Komponente mit derselben QF-Test ID in der Zieltestsuite für den Prozeduraufruf gibt.

quoteitem

4.0+ Mit Hilfe der Gruppe `quoteitem` können Sie bequem Sonderzeichen wie '@', '&' und '%' im textuellen Index eines Unterelements schützen, um zu verhindern, dass dieses als mehrere Unterelemente interpretiert wird. Aus `"${quoteitem:user@host.org}"` wird zum Beispiel `"user\@host.org"`.

quoteregex, quoteregexp

4.0+ Die Gruppe `quoteregex` mit ihrem Alias `quoteregexp` kann zum Schützen von Sonderzeichen mit spezieller Bedeutung in Reguläre Ausdrücke - *Regexps*(1023) verwendet werden. Dies ist hilfreich, wenn reguläre Ausdrücke dynamisch zusammengebaut werden.

quotesmartid

6.0.1+ Die Gruppe `quotesmartid` schützt analog zu `quoteitem` die Zeichen für Unterelemente '@', '&' und '%', außerdem die Zeichen ':', '=', '<' und '>' mit

spezieller Bedeutung für SmartIDs. Aus "`${quotesmartid:Name: A & B}`" wird zum Beispiel "`Name\: A \& B`".

qftest

Die Gruppe namens `qftest` stellt verschiedene Werte zur Verfügung, die beim Ablauf eines Tests von Bedeutung sein können. Die bisher definierten Werte können Sie den folgenden Tabellen entnehmen.

Name	Bedeutung
`32` oder `32bit`	Nicht mehr relevant, da die Unterstützung von 32 Bit Java für QF-Test mit Version 8.0 eingestellt wurde. *true* wenn QF-Test in einer 32-Bit-Java-VM läuft - was nicht bedeutet, dass dies auf einem 32 Bit Betriebssystem erfolgt - andernfalls *false*.
`64` oder `64bit`	Nicht mehr relevant, da die Unterstützung von 32 Bit Java für QF-Test mit Version 8.0 eingestellt wurde. *true* wenn QF-Test in einer 64 Bit Java-VM läuft, andernfalls *false*.
`batch`	*true* falls QF-Test im Batch Modus läuft, *false* im interaktiven Modus.
`client.baseEngineName.<name>`	Der Basisname der primären Engine des Clients, der mit *<name>* als Client(729) Attribut gestartet wurde, z.B. `fx`.
`client.browser.<name>`	Der Name bzw. Typ des Browsers des Clients, der mit *<name>* als Client(738) Attribut gestartet wurde, z.B. `safari`. Steht nur für Web-Clients zur Verfügung.
`client.deviceName.<name>`	Ein Name für das (emulierte) Gerät des Clients, der mit *<name>* als Client(750) Attribut gestartet wurde, z.B. `Pixel_3`. Steht nur zur Verfügung für Android-Clients nach der Instrumentierung und entspricht dort bei emulierten Geräten dem AVD-Namen.
`client.deviceType.<name>`	Der Typ des (emulierten) Geräts des Clients, der mit *<name>* als Client(750) Attribut gestartet wurde. Kann die Werte `emulator` (für ein emuliertes Gerät) und `device` (für ein real angeschlossenes Gerät) annehmen. Steht nur zur Verfügung für Android-Clients nach der Instrumentierung.
`client.connectionMode.<name>`	Der Name des Verbindungsmodus des Clients, der mit *<name>* als Client(738) Attribut gestartet wurde. Gültige Werte sind `qfdriver`, `cdpdriver`, `webdriver` und `embedded`. Steht nur für Web-Clients zur Verfügung.
`client.engine.<name>`	Die primäre Engine des Clients, der mit *<name>* als Client(729) Attribut gestartet wurde. Das Ergebnis besteht dabei aus dem Basisnamen der Engine und einem numerischen Index, z.B. `fx0`.
`client.engineNames.<name>`	Eine Liste aller verbundenen Engines des Clients, der mit *<name>* als Client(729) Attribut gestartet wurde, z.B. `[fx0, web_fx0]`.

client.exitCode.<name>	Der Rückgabewert des letzten Prozesses, der mit *<name>* als Client(729) Attribut gestartet wurde. Ist der Prozess noch aktiv, ist das Ergebnis leer.
client.mainVersion.<name>	Der Hauptversion des Browsers bzw. des Geräte-Betriebssystems des Clients, der mit *<name>* als Client(738) Attribut gestartet wurde, z.B. 121. Steht nur zur Verfügung für Web-Clients, nachdem das erste Browserfenster geöffnet wurde, und für Android-Clients nach der Instrumentierung.
client.output.<name>	Die Ausgaben des letzten Prozesses, der mit *<name>* als Client(729) Attribut gestartet wurde. Das Maximum an gespeichertem Text wird durch die Option Maximalgröße des Terminals für einen Client (kB)(536) bestimmt.
client.SDKVersion.<name>	Der SDK-Version des Geräte-Betriebssystems des Clients, der mit *<name>* als Client(750) Attribut gestartet wurde, z.B. 29. Steht nur zur Verfügung für Android-Clients nach der Instrumentierung.
client.stdOut.<name>	Die vom letzten Prozesses, der mit *<name>* als Client(729) Attribut gestartet wurde, auf den Standardausgabestream (stdout) geschriebenen Ausgaben. Das Maximum an gespeichertem Text wird durch die Option Maximalgröße des Terminals für einen Client (kB)(536) bestimmt.
client.stdErr.<name>	Die vom letzten Prozesses, der mit *<name>* als Client(729) Attribut gestartet wurde, auf den Fehlerausgabestream (stderr) geschriebenen Ausgaben. Das Maximum an gespeichertem Text wird durch die Option Maximalgröße des Terminals für einen Client (kB)(536) bestimmt.
client.version.<name>	Der Browserversion bzw. die Version des Geräte-Betriebssystems des Clients, der mit *<name>* als Client(738) Attribut gestartet wurde, z.B. 121.10.2967.10. Steht nur zur Verfügung für Web-Clients, nachdem das erste Browserfenster geöffnet wurde, und für Android-Clients nach der Instrumentierung.
clients	Eine Liste der Namen der aktiven Client-Prozesse, mit Zeilentrennern getrennt.
clients.all	Eine Liste der Namen aller Client-Prozesse, mit Zeilentrennern getrennt. Die Liste enthält aktive Prozesse ebenso wie kürzlich beendete, analog zum ”Clients” Menü.
count.exceptions	Anzahl der Exceptions im aktuellen Testlauf.
count.errors	Anzahl der Fehler im aktuellen Testlauf.
count.warnings	Anzahl der Warnungen im aktuellen Testlauf.
count.testCases	Gesamtanzahl der Testfälle (ausgeführt und übersprungen) im aktuellen Testlauf.
count.testCases.exception	Anzahl der Testfälle mit Exceptions im aktuellen Testlauf.
count.testCases.error	Anzahl der Testfälle mit Fehlern im aktuellen Testlauf.

count.testCases.expectedToFail	Anzahl der erwartet fehlgeschlagenen Testfälle im aktuellen Testlauf.
count.testCases.ok	Anzahl der erfolgreichen Testfälle im aktuellen Testlauf.
count.testCases.ok.percentage	Prozentsatz der erfolgreichen Testfälle im aktuellen Testlauf.
count.testCases.skipped	Anzahl der übersprungenen Testfälle im aktuellen Testlauf.
count.testcases.notImplemented	Anzahl der nicht implementierten Testfälle im aktuellen Testlauf.
count.testCases.run	Anzahl der ausgeführten Testfälle im aktuellen Testlauf.
count.testSets.skipped	Anzahl der übersprungenen Testfallsätze im aktuellen Testlauf.
dir.cache	Das Cache-Verzeichnis von QF-Test
dir.groovy	Verzeichnis von Groovy
dir.javascript	Verzeichnis von JavaScript
dir.jython	Verzeichnis von Jython
dir.log	Das Logverzeichnis von QF-Test
dir.plugin	Das Pluginverzeichnis von QF-Test
dir.root	Wurzelverzeichnis von QF-Test
dir.runlog	Protokollverzeichnis von QF-Test
dir.system	Das systemspezifische Konfigurationsverzeichnis von QF-Test.
dir.user	Das anwenderspezifische Konfigurationsverzeichnis von QF-Test
dir.version	Versionsspezifisches Verzeichnis von QF-Test
engine.<componentId>	Ermittelt die GUI-Engine, die für die angegebene Komponente zuständig ist (vgl. GUI-Engines(998)).
language	Die Sprache in welcher QF-Test seine graphische Oberfläche darstellt.
license	Der Pfad der Lizenzdatei
systemCfg	Der Pfad der Systemkonfigurationsdatei
userCfg	Der Pfad der benutzerspezifischen Konfigurationsdatei
executable	Die ausführbare qftest Programmdatei passend zur aktuell laufenden QF-Test Version, inklusive vollem Pfad zu deren bin Verzeichnis und mit .exe Anhang unter Windows. Dies ist hilfreich, falls Sie QF-Test aus QF-Test starten wollen, z.B. um einen Daemon-Aufruf auszuführen oder Reports zu generieren.
isInRerun	"true", wenn aktuelle Ausführung nochmals ausgeführt wird, sonst "false", Details siehe Fehlerhafte Knoten sofort wiederholen(355).
isInRerunFromLog	"true", wenn Testlauf aus dem Protokoll nochmals gestartet wurde, sonst "false", Details siehe Erneute Ausführung aus dem Protokoll(352).

java	Standard Java-Programm (javaw unter Windows, java unter Linux) oder das explizit mittels -java <Programm> (abgekündigt)(977) angegebene Java-Programm.
java.mainVersion	Die Hauptversion des JRE mit dem QF-Test aktuell läuft, wobei 8 für Java 1.8 genommen wird, also z.B. 8, 11 oder 17.
java.subVersion	Die Unterversion des JRE mit dem QF-Test aktuell läuft. Für Java 8 wird die Unterversion nach dem '_ genommen, was z.B. für java.version 1.8.0_302 zu 302 führt. Für Java 9 oder höher ist dies die normale Unterversion, also z.B. 9 in Fall von java.version 11.0.9.
linux	"true" unter Linux, andernfalls "false"
macOS	"true" unter macOS, andernfalls "false"
os.fullVersion	Die vollständige Version des Betriebssystems
os.mainVersion	Die Hauptversion des Betriebssystems, z.B. "10" für Windows 10
os.name	Der Name des Betriebssystems
os.version	Die konkrete Version des Betriebssystems. Unter Windows kann diese ggf. nicht vollständig sein. In diesem Fall sollten Sie auf os.fullversion zurückgreifen.
project.dir	Das Verzeichnis des aktuellen Projektes. Die Variable ist nicht definiert, wenn die aktuelle Test suite sich nicht in einem Projekt befindet.
rerunCounter	Nummer des aktuellen Versuchs der Neuausführung, sonst immer 0 (siehe Fehlerhafte Knoten sofort wiederholen(355)).
return	Der letzte mittels eines Return(678) Knotens aus einer Prozedur(672) zurückgegebene Wert.
runID	Die Run-ID des aktuellen Testlaufs. Nähere Informationen zur Run-ID finden Sie in Abschnitt 24.1(332).
screen.height	Bildschirmhöhe in Pixel
screen.width	Bildschirmbreite in Pixel

skipNode	Dieser spezielle Wert ist nur für erfahrene Anwender. Er weist QF-Test an, die Ausführung des aktuellen Knotens zu überspringen. Sein primärer Nutzen ist als Wert für eine Variable im Attribut Text(786) eines Texteingabe(784) Knotens, dessen Attribute Zielkomponente zunächst leeren(786) gesetzt ist. Ein leerer Wert würde hier zu einem Löschen des Textfeldes führen, $_{qftest:skipnode} hingegen das Feld unverändert lassen. Weiterhin kann skipnode für besondere Fälle der Ablaufsteuerung eingesetzt werden, indem z.B. eine Variable im Kommentar eines Knotens definiert und ihr selektiv der Wert $_{qftest:skipnode} übergeben wird. Beachten Sie bitte, dass hierfür praktisch immer die Lazy Binding Syntax '$_' verwendet werden sollte, da ansonsten die Expansion im Parameter eines Prozeduraufruf Knotens zum Überspringen des gesamten Aufrufs führen würde.
suite.dir	Verzeichnis der aktuellen Suite
suite.file	Dateiname der aktuellen Suite ohne Verzeichnis
suite.path	Dateiname der aktuellen Suite mit Verzeichnis
suite.name	Der Name der aktuellen Testsuite.
testCase.name	Der Name des aktuellen Testfalls, leer falls im Moment kein Testfall ausgeführt wird.
testCase.id	Die QF-Test ID des aktuellen Testfalls, leer falls im Moment kein Testfall ausgeführt wird.
testCase.qName	Der qualifizierte Name des aktuellen Testfalls, inklusive der Namen seiner Testfallsatz Parentknoten. Leer falls im Moment kein Testfall ausgeführt wird.
testCase.reportName	Der expandierte Report-Name des aktuellen Testfalls, leer falls im Moment kein Testfall ausgeführt wird.
testCase.splitLogName	Der qualifizierte Name des aktuellen Testfalls als Dateiname, inklusive der Namen seiner Testfallsatz Parentknoten als Verzeichnisse. Leer falls im Moment kein Testfall ausgeführt wird.
testSet.name	Der Name des aktuellen Testfallsatzes, leer falls im Moment kein Testfallsatz ausgeführt wird.
testSet.id	Die QF-Test ID des aktuellen Testfallsatzes, leer falls im Moment kein Testfallsatz ausgeführt wird.
testSet.qName	Der qualifizierte Name des aktuellen Testfallsatzes, inklusive der Namen seiner Testfallsatz Parentknoten. Leer falls im Moment kein Testfallsatz ausgeführt wird.
testSet.reportName	Der expandierte Report-Name des aktuellen Testfallsatzes, leer falls im Moment kein Testfallsatz ausgeführt wird.
testSet.splitLogName	Der qualifizierte Name des aktuellen Testfallsatzes als Dateiname, inklusive der Namen seiner Testfallsatz Parentknoten als Verzeichnisse. Leer falls im Moment kein Testfallsatz ausgeführt wird.

`testStep.name`	Der Name des aktuellen Testschritts (inkl. @teststep doctag), leer falls im Moment kein Testschritt ausgeführt wird.
`testStep.qName`	Der qualifizierte Reportname des aktuellen Testschritts (inkl. @teststep doctag), inklusive der Reportnamen seiner Testschritt Parentknoten aber ohne Testfall und Testfallsatz Parentknoten. Leer falls im Moment kein Testschritt ausgeführt wird.
`testStep.reportName`	Der expandierte Report-Name des aktuellen Testschritts (inkl. @teststep doctag), leer falls im Moment kein Testschritt ausgeführt wird.
`thread`	Der Index des aktuellen Threads. Immer 0, es sei denn QF-Test wird mit dem Kommandozeilenargument `-threads <Anzahl>`(994) gestartet.
`threads`	Die Anzahl der parallelen Threads. Immer 1, es sei denn QF-Test wird mit dem Kommandozeilenargument `-threads <Anzahl>`(994) gestartet.
`version`	QF-Test Version
`version.build`	QF-Test Buildnummer
`windows`	”true” unter Windows, andernfalls ”false”

Tabelle 6.1: Definitionen in der Gruppe `qftest`

6.9 Immediate und Lazy Binding

3.0+

Es gibt einen sehr subtilen Aspekt bei der Verwendung von QF-Test Variablen auf den wir noch genauer eingehen müssen:

Wenn ein Satz von Variablendefinitionen auf einen der beiden Stapel gelegt wird gibt es zwei Möglichkeiten zur Behandlung von Referenzen auf Variablen im Wert einer Definition. Hat z.B. die Variable namens ’x’ den Wert ’$(y)’, kann dieser Wert wortwörtlich gespeichert werden, so dass der Wert von ’$(y)’ erst zu einem späteren Zeitpunkt ermittelt wird, wenn irgendwo ’$(x)’ referenziert wird. Alternativ kann der Wert von ’$(y)’ schon beim Binden der Variablen ’x’ ermittelt und als Wert von x abgelegt werden. Der erste Ansatz wird als ”Lazy Binding” oder ”Late Binding” bezeichnet, der zweite als ”Immediate Binding”.

Der Unterschied zwischen beiden ist natürlich der Zeitpunkt der Expansion und damit der Kontext in dem eine Variable expandiert wird. In den allermeisten Fällen ist das Ergebnis das gleiche, aber es gibt Situationen in denen es von großer Bedeutung ist, Lazy oder Immediate Binding zu verwenden. Betrachten wir die folgenden zwei Beispiele:

Eine Testsuite-Bibliothek stellt eine Prozedur zum Start des SUT mit verschiedenen JDK

Versionen bereit. Die Variable 'jdk' wird als Parameter an diese Prozedur übergeben. Zur einfacheren Nutzung definiert der Autor der Bibliothek weitere hilfreiche Variablen auf Testsuite-Ebene, z.B. 'javabin' für das ausführbare Java-Programm mit dem Wert '/opt/java/$(jdk)/bin/java'. Zu dem Zeitpunkt, zu dem 'javabin' in den Testsuite Variablen gebunden wird, könnte 'jdk' noch undefiniert sein, so dass Immediate Binding zu einem Fehler führen würde. Doch selbst wenn 'jdk' mit einem Standardwert belegt ist hat Immediate Binding nicht den gewünschten Effekt, da sich der Wert von 'javabin' durch Übergabe eines anderen Wertes für 'jdk' an die Prozedur nicht mehr ändert. Lazy Binding ist hier also die Methode der Wahl.

Betrachten wir eine andere Bibliothek mit einer Prozedur zum Kopieren einer Datei. Die zwei Parameter namens 'source' und 'dest' legen die Ausgangsdatei und das Zielverzeichnis fest. Der Aufrufer der Prozedur möchte die Datei 'data.csv' aus dem Verzeichnis seiner Testsuite an einen anderen Ort kopieren. Die nahe liegende Idee ist, für den Parameter 'source' den Wert '${qftest:suite.dir}/data.csv' an die Prozedur zu übergeben. Mit Immediate Binding liefert '${qftest:suite.dir}' in der Tat das Verzeichnis der aufrufenden Suite. Wird allerdings Lazy Binding verwendet, findet die Expansion erst innerhalb der Prozedur statt, so dass '${qftest:suite.dir}' das Verzeichnis der Bibliotheks-Suite liefert, was im Allgemeinen nicht den gewünschten Effekt hat.

In QF-Test Versionen bis einschließlich 2.2 wurde ausschließlich Lazy Binding unterstützt. Wie die obigen Beispiele zeigen, sind aber beide Varianten sinnvoll und notwendig. Immediate Binding ist aber intuitiver und leichter nachzuvollziehen und daher inzwischen die Standard-Methode (dies kann aber mittels der Option Werte von Variablen beim Binden sofort expandieren[592] geändert werden). Die Option Lazy Binding verwenden falls sofortiges Expandieren scheitert[593] ergänzt diese und ermöglicht eine einfache Migration von älteren Testsuiten zum Gebrauch von Immediate Binding. Die Warnungen, die in diesem Zusammenhang ausgegeben werden, helfen Ihnen, die wenigen Stellen zu lokalisieren, an denen Sie wie unten beschrieben explizit Lazy Binding verwenden sollten. Bis auf die äußerst seltenen Fälle, in denen Lazy Binding benötigt wird, Immediate Binding aber auch funktioniert, so dass kein Rückgriff auf Lazy Binding gemacht wird, sollten alle Tests ohne Änderungen lauffähig sein.

In den wenigen Fällen, in denen es einen Unterschied macht, ob eine Variable mittels Immediate oder Lazy Binding definiert wird, kann dies unabhängig von der Voreinstellung durch eine alternative Variablen-Syntax erzwungen werden. Für Immediate Binding verwenden Sie '$!' anstatt nur '$', Lazy Binding erhalten Sie mittels '$_'. Um z.B. auf Testsuite-Ebene eine Variable zu definieren, deren Wert eine Datei im Verzeichnis dieser Suite ist, verwenden Sie '$!{qftest:suite.dir}/somefile'. Wenn Sie wie in obigem 'jdk' Beispiel Lazy Binding benötigen, verwenden Sie '$_(jdk)'.

Hinweis

Mit Lazy Binding war es egal, in welcher Reihenfolge Variablen oder Parameter in einem Knoten oder Datentreiber definiert waren, da während des Bindens keine Expansion stattfand. Bei Immediate Binding werden Variablen von oben nach unten, bzw.

bei Datentreibern von links nach rechts expandiert. Das bedeutet, dass die Definition von x=1 und y=$(x) funktioniert und y den Wert 1 erhält, wenn x zuerst definiert wird. Kommt hingegen die Definition von y zuerst, führt dies zu einem Fehler oder dem oben beschriebenen Rückfall auf Lazy Binding.

Kapitel 7

Problemanalyse und Debugging

Da der eigentliche Zweck der Testautomatisierung darin besteht, Probleme im SUT aufzudecken, ist davon auszugehen dass Tests hin und wieder fehlschlagen.

Nachdem ein Testlauf beendet ist, erscheint in der Statuszeile des Hauptfensters von QF-Test eine Meldung mit dem Ergebnis. Im Idealfall lautet diese ”Keine Fehler”. Sind Probleme aufgetreten, wird die Zahl der Warnungen, Fehler und Exceptions angezeigt und gegebenenfalls zusätzlich ein Dialogfenster geöffnet. In diesem Fall ist es Ihre Aufgabe herauszufinden, was schiefgelaufen ist.

Manchmal ist die Ursache eines Problems offensichtlich, in den meisten Fällen jedoch nicht. Am wichtigsten ist es in so einem Fall zu klären, ob der Test auf Grund eines Fehlers im SUT fehlgeschlagen ist, oder ob sich das SUT korrekt verhalten hat aber die Testlogik einen Fehler aufweist. Das Dilemma besteht darin, dass ein mögliches Problem im SUT nicht übersehen werden darf und so früh wie möglich gemeldet werden sollte. Andererseits verschwendet ein unberechtigter Fehlerreport Zeit und zieht eventuell den Unmut der Entwicklungsabteilung nach sich. Daher muss jedes Problem genau analysiert werden und jeder vermutete Bug im SUT sollte idealerweise reproduzierbar sein bevor er gemeldet wird.

Bei dieser wichtigen Aufgabe bietet QF-Test in zweierlei Hinsicht Unterstützung. Für jeden Testlauf wird ein detailliertes Protokoll erstellt, welches alle relevanten Informationen für eine Post-mortem Analyse enthält, inklusive Bildschirmfotos von dem Zeitpunkt, an dem ein Fehler aufgetreten ist. Der integrierte Test Debugger unterstützt Sie dagegen bei der Analyse des Kontroll- und Informationsflusses zur Laufzeit eines Tests.

Video

Das Video

’Fehleranalyse’
https://www.qftest.com/de/yt/fehleranalyse-40.html

zeigt ein kurzes Beispiel zur Vorgehensweise bei der Fehleranalyse.

7.1 Das Protokoll

Beim Abspielen eines Tests erstellt QF-Test ein Protokoll, in dem jede einzelne Aktion notiert wird. Die Protokolle der zuletzt ausgeführten Tests sind über das Wiedergabe Menü zugänglich. Das aktuelle Protokoll kann auch mittels Strg-L oder dem entsprechenden Button in der Toolbar geöffnet werden. Abschnitt 7.1.7(144) gibt eine Übersicht über Optionen, die die Erstellung von Protokollen beeinflussen.

Die Struktur dieses Protokolls ist der einer Testsuite sehr ähnlich, mit einem Unterschied: Knoten werden bei ihrer Ausführung in das Protokoll aufgenommen. Wird ein Knoten mehrfach ausgeführt, was z.B. bei Vorbereitung(638) und Aufräumen(641) Knoten häufig der Fall ist, taucht er auch mehrfach im Protokoll auf. Folgende Abbildung zeigt eine typische Situation:

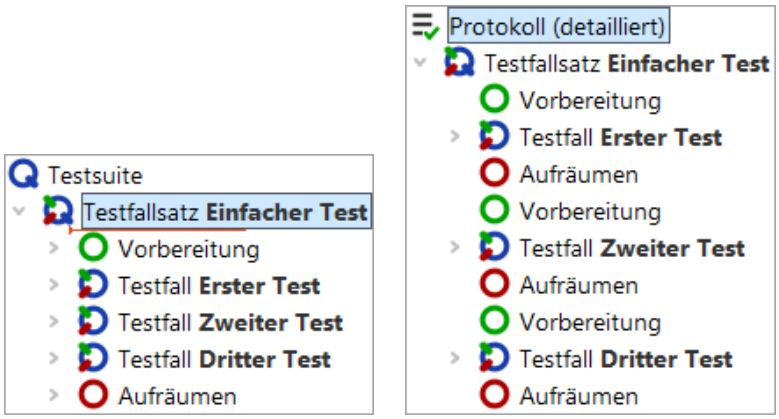


Abbildung 7.1: Ein einfacher Test und sein Protokoll

Das Protokoll ist das entscheidende Hilfsmittel, wenn es herauszufinden gilt, **was** bei einem Testlauf fehlgeschlagen ist, **wo** es passiert ist und - im besten Fall - auch **warum** es passiert ist. Daher liegt das Gewicht bei einem Protokoll bei der Vollständigkeit der Information. Darunter leidet natürlich die Lesbarkeit und die Übersicht. Beides ist Aufgabe von Reports, deren Erstellung in Kapitel 24(330) beschrieben wird.

Neben den Knoten, die aus der Testsuite übernommen wurden, enthält ein Protokoll insbesondere Fehlerinformationen, optionale Anmerkungen, verschiedene Arten von Meldungen sowie Informationen über Variablenexpansion und das Laufzeitverhalten.

Da die gesammelten Informationen über einen längeren Testlauf gewaltige Mengen an Arbeitsspeicher verbrauchen können, verfügt QF-Test über mehrere Möglichkeiten, damit umzugehen. Die beste davon, gleichzeitig die Standardeinstellung, sind geteilte

Protokolle. Diese werden in Abschnitt 7.1.6(143) näher erläutert. Die dabei entstehenden `*.qzp` Dateien im ZIP Format reduzieren nicht nur den Platz auf der Festplatte. Teil der Protokolle können bereits bei der Ausführung ausgelagert und der dafür benötigte Arbeitsspeicher wieder freigegeben werden. Gleiches gilt bei der Verarbeitung von Protokollen, z.B. zur Erstellung von Reports. Die ältere Option Kompakte Protokolle erstellen(590) sowie die alternativen Dateiformate `*.qrz` und `*.qrl` bieten zusätzliche Flexibilität, werden aber primär aus Kompatibilitätsgründen erhalten.

7.1.1 Fehlerzustände

Es gibt drei Arten von Fehlerzuständen, die sich in ihrer Schwere unterscheiden:

Warnungen
: Warnungen weisen auf Probleme hin, die normalerweise nicht ernst sind, aber in Zukunft zu schwereren Problemen führen könnten, so dass es sich lohnen kann, einen Blick darauf zu werfen. So gibt QF-Test z.B. Warnungen aus, wenn eine Komponente bei der Wiedererkennung nur knapp und mit signifikanten Abweichungen gefunden werden konnte.

Fehler
: Ein Fehler ist als ernstzunehmendes Problem anzusehen, dessen Ursache geklärt werden muss. Er weist darauf hin, dass das SUT gewisse Anforderungen nicht erfüllt. Die häufigste Art von Fehlern sind Abweichungen in Check Text(806) Knoten.

Exceptions
: Exceptions sind die schwersten Fehler. Sie werden in Situationen geworfen, in denen QF-Test einen Testlauf nicht sinnvoll fortsetzen kann. Die meisten Exceptions deuten auf einen Fehler in der Testlogik hin. Eine Exception kann aber genauso gut durch einen Fehler im SUT ausgelöst werden. So wird z.B. eine `ComponentNotFoundException`(958) geworfen, wenn im SUT keine passende Komponente für einen Event gefunden wurde. Eine Liste aller möglichen Exceptions finden Sie in Kapitel 43(958).

Jeder Knoten eines Protokolls hat einen von vier Fehlerzuständen: *Normal*, *Warnung*, *Fehler* oder *Exception*. Dieser Zustand wird durch einen Rahmen um das Icon des Knotens dargestellt, dessen Farbe Orange für *Warnung*, rot für *Fehler* und fett rot für *Exception* ist.

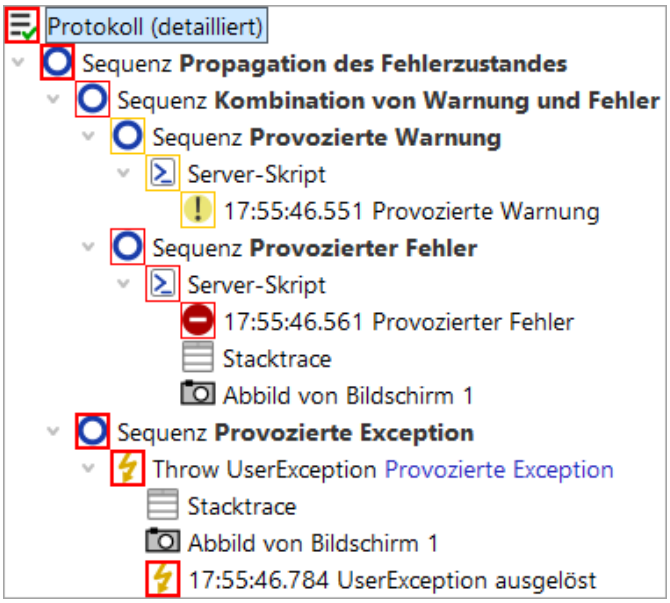


Abbildung 7.2: Fehlerzustände im Protokoll

Der Ausschnitt aus einem Protokoll in obiger Abbildung illustriert, wie Fehlerzustände von unten nach oben propagieren. Der *Exception* Zustand hat mehr Gewicht als der *Fehler* Zustand, der wiederum die *Warnung* überdeckt. Die schwerste Art von Fehler, die bis ganz nach oben im Baum propagiert, bestimmt das Endergebnis des Testlaufs und damit auch den Rückgabewert von QF-Test, wenn es im Batchmodus gestartet wurde (vgl. Abschnitt 44.3(996)).

Wenn nötig kann die Propagation von Fehlern auch (auf Sequenz(618) Ebene) begrenzt werden, z.B. für einen bereits bekannten Fehler, der keinen Einfluss auf das Gesamtresultat haben soll. Diese Einschränkung geschieht für alle Arten von Sequenz Knoten mit Hilfe des Attributs Maximaler Fehler(619). Exceptions können mit Hilfe der Try(704) und Catch(707) Knoten abgefangen werden. Das Attribut Maximaler Fehler(710) des Catch Knotens legt dabei fest, welche Art von Fehlerzustand an Stelle der Exception treten soll.

7.1.2 Navigation im Protokoll

Die grundlegenden Bearbeitungsmöglichkeiten im Protokoll sind analog zur Testsuite, mit dem Unterschied, dass die Attribute der Knoten, die aus der Testsuite übernommen wurden, nicht geändert und dass keine Knoten entfernt oder eingefügt werden können. Knoten können aber mit einer Bemerkung versehen werden, z.B. um den Grund für einen Fehler zu dokumentieren.

Die erste Frage beim Blick auf ein Protokoll ist üblicherweise: **”Was ist passiert?”**

Der Button, bzw. die Funktion Bearbeiten→Nächsten Fehler finden, kurz Strg-N, bewegt die Selektion an die nächste Stelle, an der ein Problem tatsächlich aufgetreten ist.

Analog sucht bzw. Bearbeiten→Vorherigen Fehler finden (Strg-P) rückwärts.

Die Option Unterdrückte Fehler überspringen(581) legt fest ob nach Fehlern gesucht werden soll, die nicht bis nach oben propagiert wurden. Der Menüeintrag Bearbeiten→Unterdrückte Fehler überspringen ist eine Abkürzung zum schnellen Umschalten der letzteren Option.

Die nächste Frage könnte lauten: **"Wo ist das passiert?"**

Obwohl ein Protokoll einer Testsuite in vieler Hinsicht ähnlich ist, ist der Zusammenhang nicht immer offensichtlich, vor allem, wenn Aufrufe tief verschachtelt sind. Die Funktion Bearbeiten→Knoten in Testsuite finden (Strg-T) bringt Sie exakt zu dem Knoten in der Testsuite, der dem selektierten Knoten im Protokoll entspricht. Voraussetzung hierfür ist, dass die Testsuite auffindbar ist und nicht in einer Form geändert wurde, die das verhindert. Wenn das Protokoll aus einer Datei geladen wurde, befindet sich die Testsuite eventuell nicht an der selben Stelle wie bei der Ausführung des Tests. Kann die Suite nicht lokalisiert werden, öffnet sich ein Dialog, in dem Sie selbst eine Datei für die Testsuite auswählen können. Wenn Sie dabei die falsche Testsuite angeben oder wenn automatisch eine falsche Version der Testsuite gefunden wurde, kann es sein, dass Sie bei einem völlig anderen Knoten landen. In diesem Fall können Sie mittels Bearbeiten→Zugehörige Testsuite lokalisieren explizit eine andere Testsuite auswählen.

Diese Zuordnung können Sie über die Protokolloptionen auch voreinstellen (siehe Verweise zwischen Verzeichnissen mit Testsuiten(591)).

7.1.3 Laufzeitverhalten

QF-Test protokolliert für jede Ausführung eines Knotens die Startzeit und zwei Formen der Laufzeit: 'Echtzeit' ist die tatsächlich zwischen Betreten und Verlassen des Knotens vergangene Zeit. Sie beinhaltet explizite Verzögerungen durch das Attribut 'Verzögerung vorher/nachher', Unterbrechungen durch den Benutzer beim Debuggen von Tests oder anderen Overhead, wie die Aufnahme von Bildschirmabbildern. Die tatsächlich für Tests aufgewendete Zeit, die im Attribut 'Dauer' aufsummiert wird, ist daher ein besserer Indikator für die Performance des SUT.

Für ein besseres Verständnis des Laufzeitverhaltens eines Tests kann die Anzeige der relativen Dauer über den Toolbar-Button , das Menü Ansicht→Anzeige für relative Dauer einblenden oder die Option Relative Dauer anzeigen(579) aktiviert werden. Für jeden Knoten werden farbige Balken dargestellt,

deren Länge sich nach dem prozentualen Anteil der Zeit richtet, die für diesen Knoten relativ zur Zeit seines Parent-Knotens aufgewendet wurde. Damit lassen sich Performance-Engpässe leicht auffinden, indem man jeweils die Knoten mit den längsten Balken betritt:

Abbildung 7.3: Anzeige der relativen Dauer im Protokoll

Die Option Anzeigeform für relative Dauer(579), deren Werte auch direkt über das Menü Ansicht→Anzeigeform für relative Dauer zugänglich sind, legt fest, ob sich die Anzeige auf die Dauer, die Echtzeit oder beides bezieht. Letzteres ist besonders effektiv, bedarf aber einer gewissen Eingewöhnung.

7.1.4 Rückgabewerte anzeigen

Ist die Option Rückgabewerte von Prozeduren anzeigen(580) gesetzt (im Protokoll auch direkt über das Ansicht Menü erreichbar), werden Rückgabewerte von Prozedur(672) Knoten im Baum neben dem entsprechenden Prozeduraufruf(675) Knoten angezeigt.

7.1.5 Werte von fehlgeschlagenen Checks als gültig akzeptieren

Ein wichtiges Feature von QF-Test ist die Fähigkeit, sehr einfach den aktuellen Wert eines fehlgeschlagenen Check Knotens als gültigen Wert zu übernehmen. Wenn QF-Test einen gescheiterten Check in das Protokoll schreibt, speichert es dort auch den

kompletten Status der Zielkomponente des Check Knotens im SUT mit. Dies ist sehr viel hilfreicher als eine einfache Fehlermeldung, die zum Beispiel nur mitteilt, dass eine Tabellenspalte 10 statt der erwarteten 9 Einträge enthält, aber nicht was diese Werte sind.

Wenn Sie bei der Analyse eines fehlgeschlagenen Checks feststellen, dass der Wert im SUT korrekt, der in der Testsuite gespeicherte Wert dagegen falsch war, können Sie einfach Strg-U drücken oder den Eintrag Check-Knoten mit erhaltenen Daten aktualisieren im Kontextmenü auswählen, um den Wert aus dem Protokoll in den zugehörigen Check Knoten in der Testsuite zu übernehmen.

Warnung: QF-Test berücksichtigt hierbei im Moment keine regulären Ausdrücke in Check Text(806) oder Check Elemente(818) Knoten, diese werden einfach überschrieben.

7.1.6 Geteilte Protokolle

3.0+

Protokolle für lang laufende Tests können sehr groß werden und enorm viel Speicher verbrauchen, insbesondere wenn viele Screenshots enthalten sind. Kompakte Protokolle können helfen, aber nicht genug um Tests über mehrere Tage zu ermöglichen, ohne das Protokoll komplett auszuschalten. Der beste Weg, dieses Problem zu umgehen, sind geteilte Protokolle.

Bei geteilten Protokollen entfernt QF-Test, immer wenn ein gewisser Teil des Tests abgeschlossen ist, das zugehörige Protokolle, speichert es als separate Datei und ersetzt es durch einen einzelnen Knoten, der einen Verweis auf das abgeteilte Protokoll enthält. Die abgeteilten Protokolle sind eigenständig und können unabhängig vom Hauptprotokoll betrachtet und archiviert werden. Normalerweise werden sie aber indirekt über das Hauptprotokoll angesprochen. Beim Navigieren durch das Hauptprotokoll, oder beim Erstellen von Reports, lädt QF-Test die benötigten abgeteilten Protokolle automatisch nach und entfernt sie wieder aus dem Speicher, wenn sie nicht mehr benötigt werden. Dadurch können auch extrem große Protokolle betrachtet werden, ohne sonderlich viel Speicher zu verbrauchen. Operationen wie Suche oder Reportgenerierung, die das gesamte Protokoll traversieren müssen, dauern natürlich etwas länger. Das Springen von Fehler zu Fehler geht aber nach wie vor schnell und das Laden des Hauptprotokolls wird drastisch verkürzt.

Es gibt zwei Wege, geteilte Protokolle zu speichern: Alles zusammen in einer einzelnen ZIP-Datei mit der Endung `.qzp` oder mit den abgeteilten Protokollen in einem eigenen Verzeichnis. Letzteres wird nach dem Hauptprotokoll benannt, wobei die Endung `.qrl` bzw. `.qrz` entfernt und stattdessen `_logs` angehängt wird. Innerhalb einer `.qzp` ZIP-Datei wird die Struktur identisch aufgebaut, so dass es möglich ist, diese manuell ein- oder auszupacken, ohne die internen Verweise im Protokoll zu zerstören. Diese Kompatibilität ist der Grund dafür, dass in der Standardeinstellung die abgeteilten Pro-

tokolle innerhalb einer ZIP-Datei komprimiert mit der Endung `.qrz` abgelegt werden. Dies ist zwar etwas weniger effizient als unkomprimierte `.qrl` Dateien, ermöglicht es dafür aber, die ZIP-Datei auszupacken, ohne dass dabei die Gesamtgröße explodiert.

Um geteilte Protokolle zu nutzen können Sie explizit die Punkte definieren, an denen das Protokoll aufgeteilt wird. Dies geschieht über das Attribut Name für separates Protokoll(648) eines Datentreiber(646), Testfall(599), Testfallsatz(606), Testaufruf(614) oder Testschritt(621) Knotens. Bei Verwendung in einem Datentreiber werden die Protokolle für jede Iteration abgeteilt, andernfalls das Protokoll des jeweiligen Knotens, der das Attribut definiert. Alternativ werden Protokolle automatisch ab einer gewissen Größe abgeteilt. Diese Funktionalität ist über die Option Minimale Größe für automatisches Teilen (kB)(583) konfigurierbar.

Bei der Verwendung von geteilten Protokollen empfiehlt es sich, die Option Kompakte Protokolle erstellen(590) auszuschalten, so dass alle Details im Protokoll erhalten bleiben. Dies braucht zwar etwas mehr Plattenplatz, ist aber sehr hilfreich bei der Fehlersuche.

Geteilte Protokolle sind außerdem sehr praktisch, um den Fortschritt eines Tests im Batchmodus zu verfolgen. In diesem Zusammenhang ist es besonders hilfreich, dass für die Dateinamen der abgeteilten Protokolle die gleichen Platzhalter wie für die Angabe des Protokollnamens auf der Kommandozeile verwendet werden können. Insbesondere kann so der Fehlerstatus des abgeteilten Protokolls Teil seines Dateinamens sein. Detaillierte Informationen finden Sie in der Dokumentation des Attributs Name für separates Protokoll(648).

7.1.7 Protokoll-Optionen

Die Erstellung und der Inhalt von Protokollen werden durch diverse Optionen gesteuert. Unter anderem kann eingestellt werden, ob kompakte oder detaillierte Protokolle geschrieben, ob der ganze Bildschirm und/oder die Applikationsfenster protokolliert oder ob Protokolle ganz unterdrückt werden. Alle Optionen sind detailliert in Abschnitt 41.11(578) beschrieben.

7.1.8 Eine Testsuite aus dem Protokoll erstellen

3.3+

Falls unterschiedliche Beteiligte in der Testentwicklung involviert sind, mag es in manchen Fällen von Nutzen sein, dass Sie aus einem Protokoll eine lauffähige Testsuite erstellen, um Testläufe schnell nachstellen zu können.

Sie können aus einem Protokoll eine Testsuite erstellen, wenn Sie im Protokoll auf einen beliebigen Knoten mit der rechten Maustaste klicken und Testsuite aus Protokoll erstellen aus dem Kontextmenü auswählen.

Nun wird eine neue Datei erstellt, welche unter Extrasequenzen(629) alle ausgeführten Schritte sowie die Fenster und Komponenten beinhaltet.

Hinweis

Es werden nur die ausgeführten und verwendeten Knoten in die neue Testsuite übernommen. Variablen werden sofort expandiert und nur der entsprechende Wert wird in der neu erstellten Testsuite abgelegt. Gliederungsknoten wie Prozeduren oder Kontrollstrukturen werden nicht erstellt.

Damit die Generierung funktioniert, müssen vor der Ausführung des Tests allerdings folgende Optionen (unter Protokoll -> Inhalt) gesetzt sein:

- Kompakte Protokolle erstellen(590) muss ausgeschaltet sein.
- Variablenexpansion protokollieren(587) muss eingeschaltet sein.
- Parentknoten von Komponenten protokollieren(587) muss eingeschaltet sein.

Falls Sie Zugriff auf alle vorhandenen Testsuiten haben, so können Sie die Informationen aus diesen Suiten nutzen und im Kontextmenü den Punkt Testsuite mit vorhandener Struktur erstellen auswählen. Der Unterschied zum obigen Verfahren ist, dass die Informationen über die Komponenten aus den entsprechenden Testsuiten anstatt aus dem Protokoll geholt werden. Deshalb ist es für diesen Modus auch nicht notwendig die Option Parentknoten von Komponenten protokollieren(587) eingeschaltet zu haben.

7.1.9 Protokolle zusammenführen

4.1+

Während der Testentwicklung könnten Sie in die Situation kommen, dass Sie einen Testreport erzeugt haben, der den Abschluss eines Testzyklus darstellen soll. Allerdings kann es immer wieder dazu kommen, dass einzelne Testfälle aufgrund subtiler Probleme nachgetestet werden müssen und Sie die Resultate der Nachtests eigentlich im Report anzeigen wollen. Für ein solches Szenario können Sie mehrere Protokolle zusammenführen und die ursprünglichen fehlerhaften Testläufe durch die Resultate des Nachtests ersetzen wollen. Dies erfolgt mittels Aufruf von der Kommandozeile.

Ein typischer Kommandozeilenaufruf hierfür sieht wie folgt aus:

```
qftest -batch -mergelogs -mergelogs.mode=replace
               -mergelogs.masterlog full_log.qzp
               -mergelogs.resultlog newresult_log.qzp rerun.qzp
```

Beispiel 7.1: Beispielaufruf um Protokolle zusammenzuführen

Der obige Aufruf liest die Resultate des Nachtlaufes aus dem Protokoll `rerun.qzp`, sucht nach dem Testfall im eigentlichen Protokoll `full_log.qzp` und speichert das angepasste Ergebnis im Protokoll `newresult_log.qzp`. Sie können hier auch den

Parameter `mergelogs.mode` auf den Wert `merge` setzen. Dieser Modus ersetzt die bestehenden Testfälle nicht, sondern fügt die neuen Testfälle in das Hauptprotokoll ein.

Ein zweiter Anwendungsfall besteht darin, dass Sie Protokolle aus mehreren Testläufen in ein Protokoll zusammenführen wollen, um auch nur einen Testreport am Ende erzeugt zu bekommen. Dies kann auch mittels Kommandozeilenaufruf bewerkstelligt werden und sieht wie folgt aus:

```
qftest -batch -mergelogs -mergelogs.mode=append
               -mergelogs.resultlog newresult_log.qzp run1.qzp run2.qzp
```

Beispiel 7.2: Beispielaufruf um Protokolle in eines zusammenzuführen

Dieser Aufruf liest die Protokolle `run1.qzp` und `run2.qzp` und führt diese im neuen Protokoll `newresult_log.qzp` zusammen. In diesem Modus ist der Parameter `mergelogs.masterlog` optional. Wenn der Parameter gesetzt wird, wird das entsprechende Protokoll als Wurzel für das Ergebnisprotokoll benutzt.

7.2 Der Debugger

Wie bei jeder komplexen Entwicklung wird es ab einem gewissen Punkt nötig sein, Probleme in einer Testsuite zu debuggen, die nicht mehr einfach durch Analysieren der Elemente und der Struktur einer Testsuite zu lösen sind. Zu diesem Zweck verfügt QF-Test über einen intuitiven Debugger. Sollten Sie bereits mit dem Debuggen von Programmen in Java oder anderen Programmiersprachen vertraut sein, werden Sie sich mit seiner Funktionsweise und Bedienung schnell zurechtfinden.

7.2.1 Aktivieren des Debuggers

Der QF-Test Debugger kann direkt aufgerufen werden, indem ein oder mehrere Knoten selektiert und der ”Einzelschritt ausführen”  oder der ”Gesamten Knoten ausführen” Button gedrückt werden. Die zugehörigen Tastaturkürzel und Menüeinträge sind F7 und F8 bzw. Debugger→Einzelschritt ausführen und Debugger→Gesamten Knoten ausführen Diese Operationen werden in Abschnitt 7.2.3(148) näher erläutert.

Wenn Sie einen Test normal über den ”Wiedergabe” Button starten (siehe Abschnitt 4.2(41)), wird der Debugger nur aktiv, wenn eine der folgenden Bedingungen eintritt:

- Ein benutzerdefinierter Breakpoint wird angetroffen. Näheres zu Breakpoints finden Sie in Abschnitt 7.2.4(149).
- Der Testlauf wird manuell durch Drücken des ”Pause” Buttons, von F9 oder über den Wiedergabe→Pause Menüeintrag unterbrochen.
- Eine Exception wird geworfen oder ein Fehler oder eine Warnung tritt auf und die entsprechende Option zum automatischen Unterbrechen ist gesetzt (vgl. Option Automatisch unterbrechen(577)).

Wenn der Debugger die Ausführung des Tests anhält, wird der Knoten, der als nächster ausgeführt wird, mit einem farbigen Rahmen um das Icon markiert. Die Farbe des Rahmens signalisiert den Grund der Unterbrechung. Nach manuellem Eingreifen, einem Breakpoint oder bei schrittweiser Ausführung ist der Rahmen schwarz. Orange, rot und fett rot signalisieren einen Stopp nach einer Warnung, einem Fehler oder einer Exception, entsprechend der Fehlermarkierung im Protokoll.

Hinweis

Wird der Debugger aufgrund einer Warnung, eines Fehlers oder einer Exception betreten, wird die Ausführung auf den Beginn des fehlerhaften Knotens zurückgesetzt, so dass Sie die Möglichkeit haben, die Ursache zu beseitigen und den Knoten erneut auszuführen. Wenn das nicht gewollt oder nicht möglich ist, können Sie den Knoten einfach überspringen (vgl. Abschnitt 7.2.3(148)).

In der Standardansicht (Workbench-Ansicht) kann der Debugger direkt vom normalen Testsuitefenster aus bedient werden. Informationen zum Öffnen des Debugger-Fensters in der Einzelfensteransicht finden Sie in Das separate Debugger-Fenster(150).

7.2.2 Anzeige der aktuellen Variablenwerte

Im Debug-Modus zeigt der untere Teil des Testsuite-Fensters eine Liste der Variablen(116) mit ihren Werten für den aktuellen Ausführungsstand. Die Liste ist in einen Primärstapel (oberer Teil) und in einen Sekundärstapel darunter aufgeteilt. Variablen mit dem gleichen Namen können an beliebig viele der Knoten gebunden sein. Der tatsächlich von QF-Test verwendete Wert ergibt sich durch die Reihenfolge der Knoten von oben nach unten. Das heißt, wenn Variablen mit dem gleichen Namen an mehrere Knoten gebunden wurde, wird der Wert derjenigen verwendet, deren Knoten sich am weitesten oben in der Liste befindet.

Ein Klick auf einen der Knoten übernimmt die Variablendefinitionen eines Knotens in den rechten Teil des Fensters, wo sie bearbeitet, neue Variablen hinzugefügt oder bestehende Variablen gelöscht werden können. Diese Änderungen wirken sich sofort auf den aktuellen Testlauf aus, sind aber temporärer Natur, das heißt sie werden nicht in die Variablendefinitionen des zugehörigen Knotens übernommen. Der Variablenwert ist nach der Bearbeitung immer eine Zeichenkette. Mit einem Doppelklick auf den Knoten

in der Variablenlist können Sie schnell zu dem Knoten in seiner Testsuite navigieren und dort den Wert permanent setzen.

Die Werte der Variablen sind im Normalfall Zeichenketten. Falls dies nicht zutrifft, wird der Typ des Variablenwerts in runden Klammern vor dem tatsächlich Wert dargestellt. Falls Sie solche Werte in der Variablenliste bearbeiten, werden sie automatisch in Zeichenketten umgewandelt - erkennbar daran, dass der Typ nicht mehr angezeigt wird.

Für den primären Stapel werden alle Knoten dargestellt, selbst wenn sie keine Variablen binden, für den sekundären Stapel nur solche, an die Variablen gebunden wurden.

Über die Einstellung Ansicht→Terminal→Baum dem Terminal vorziehen können Sie steuern, ob die Variablendefinitionen über die ganze Fensterbreite gehen oder nur den rechten Teil unter der Detailanzeige einnehmen.

7.2.3 Debugger Kommandos

Die meisten Debugger Kommandos entsprechen denen anderer Debugger. Einige Kommandos gehen dabei über die übliche Funktionalität hinaus.

Das schrittweise Debuggen einer Testsuite wird durch folgende drei Operationen ermöglicht:

- Der ”Einzelschritt ausführen” Button  (F7, Debugger→Einzelschritt ausführen) führt den aktuellen Knoten aus und setzt die Ausführungsmarke auf den nächsten auszuführenden Knoten, egal wo sich dieser in der Baumstruktur befindet. Diese Funktion ist z.B. hilfreich, um eine Prozedur oder eine Sequenz zu debuggen.

- Der ”Gesamten Knoten ausführen” Button  (F8, Debugger→Gesamten Knoten ausführen) führt den aktuellen Knoten und alle seine Kindknoten aus und setzt die Ausführungsmarke anschließend auf den nächsten Knoten der selben Ebene. Hiermit können Sie eine Prozedur oder eine Sequenz als ganzes ausführen, ohne sich einzeln durch ihre Kindknoten zu arbeiten.

- Der ”Bis Knotenende ausführen” Button  (Strg-F7, Debugger→Bis Knotenende ausführen) führt den aktuellen Knoten und alle folgenden Knoten auf der selben Ebene aus (inklusive ihrer Kindknoten) und setzt dann die Ausführungsmarke auf den nächsten Knoten der nächsthöheren Ebene. Diese Operation erlaubt es z.B., beim Debuggen einer Prozedur oder einer Sequenz diese ohne weitere Unterbrechung bis zum Ende auszuführen.

Die folgenden Funktionen erweitern den QF-Test Debugger um die Möglichkeit, Knoten einfach zu überspringen, ohne sie auszuführen.

- Der "Knoten überspringen" Button  (Shift-F9, Debugger→Knoten überspringen) springt ohne Ausführung über den aktuellen Knoten und verschiebt die Markierung auf den nächsten Knoten der selben Ebene.
- Der "Aus Knoten herausspringen" Button  (Strg-F9, Debugger→Aus Knoten herausspringen) beendet sofort die Ausführung der aktuellen Prozedur oder Sequenz und springt zum nächsten Knoten der nächsthöheren Ebene.

Noch mächtiger ist die Möglichkeit, den Test mit einem beliebigen anderen Knoten fortzusetzen, sogar in einer anderen Testsuite. Dabei werden so viele Informationen wie möglich im aktuellen Ausführungskontext erhalten, inklusive der gebundenen Variablen. Je näher der neue Zielknoten dem aktuellen Knoten ist, desto mehr Informationen können erhalten werden.

Sie können den Test durch Drücken von Strg-, oder über den Menüeintrag Wiedergabe→Ausführung hier fortsetzen bzw. den entsprechenden Eintrag im Kontextmenü beim selektierten Knoten fortsetzen. Dabei wird nur der aktuelle Knoten gewechselt, die Ausführung läuft nicht automatisch wieder an, sondern kann durch Einzelschritte oder andere Aktionen gezielt fortgeführt werden.

Folgende weitere Kommandos stehen zur Verfügung:

- Der "Exception erneut werfen" Button  (Debugger→Exception erneut werfen) ist nur dann aktiv, wenn der Debugger aufgrund einer Exception aufgerufen wurde. Damit können Sie die Exception weiterreichen und so von der Testsuite behandeln lassen, als wenn der Debugger gar nicht erst eingegriffen hätte.
- Der "Aktuellen Knoten finden" Button  (Debugger→Aktuellen Knoten finden) setzt die Selektion im Baum direkt auf den als nächstes auszuführenden Knoten. Dies ist eine nützliche Abkürzung, um nach einigem Herumwandern in der Testsuite zurück zur aktuellen Ausführung zu gelangen.

7.2.4 Breakpoints setzen und löschen

Das Setzen eines Breakpoints für einen Knoten veranlasst den Debugger, einen Testlauf vor dem Betreten dieses Knotens anzuhalten. Breakpoints werden im Baum durch ein "(B)" vor dem Namen des Knotens gekennzeichnet.

Breakpoints können mittels Strg-F8 oder dem Menüeintrag Debugger→Breakpoint an/aus individuell gesetzt oder gelöscht werden. Wenn Sie mit dem Debuggen fertig sind, können Sie eventuell übrig gebliebene Breakpoints mittels Debugger→Alle Breakpoints löschen entfernen. Dieses Kommando entfernt alle Breakpoints aus *allen* geöffneten Testsuiten.

Hinweis Breakpoints sind kurzlebig und werden daher nicht mit der Testsuite abgespeichert.

7.2.5 Das separate Debugger-Fenster

Wenn Sie mit der Einzelfensteransicht arbeiten (Workbench-Ansicht aktivieren[496] ist nicht gesetzt), muss der Debugger in einem eigenen Debugger-Fenster bedient werden. Dieses kann mittels Debugger→Debugger-Fenster öffnen nach Anhalten des Testlaufs geöffnet werden.

Das Debugger-Fenster kann auch durch Setzen der Option Debugger-Fenster immer öffnen[577] immer automatisch geöffnet werden, wenn der Debugger die Ausführung eines Tests unterbricht. Diese Option ist auch direkt über das Menü Debugger→Optionen zugänglich. Wenn Sie das Debugger-Fenster explizit öffnen oder schließen, wird diese Entscheidung für den Rest des Testlaufs respektiert und die Option so lange außer Kraft gesetzt.

Das Debugger-Fenster ist den normalen Testsuitefenstern sehr ähnlich. Sie können Knoten selektieren und deren Attribute in der Detailansicht bearbeiten. Es können jedoch keine Knoten entfernt oder hinzugefügt werden und es stehen keine Dateioperationen und kein Rekorder oder andere komplexe Funktionen zur Verfügung. An diese gelangen Sie sehr einfach, wenn Sie mittels Strg-T aus dem Debugger-Fenster direkt zum selben Knoten im Fenster seiner Testsuite springen. Sie finden diese Funktion auch als Knoten in Testsuite finden im Bearbeiten Menü oder dem Kontextmenü.

Kapitel 8

Aufbau und Organisation einer Testsuite

Zum Schreiben von aussagekräftigen und zuverlässigen Tests gehört mehr als nur das Aufnehmen und Wiedergeben von Sequenzen. Sie können eine Testsuite zwar innerhalb kurzer Zeit mit diversen Sequenzen anfüllen, werden aber über kurz oder lang den Überblick verlieren, wenn Sie nicht darauf achten, den Tests eine verständliche Struktur zu geben.

Bevor Sie mit der Aufnahme von Sequenzen beginnen, die Sie anschließend mittels der nachfolgend beschriebenen Elemente zu Testfällen und Testsätzen zusammenstellen, ist es wichtig, dass

- Sie eine gute Vorstellung davon haben, was Sie eigentlich testen wollen.
- Sie die richtigen Dinge testen.
- Ihre Tests zuverlässig und wiederholbar sind.
- Die Tests einfach gewartet werden können.
- Die Ergebnisse Ihrer Tests aussagekräftig sind.

Zur Strukturierung der Tests stellt QF-Test diverse Elemente zur Verfügung. Dies sind zum einen die in Kapitel 5(47) besprochenen Komponenten(930) und zum anderen die in diesem Kapitel vorgestellten Testsätze, Testfälle, Testschritte und Sequenzen sowie Events(775), Checks(805) etc.

8.1 Struktur der Testsuite

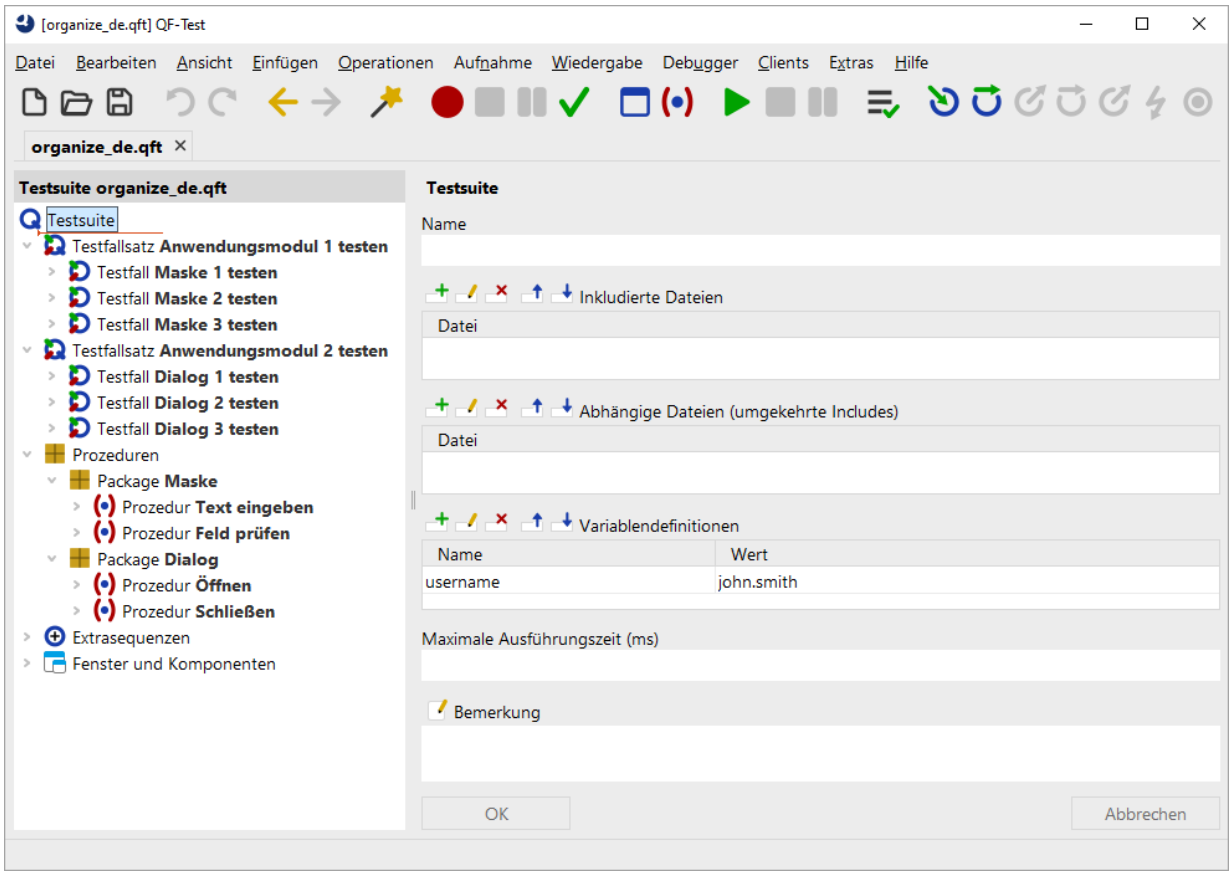


Abbildung 8.1: Struktur einer Testsuite

QF-Test bietet Strukturelemente auf unterschiedlichen Ebenen.

Auf oberster Ebene gibt es die Testsuite(595), in der die Tests sowie Komponenten abgespeichert werden. Jede Testsuite wird in einer eigenen Datei abgespeichert. Zur erleichterten Handhabung einer größeren Anzahl von Testsuite Dateien können diese in Projekte(180) zusammengefasst werden.

Die Testsuite hat eine feste Struktur, beginnend mit dem Testbereich, der beliebig viele Testfallsatz(606) Knoten enthalten kann, in die wiederum beliebig viele Testfall(599) Knoten oder auch weitere Testfallsätze eingefügt werden können.

Anschließend gibt es einen Bereich für Prozeduren(672). Als Strukturelement stehen hier Package(680) Knoten zur Verfügung, die beliebig viele Prozedur(672) oder weitere Package Knoten enthalten können.

Der folgende Bereich Extrasequenzen(629) bietet die Möglichkeit, im Rahmen der Testent-

wicklung beliebige Knoten abzuspeichern und auszutesten.

Der letzte Knoten Fenster und Komponenten(942) ist reserviert für die in den Tests benötigten Komponenten.

Testfälle selbst können wiederum mit Hilfe von Testschritt(621) und Sequenz(618) Knoten gegliedert werden.

Ein weiteres wichtiges Strukturelement sind Vorbereitung(638) und Aufräumen(641) Knoten. Diese dienen der Erstellung von Vor- und Nachbedingungen und können ihrerseits in Abhängigkeit Knoten(161) zusammengefasst werden. Abhängigkeit(630) Knoten bieten darüber hinaus auch Strukturen für die Behandlung von Fehlern und unvorhergesehenen Ereignissen.

Die eigentliche Testausführung geschieht über Events(775) wie zum Beispiel Mausklicks, Checks(805), Abfragen(839), Knoten zur Ablaufsteuerung(684) und solche, die Prozesse(724) steuern sowie weitere, in Verschiedenes(851) beschriebene Knoten.

Die Verknüpfung der Tests mit den Testdaten erfolgt über Datentreiber(646).

8.2 Testfallsatz und Testfall Knoten

8.2.1 Verwaltung von Tests mit Hilfe von Testfallsatz und Testfall Knoten

2.0+

Mit den Knoten Testfallsatz(606) und Testfall(599) bietet QF-Test eine einfache, pragmatische Form der Testfallverwaltung direkt innerhalb von QF-Test. Sie sind die wichtigsten Bausteine zur Strukturierung einer Testsuite. Sie bieten die Möglichkeit mittels Abhängigkeit Knoten(161) die Testfälle so zu erstellen, dass sie völlig unabhängig voneinander sind. Notwendige Aufräumarbeiten werden bei entsprechend aufgebauten Abhängigkeiten ebenso automatisch erledigt wie die Vorbereitungen für den aktuellen Test sowie die Fehlerbehandlung.

Auch hinsichtlich der Ergebnisdokumentation kommt den Testfall Knoten eine zentrale Rolle zu. In den Reports ist der Testfall das Strukturelement, auf dem die Fehlerstatistik und auch die Fehlerdokumentation basiert.

8.2.2 Konzepte

Ein Testfall(599) Knoten entspricht konzeptuell einem einzelnen elementaren Testfall und ist damit das entscheidende Bindeglied zwischen Testplanung, Testdurchführung und Testauswertung. Mit Hilfe von Abhängigkeit Knoten(161) können Testfälle so voneinander isoliert werden, dass sie in beliebiger Reihenfolge ausgeführt werden können und auf

unvorhergesehenes Verhalten reagiert werden kann. Damit ist es möglich, Teile von funktionalen Tests als Build-Test auszuführen oder etwa einen erneuten Testlauf durchzuführen, bei dem nur die fehlgeschlagenen Testfälle wiederholt werden.

Testfallsätze sind im Prinzip einfach Sammlungen von zusammengehörigen Testfällen, die ähnliche Vor- und Nachbedingungen haben. Testfallsätze können auch verschachtelt werden. Die Struktur der Testfallsatz und Testfall Knoten ist damit ähnlich der Struktur der Prozeduren und Packages(157) Knoten. Der Testsuite(595) Knoten kann als spezielle Form eines Testfallsatzes angesehen werden.

Testsuite, Testfallsatz und Testfall Knoten können von jedem anderen Ort aus mit Hilfe eines Testaufruf(614) Knotens aufgerufen werden. Auf diesem Weg können sehr einfach Tests erstellt und verwaltet werden, die nur eine Untermenge von bestehenden Tests ausführen. Mit Blick auf den Report sollten Testaufruf Knoten nicht innerhalb eines Testfall Knotens ausgeführt werden, weil damit Testfälle verschachtelt würden und vom Report nicht mehr richtig aufgelöst werden könnten. In diesem Fall wird eine Warnung ausgegeben.

8.2.3 Variablen und besondere Attribute

Defaultwert

Da Testfallsatz und Testfall Knoten durch Testaufruf Knoten aufgerufen werden können, verfügen sie analog zum Prozedur(672) Knoten über einen Satz von Standardwerten für die Parameter. Diese werden auf dem Sekundärstapel gebunden und können über einen Testaufruf oder den Kommandozeilenparameter `-variable <Name>=<Wert>`(994) überschrieben werden.

Variablendefinitionen

Für einen Testfall kann man zudem Variablen definieren, die auf dem Primärstapel gebunden werden und nicht von außen mittels Testaufruf oder Kommandozeile überschrieben werden können. Primär- und Sekundärstapel sind in Abschnitt 6.2(118) näher beschrieben.

Charakteristische Variablen

In der Liste der Charakteristischen Variablen können die Namen der Variablen angegeben werden, deren Werte zum Beispiel bei datengetriebenem Testen für einen Durchlauf des jeweiligen Testfallsatz oder Testfall Knotens charakteristisch sind. Die Werte dieser

Variablen werden in Protokoll und Report angezeigt und helfen somit bei der Analyse von Problemen.

Bedingung

Ein weiteres nützliches Attribut ist die Bedingung, vergleichbar mit der Bedingung[694] eines If[693] Knotens. Falls die Bedingung nicht leer ist, wird der Knoten nur ausgeführt, falls der Wert des Ausdrucks wahr ist. Andernfalls wird der Test übersprungen.

Fehlschlagen erwartet wenn...

Manchmal wird das Fehlschlagen eines Testfalls für eine bestimmte Zeitdauer erwartet, z. B. wenn dieser erstellt wird, bevor ein Feature oder Bug-Fix im SUT verfügbar ist. Das Attribut Fehlschlagen erwartet wenn... erlaubt es, solche Testfälle zu markieren, so dass diese getrennt gezählt werden und nicht in die prozentuale Fehlerstatistik eingehen.

8.3 Sequenz und Testschritt Knoten

Primäre Bausteine einer Testsuite sind die Sequenz[618] und Testschritt[621] Knoten, die ihre Unterknoten einen nach dem anderen ausführen. Sie dienen der Strukturierung der Unterknoten eines Testfall.

Der Unterschied zwischen Sequenz und Testschritt Knoten besteht darin, dass Testschritt Knoten im Report protokolliert werden, Sequenz Knoten hingegen nicht.

8.4 Vorbereitung und Aufräumen Knoten

Neben den eigentlichen Testschritten enthält jeder Testfall auch Vor- bzw. Nachbedingungen. Die Vorbedingungen werden in Vorbereitung Knoten implementiert, Nachbedingungen in Aufräumen Knoten. In einfachen Fällen werden diese als Unterknoten eines Testfall oder Testfallsatz Knotens eingefügt.

QF-Test bietet allerdings noch eine mächtigere Variante für das Behandeln von Vor- und Nachbedingungen nämlich Abhängigkeit Knoten[161]. Diese ermöglichen nicht nur die Wiederverwendung, sondern bieten auch Möglichkeiten an, um auf unterschiedliche Verhalten während des Testlaufes zu reagieren.

Testfall Knoten mit durchdachten, passenden Vorbereitung und Aufräumen Knoten haben folgende wichtige Eigenschaften:

- Der Testfall kann unabhängig davon, ob vorhergehende Testfälle erfolgreich waren - oder eben nicht - ausgeführt werden.
- Testsuite bzw. der Testfallsatz können an beliebiger Position um weitere Testfall Knoten ergänzt werden, ohne andere Testfälle zu beeinflussen.
- Sie können einen Testfall erweitern oder ihn einfach ausprobieren, ohne vorhergehende Testfälle ausführen zu müssen, um das SUT in den passenden Zustand zu bringen.
- Sie können jede beliebige Teilmenge der Testfall Knoten ausführen, wenn Sie nicht den gesamten Testfallsatz benötigen.

Im einfachsten Fall wird der gleiche Ausgangszustand für alle Testfälle eines Testfallsatzes benötigt. Dies kann mittels der folgenden Struktur realisiert werden:

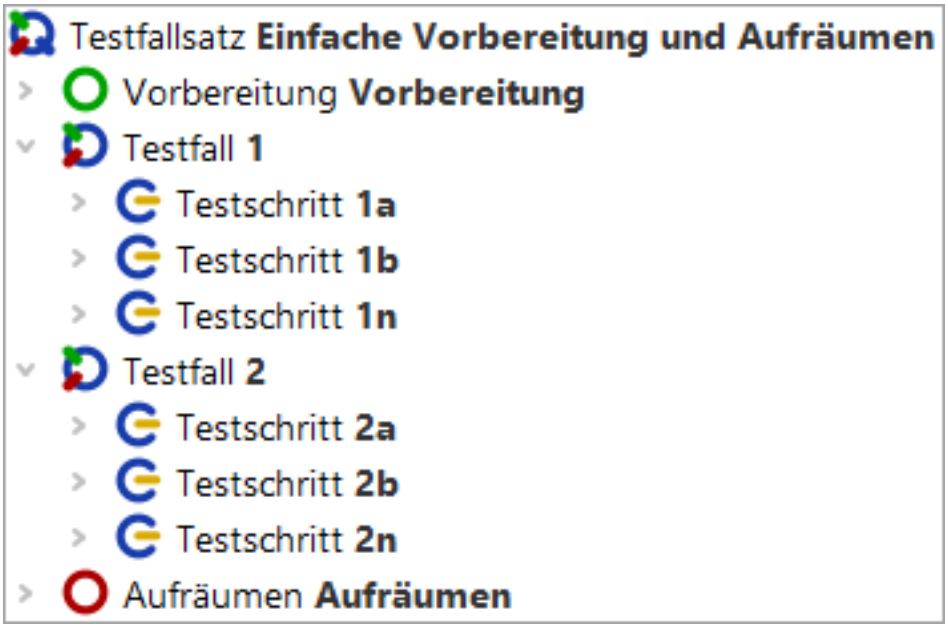


Abbildung 8.2: Teststruktur mit einfacher Vorbereitung und Aufräumen

Hier wird zu Beginn jedes Testfalls die Vorbereitung und nach Ende des Testfalls das Aufräumen ausgeführt. Im Protokoll ist die Ausführungsreihenfolge genau zu erkennen:

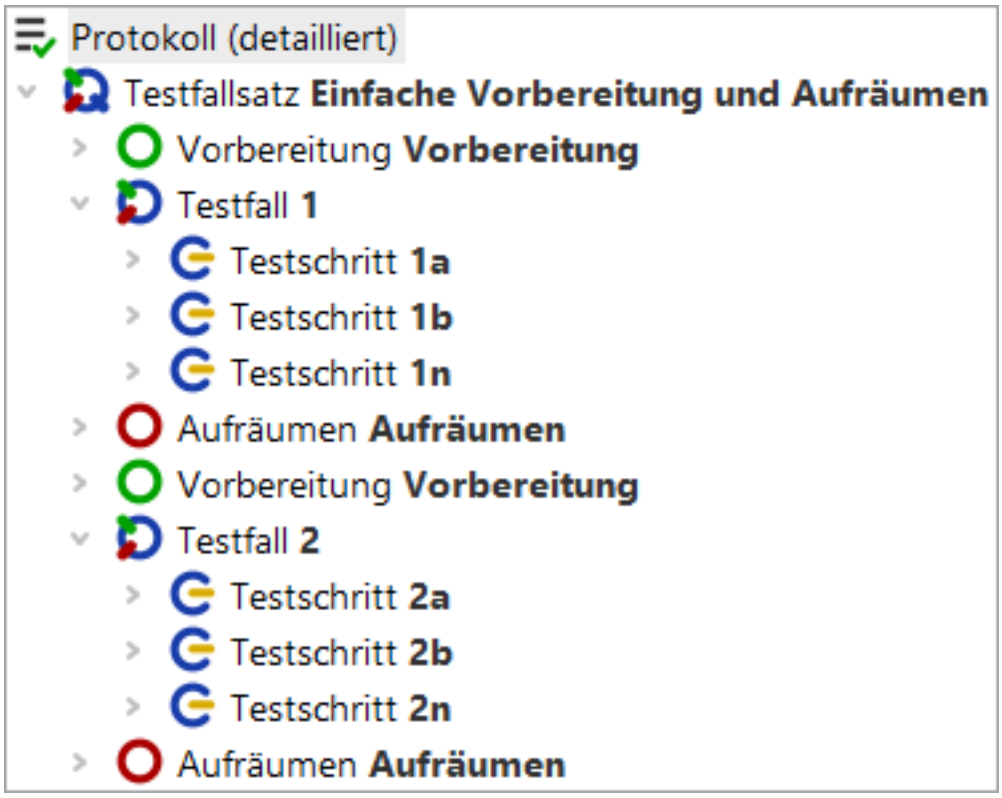


Abbildung 8.3: Ausführungsreihenfolge bei einfacher Vorbereitung und Aufräumen

Beim Testfallsatz Knoten werden Vorbereitung und Aufräumen Knoten vor und nach jedem im Testfallsatz enthalten Knoten (Testfall und Testfallsatz) ausgeführt. Bei einem Testfall Knoten werden hingegen Vorbereitung und Aufräumen nur einmal ganz zu Beginn und Ende ausgeführt.

Falls die Vorbereitung im obigen Beispiel den Start der Anwendung und die Aufräumen Knoten das Beenden dieser beinhaltet, würde vor jedem Testfall die Anwendung gestartet und danach wieder gestoppt. Diese Vorgehensweise ist wenig effizient. Daher bietet QF-Test eine umfassendere Struktur, den Abhängigkeit Knoten[161], zur Herstellung der Vorbedingungen und dem effizienten Umgang mit Aufräumarbeiten.

8.5 Prozeduren und Packages

In mancher Hinsicht ist das Schreiben von Tests dem Programmieren nicht unähnlich. Nachdem die ersten Schritte gemeistert sind, tendieren Tests ebenso wie Programmcode dazu, unkontrolliert auszuufern. Das funktioniert so lange ganz gut, bis irgendein grundlegender Baustein, auf den man sich verlassen hat, geändert werden muss. Ohne

saubere Struktur brechen Programme ebenso wie Tests an diesem Punkt in sich zusammen, da der Aufwand für die Anpassung an die neue Situation höher ist als gleich von vorne anzufangen.

Ein Schlüsselpunkt, um dieses Problem zu verhindern, ist die Vermeidung von Redundanz. Wenn Sie sich zu sehr auf die Aufnahmefunktion allein verlassen, besteht die Gefahr, genau diese Redundanz zu erzeugen. Ein Beispiel: Sie nehmen verschiedene Sequenzen auf, die mit den Komponenten eines Dialogs interagieren. Um diese Sequenzen möglichst unabhängig voneinander zu halten, beginnen Sie jede Sequenz damit, dass Sie den Dialog öffnen. Analog beenden Sie die Sequenzen mit dem Schließen des Dialogs. Das ist eigentlich eine gute Idee, erzeugt aber Redundanz, da die Events zum Öffnen und Schließen des Dialogs in jeder einzelnen Sequenz vorkommen. Stellen Sie sich vor, was passiert, wenn sich das SUT auf eine Weise ändert, die dazu führt dass dieser Teil nicht mehr funktioniert, z.B. weil erst ein kleines Bestätigungsfenster geschlossen werden muss, bevor der Dialog geschlossen werden kann. Jetzt müssen Sie durch die gesamte Testsuite gehen, alle Stellen finden, an denen der Dialog geschlossen wird und jede einzelne Stelle an die neuen Gegebenheiten anpassen. Der blanke Horror...

Um noch einmal auf die Analogie zurückzukommen: Die entsprechende Art der Programmierung wird *Spaghetti Programmierung* genannt und führt zu der gleichen Art von Wartungsproblemen. Diese können vermieden werden, wenn identische Teile an einer Stelle zusammengefasst werden, wo sie bei Bedarf aufgerufen werden. Eine Anpassung an neue Gegebenheiten erfordert dann nur noch Modifikationen an dieser einen Stelle.

Abbildung 8.4: Packages und Prozeduren

QF-Test verfügt über einen Satz von Knotentypen, der diese Art der Modularisierung ermöglicht. Dabei handelt es sich um die Knoten Prozedur(672), Prozeduraufruf(675) und Package(680). Eine Prozedur ist einer Sequenz(618) sehr ähnlich, abgesehen davon, dass der Name(673) der Prozedur zur Referenzierung durch einen Prozeduraufruf Knoten dient. Ein Prozeduraufruf wird ausgeführt, indem die Kontrolle an die entsprechende Prozedur übergeben wird. Mit dem letzten Kindknoten der Prozedur ist dann auch der Prozeduraufruf beendet.

Packages sind dazu da, Prozeduren noch mehr Struktur zu geben, indem zusammengehörende Prozeduren in einem Package zusammengefasst werden können. Eine Hierarchie von Packages und Prozeduren ist unter dem Knoten Prozeduren angesiedelt.

Eine Prozedur, die immer exakt die selben Schritte ausführt, egal wie und woher sie aufgerufen wird, ist nur von sehr begrenztem Nutzen. Um obiges Beispiel fortzuführen, könnte eine Prozedur beispielsweise den Dialog öffnen und seine Felder mit einigen Werten vorbelegen. Diese Werte sollten dann natürlich nicht hart in der Prozedur verdrahtet sein, sondern wir wollen sie beim Prozeduraufruf individuell festlegen. Zu diesem Zweck können Parameter für eine Prozedur definiert werden. Im Prozeduraufruf werden dann Werte für diese Parameter festgelegt. Diese sind nur für genau diese Ausführung der Prozedur gültig. Eine ausführliche Beschreibung zur Definition von Parametern und allgemein Variablen in QF-Test, finden Sie in Kapitel 6(116). Zum besseren Verständnis ihres Zusammenspiels sollten Sie auch einen Blick auf die detaillierte Dokumentation der Prozedur(672) und Prozeduraufruf(675) Knoten werfen.

Eine Testsuite-Bibliothek mit allgemein nützlichen Prozeduren wird von QF-Test unter dem Namen `qfs.qft` zur Verfügung gestellt. Dieser Bibliothek ist ein ganzes Kapitel des Tutorials gewidmet. Abschnitt 26.1(359) erklärt, wie Sie die Bibliothek direkt in Ihre Testsuiten einbinden.

8.5.1 Lokale Prozeduren und Packages

3.1+

Wenn Sie in mehreren Testsuiten arbeiten, dann könnten Sie in die Situation kommen, dass Sie manche wiederverwendbaren Sequenzen bzw. Testschritte nur von einer bestimmten Testsuite aus ansprechen möchten. Wenn Sie solche lokale Prozeduren erstellen wollen, dann müssen Sie als erstes Zeichen des Prozedurnamens ein '_' definieren. Das '_' markiert die Prozedur als lokal in der jeweiligen Testsuite.

Aufrufe von lokalen Prozeduren können nur innerhalb der Testsuite eingefügt werden, in der diese Prozedur definiert ist. Sie können dasselbe Konzept auch für lokale Packages nutzen.

8.5.2 Relative Prozeduren

3.1+

Wenn Sie Prozeduren in anderen Prozeduren aufrufen, könnte es manchmal von Vorteil sein, nicht den vollen Namen der Prozedur definieren zu müssen.

So genannte 'relative' Prozeduraufrufe können nur in Packages eingefügt werden, welche das Attribut Grenze für relative Aufrufe (siehe auch Grenze für relative Aufrufe(681)) gesetzt haben. Der Aufbau von relativen Aufrufen sieht wie folgt aus:

Ebene	Aufruf
Prozeduren auf derselben Ebene	.Name der Prozedur
Prozeduren eine Ebene höher	..Name der Prozedur
Prozeduren eine Ebene tiefer	.Name des Package.Name der Prozedur

Tabelle 8.1: Relative Prozeduraufrufe

Wie Sie sehen können, wird für jede Ebene einfach ein Punkt hinzugefügt. Eine Prozedur zwei Ebenen höher, wird also mittels drei Punkten referenziert (Die aktuelle Ebene zählt auch mit).

8.5.3 Einfügen von Prozeduraufruf Knoten

Sie sollten Tests in einzelnen Testschritten organisieren, wobei idealerweise jeder Testschritt einem QF-Test Prozedur Knoten entspricht. QF-Test bietet unterschiedliche Methoden um Prozeduraufruf Knoten anzulegen:

1. Mittels Auswahl im Menü Einfügen→Prozeduren→Prozeduraufruf
2. Mittels Rechtsklick und Auswahl von Knoten einfügen→Prozeduren→Prozeduraufruf
3. Wenn Sie einen Prozedur Knoten an eine Stelle, wo ein Prozeduraufruf eingefügt werden soll, kopieren, wird dieser eingefügt.
4. Mittels Drag&Drop Operation, d.h. Ziehen eines Prozedur Knotens an die gewünschte Stelle.
5. Mittels Tastaturkürzel Strg-A
6. Mittels der Umwandlung einer Sequenz oder eines Testschritt in eine Prozedur, wie in Abschnitt 8.5.5(161) beschrieben. Tastaturkürzel Strg-Shift-P

Diese Methoden funktionieren auch für Bezug auf Abhängigkeit Knoten bis auf die letzten beiden.

8.5.4 Parametrisieren von Knoten

3.1+

Sie können Parameter von Prozeduren, Abhängigkeiten oder Testfälle automatisch mittels des Menüs Operationen→Knoten parametrisieren erstellen.

Der Parametrisierungsdialog ermöglicht es Ihnen noch weitere Details über das Erstellen der Parameter zu definieren, z.B. ob nur Parameter für Texteingaben oder Checkknoten zu erstellen sind.

8.5.5 Konvertieren von Sequenzen und Testschritte in Prozeduren

3.0+

Diese Konvertierung kann sehr hilfreich sein um sofort während der Entwicklung Prozeduren zu erstellen. Unter Extrasequenzen können Sie Sequenzen in Prozeduren konvertieren und in den Prozeduren Bereich verschieben.

3.1+ Wenn Sie eine Sequenz oder Testschritt unter einem Testfall konvertieren, dann erstellt QF-Test automatisch eine Prozedur und fügt an Stelle der Sequenz oder des Testschritt den entsprechenden Prozeduraufruf ein.

Die Konvertierung kann mittels dem Menü Operationen→Knoten konvertieren in→Prozedur oder über das Tastaturkürzel Shift-Strg-P angestoßen werden.

8.6 Abhängigkeit Knoten

Video Video:

Abhängigkeiten
https://www.qftest.com/de/yt/abhaengigkeiten-basics-45.html

8.6.1 Funktionsweise

Abhängigkeiten stellen eine mächtige und optimierte Variante für die Herstellung von Vor- und Nachbedingungen dar. Sie sind unverzichtbar zur Ausführung von Tests im QF-Test Daemon-Modus[1276]. Ihre prinzipielle Funktionsweise ist wie folgt:

1. Erstellung einer Liste der für den Testfall benötigen Abhängigkeiten.
2. Vergleich der aktuellen Abhängigkeitenliste mit der des vorangehenden Testfalls.
3. Ausführung der Aufräumen Knoten derjenigen Abhängigkeiten, die in der Abhängigkeitenliste des aktuellen Testfalls nicht mehr vorkommen oder deren Charakteristische Variablen[168] sich verändert haben sowie derjenigen Abhängigkeiten, die auf diesen basieren.
4. Ausführung **aller** Vorbereitung Knoten der aktuellen Abhängigkeitenliste.

Testfälle als auch andere Abhängigkeiten können sich mittels Bezug auf Abhängigkeit(635) Knoten auf Abhängigkeit(630) Knoten, die im Prozeduren(672) Bereich abgelegt wurden, beziehen. Somit können Vorbereitung und Aufräumen Knoten, die sich in einem Abhängigkeiten Knoten befinden von mehreren Testfällen genutzt werden - im Gegensatz zu denjenigen, die direkt in einem Testfall oder Testfallsatz Knoten liegen.

Für das Verständnis der Arbeitsweise von Abhängigkeiten ist es hilfreich, sich das Vorgehen eines Testers bei manuellen Tests anzusehen: Zunächst führt er alle notwendigen Vorbereitungen und dann den ersten Testfall aus. Wenn bei der Ausführung des Testfalls Fehler auftreten, wird er gegebenenfalls Fehlerbehandlungsroutinen ausführen. Als nächstes prüft er, welche Vorbedingungen für den nächsten Test gelten und wird **erst dann** eventuelle Aufräumarbeiten durchführen. Dabei wird er nur das aufräumen, was für nächsten Testfall nicht mehr benötigt wird. Als nächstes geht er **alle** Vorbedingungen durch und prüft, ob die notwendigen Voraussetzungen noch bestehen oder ob zum Beispiel auf Grund von Fehlern im vorhergehenden Testfall wiederhergestellt werden müssen. Falls das SUT in einen stark fehlerhaften Zustand versetzt wurde, führt er weitere Aufräumarbeiten aus bevor er die Vorbedingungen für den nächsten Testfall herstellen kann.

Genau diese Vorgehensweise können Sie über QF-Test Abhängigkeiten abbilden.

Mit den Abhängigkeiten werden die Nachteile der klassischen Vorbereitung und Aufräumen Knoten(155), dass zum einen die Vorbereitung Knoten nur über die Verschachtelungen von Testfallsätzen geschachtelt werden können und zum anderen in jedem Fall der Aufräumen Knoten ausgeführt wird, behoben. Beides ist nicht sonderlich effizient. Abhängigkeit Knoten geben nicht nur eine Antwort auf diese beiden Punkte sondern bieten zudem Strukturelemente für die Behandlung von Fehlern und unvorhergesehenen Ereignissen.

Abhängigkeiten werden in vielen mitgelieferten Beispieltestsuiten verwendet, zum Beispiel:

- In `doc/tutorial` die Testsuite namens `dependencies.qft`. Die Abhängigkeiten in dieser Testsuite werden ausführlich im Tutorial in Kapitel 16 erläutert.
- Im Verzeichnis `demo/carconfigSwing` die Testsuite `carconfigSwing_de.qft`, die ein realistisches Beispiel enthält.
- Die SWT Demo Suite namens `swt_addressbook.qft`, die für SWT-Anwender ein Beispiel bietet.
- In `demo/eclipse` die Testsuite namens `eclipse.qft`, in der Sie eine verschachtelte Abhängigkeit finden.
- In `doc/tutorial` verwendet das Datentreiber-Demo `datadriver.qft` Abhängigkeiten.

Wenn Sie im Debugger in Einzelschritten durch diese Testsuiten gehen und sich die zugehörigen Variablenbindungen und Protokolle ansehen, sollten Sie ein gutes Gefühl für diese Technik bekommen. Bitte beachten Sie, dass Sie veränderte Testsuiten am besten in einem projektspezifischen Ordner speichern.

8.6.2 Verwendung von Abhängigkeiten

Abhängigkeiten können an zwei Stellen definiert werden:

- Allgemeine Abhängigkeiten, die oft wiederverwendet werden und als Grundlage für weitere Abhängigkeiten dienen, können genau wie ein Prozedur(672) Knoten implementiert und unterhalb des Prozeduren(682) Knotens platziert werden, zum Beispiel in einem Package(680) Knoten namens "Abhängigkeiten". Der qualifizierte Name wird genau wie der einer Prozedur gebildet. Abhängigkeiten können analog zu einem Prozeduraufruf(675) durch einen Bezug auf Abhängigkeit Knoten referenziert werden.
- Alternativ können Abhängigkeiten Knoten am Beginn eines Testsuite, Testfallsatz oder Testfall Knotens implementiert werden.

Zusätzlich zu ihrer eigenen Abhängigkeit können Testfälle und Testfallsätze die Abhängigkeit von ihrem Parentknoten erben.

Eine Abhängigkeit sollte sich jeweils nur um eine Vorbedingung kümmern. Hat diese ihrerseits Vorbedingungen, sollten diese zu Grunde liegenden Schritte von anderen Abhängigkeiten übernommen werden. Dies kann implizit durch Erben der Abhängigkeit von einem Parentknoten oder explizit durch einen Bezug auf Abhängigkeit Knoten geschehen.

Die eigentliche Implementierung der Vor- und Nachbedingungen geschieht in Vorbereitung und Aufräumen Knoten innerhalb der Abhängigkeit.

Enthält ein Testfallsatz oder Testfall Knoten sowohl Abhängigkeit Knoten als auch Vorbereitung und Aufräumen Knoten, wird die Abhängigkeit zuerst aufgelöst. Vorbereitung und Aufräumen Knoten haben keinen Einfluss auf den Stapel von Abhängigkeiten.

8.6.3 Abhängigkeiten - Ausführung und Stapel von Abhängigkeiten

Die Ausführung einer Abhängigkeit gliedert sich in drei Phasen:

1. Erstellung einer Liste benötigter Abhängigkeiten und Abgleich mit der Liste früher ausgeführter Abhängigkeit Knoten
2. Ausführung von Aufräumen Knoten, falls notwendig
3. Ausführung der Vorbereitung Knoten aller benötigen Abhängigkeiten

Die im vorliegenden Kapitel verwendeten Beispiele beziehen sich alle auf Testfälle, die eine Kombination der folgenden Vorbedingungen und Nachbereitungen erfordern:

Beispiel

Abhängigkeit A: Anwendung gestartet
Vorbereitung: Anwendung bei Bedarf starten
Aufräumen: Anwendung stoppen

Abhängigkeit B: Anwender angemeldet
Vorbereitung: Anwender bei Bedarf anmelden
Aufräumen: Anwender abmelden

Abhängigkeit C: Anwendungsmodul 1 geladen
Vorbereitung: Anwendungsmodul 1 bei Bedarf laden
Aufräumen: Anwendungsmodul beenden

Abhängigkeit D: Anwendungsmodul 2 geladen
Vorbereitung: Anwendungsmodul 2 bei Bedarf laden
Aufräumen: Anwendungsmodul 2 beenden

Abhängigkeit E: Dialog in Modul 2 geöffnet
Vorbereitung: Dialog in Modul 2 bei Bedarf öffnen
Aufräumen: Dialog schließen

Abhängigkeit C hängt von B ab, B wiederum von A.
Abhängigkeit E hängt von D ab, D von B und damit auch von A.

Vor der Ausführung eines Testfall Knotens prüft QF-Test, ob dieser einen Abhängigkeit Knoten besitzt oder erbt. In diesem Fall prüft QF-Test, ob der Abhängigkeit Knoten seinerseits weitere Abhängigkeiten einbezieht. Auf Basis dieser Analyse erstellt QF-Test zunächst eine Liste der auszuführenden Abhängigkeiten. Dies entspricht Schritt 1 des nachfolgenden Beispiels.

Als nächstes prüft QF-Test, ob bereits für vorhergehende Testfälle Abhängigkeiten ausgeführt wurden. Falls ja, vergleicht QF-Test die Liste der ausgeführten Abhängigkeiten mit den aktuell benötigten. Für nicht mehr benötigte Abhängigkeiten werden nun die Aufräumen Knoten ausgeführt. Danach durchläuft QF-Test alle Vorbereitung Knoten, wobei mit den grundlegenden Abhängigkeiten begonnen wird. Der Name des ausgeführten Abhängigkeiten Knotens wird jeweils vermerkt. Diese Liste wird Abhängigkeitenstapel genannt.

Beispiel: Testfall 1
Test von Anwendungsmodul 1.
Es handelt sich um den ersten ausgeführten Testfall Knoten.

1. Schritt:
   Die Analyse der Abhängigkeiten ergibt eine Liste mit den Abhängigkeiten A-B-C (Anwendung gestartet, Anwender angemeldet, Modul 1 geladen).
2. Schritt
   Abgleich der Abhängigkeitenliste mit dem Abhängigkeitenstapel: In diesem Beispiel ist der Abhängigkeitenstapel leer, da noch kein Testfall ausgeführt wurde.
3. Schritt
   Ausführung der Vorbereitung Knoten, beginnend mit A (Anwendung starten), dann B (Anwender anmelden, Anwendername: Standard), und C (Modul 1 laden).
   Auf dem Abhängigkeitenstapel wird die Ausführung der Abhängigkeiten A-B-C vermerkt.
4. Schritt
   Ausführung des Testfall Knotens.

Diese Schritte lassen sich im Protokoll genau nachverfolgen:

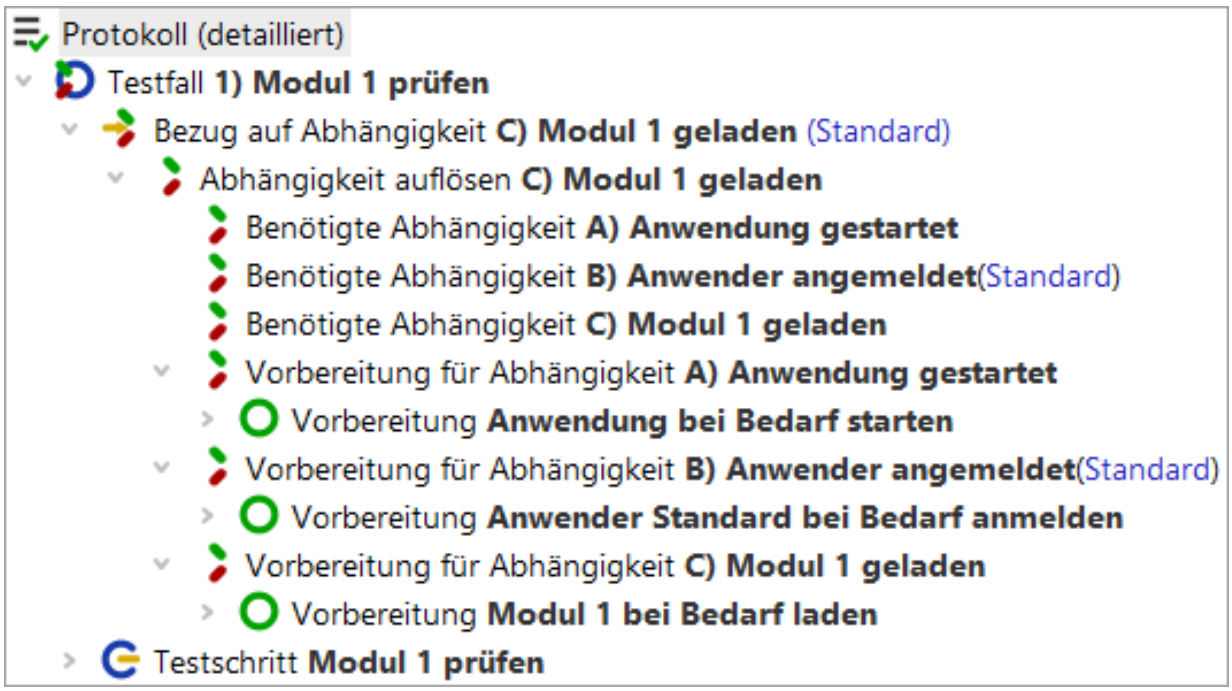


Abbildung 8.5: Stapel von Abhängigkeiten A-B-C

Nach Ausführung des Testfall Knotens verbleibt die Anwendung zunächst in dem Zustand, in den der Testfall sie versetzt hat. Erst nach der Analyse der Abhängigkeiten des

nächsten Testfall Knotens werden bei Bedarf Abhängigkeiten abgebaut - in umgekehrter Ausführungsreihenfolge der Vorbereitung Knoten. Anschließend werden die Vorbereitung Knoten aller benötigen Abhängigkeiten durchlaufen. Dieser Punkt ist sehr wichtig. Es werden also nicht einfach die restlichen Vorbereitung Knoten ausgeführt, sondern die aller einbezogenen Abhängigkeit Knoten. Der Grund hierfür ist, dass es, wie bereits erwähnt, in der Natur des Testens liegt, dass es zu Fehlerzuständen kommen kann. Daher wird vor Ausführung eines Vorbereitung Knotens geprüft, ob auch die jeweils zugrunde liegenden Voraussetzungen erfüllt sind. Die Vorbereitung Knoten sollten daher so aufgebaut sein, dass zunächst geprüft wird, ob die Voraussetzung bereits gegeben ist und nur bei Bedarf der gesamte Vorbereitung Knoten durchlaufen wird.

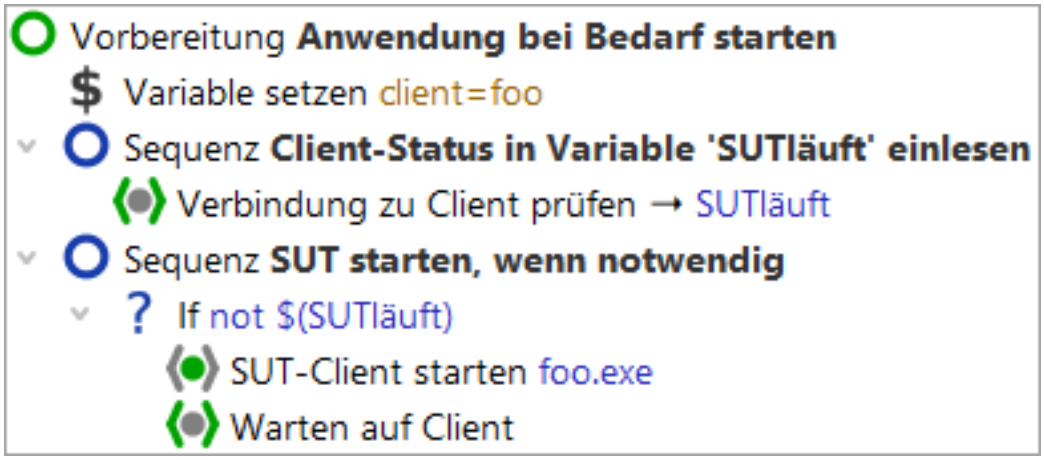


Abbildung 8.6: Typischer Vorbereitung Knoten

Außerdem sollten Vorbereitung Knoten und Aufräumen Knoten immer in der Lage sein, Programmzustände, die auf Grund von Fehlern nicht dem Sollzustand entsprechen, zu bereinigen, so dass der nachfolgende Testfall nicht von vorangehenden Fehlern beeinträchtigt wird. Auch sollte ein Aufräumen Knoten nicht zu einem Fehler führen, wenn der angestrebte Zustand bereits besteht. Wenn das SUT zum Beispiel beendet werden soll, darf es zu keinem Fehler kommen, wenn es, aus welchem Grund auch immer, schon vor Ausführung des Aufräumen Knotens nicht mehr läuft.

Beispiel: Testfall 2

Test eines Dialogs in Anwendungsmodul 2

1. Schritt:
   Die Analyse der einbezogenen Abhängigkeiten ergibt die Abhängigkeitenliste A-B-D-E (Anwendung gestartet, Anwender angemeldet, Modul 2 geladen, Dialog geöffnet).
2. Schritt:
   Abgleich der Abhängigkeitenliste mit dem Abhängigkeitenstapel: Abhängigkeit

C ist keine Voraussetzung für Testfall 2. Daher wird der Aufräumen Knoten der Abhängigkeit C ausgeführt (Beenden von Modul 1).

3. Schritt:
   Ausführung der Vorbereitung Knoten, beginnend mit A (Es wird erkannt, dass die Anwendung bereits läuft und der Rest des Vorbereitung Knotens übersprungen.), dann B (Es wird erkannt, dass der Anwender bereits angemeldet ist und der Rest des Vorbereitung Knotens übersprungen.), dann D (Es wird erkannt, dass Modul 2 noch nicht geladen ist - der Vorbereitung Knoten wird komplett ausgeführt.), dann E (analog zu D).
4. Schritt:
   Ausführung des Testfall Knotens 2.

Auch der Aufräumen-Schritt erscheint im Protokoll:

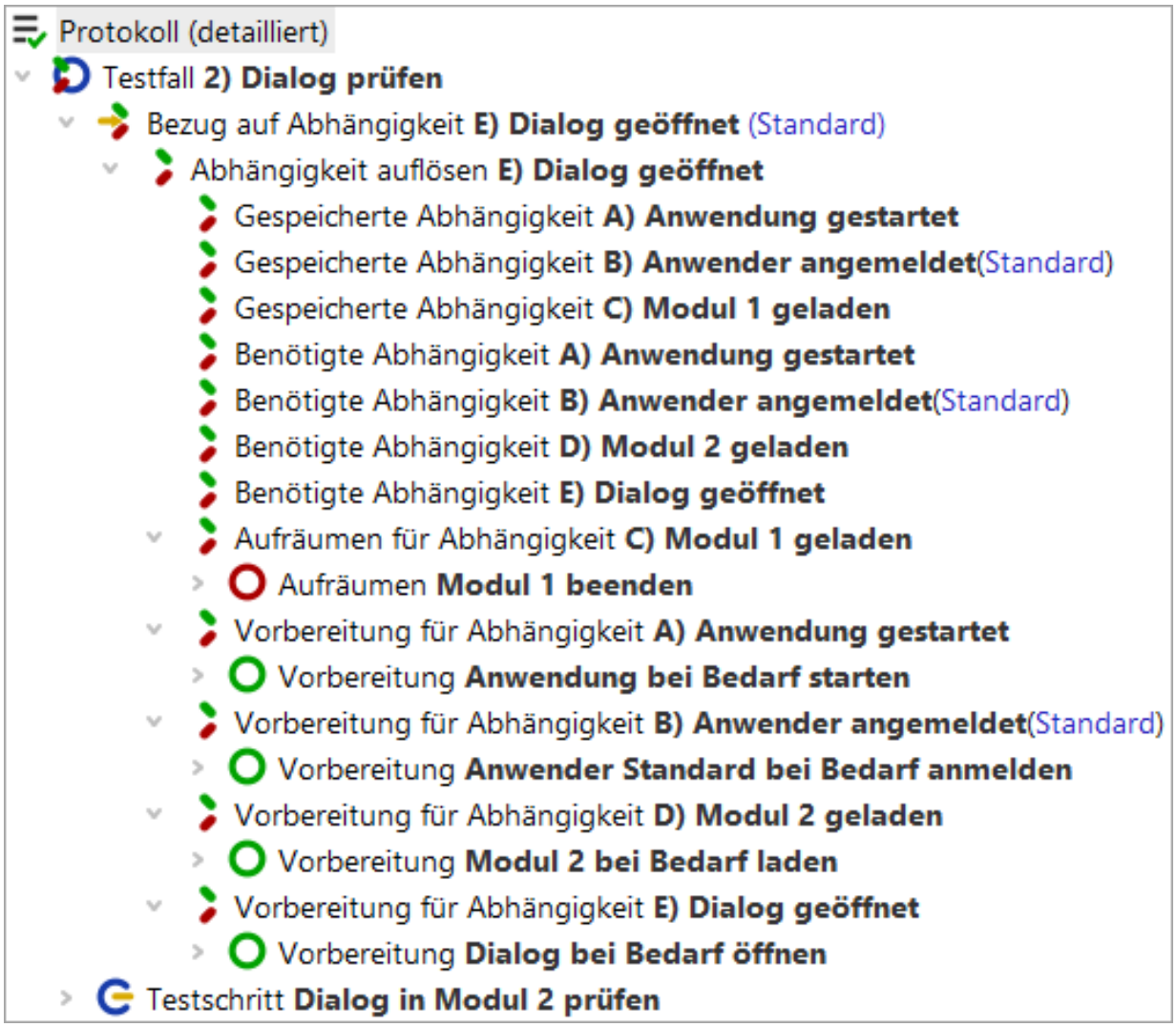


Abbildung 8.7: Stapel von Abhängigkeiten A-B-D-E

8.6.4 Charakteristische Variablen

Eine Abhängigkeit kann ihrerseits von den Werten gewisser Variablen abhängen. Diese Variablen werden als Charakteristische Variablen einer Abhängigkeit bezeichnet. Beim Abgleich der benötigten Abhängigkeit Knoten mit den bereits ausgeführten auf dem Abhängigkeitenstapel werden die Charakteristischen Variablen mit herangezogen. Zwei Abhängigkeit Knoten gelten nur dann als gleich, wenn auch die Werte aller Charakteristischen Variablen zwischen der vorherigen und der aktuellen Ausführung übereinstimmen. Somit ist es auch möglich, dass eine Abhängigkeit sich, direkt oder indirekt, mehrfach auf die selbe Abhängigkeit bezieht, aber mit unterschiedlichen Werten für die Charakteristischen

Variablen. In diesem Fall taucht die bezogene Abhängigkeit mehrfach in der Liste der benötigten Abhängigkeiten und nach deren Ausführung auf dem Abhängigkeitenstapel auf, jeweils mit dem entsprechenden Wert der Charakteristischen Variablen.

Wird die Abhängigkeit abgebaut und ihr Aufräumen Knoten ausgeführt, erhalten die Charakteristischen Variablen innerhalb des Aufräumen Knotens die gespeicherten Werte, also die, die sie zum Zeitpunkt der Ausführung des Vorbereitung Knotens hatten. Dadurch wird sicher gestellt, dass die Ausführung eines Testfall Knotens mit anderen Variablenwerten die Aufräumen Knoten beim Abbau des Abhängigkeitenstapels nicht aus der Bahn werfen kann. Denken Sie zum Beispiel an die häufig für den Namen des SUT Clients verwendete Variable ”client”. Wird zunächst ein Satz von Tests für ein SUT ausgeführt und benötigt der nächste Testfall ein ganz anderes SUT, wird eventuell die Variable ”client” umdefiniert. Der Aufräumen Knoten zum Beenden des alten SUT muss aber den alten Wert für ”client” erhalten, sonst könnte er den alten SUT Client gar nicht beenden. Darum kümmert sich QF-Test automatisch, sofern sich ”client” unter den Charakteristischen Variablen der Abhängigkeit befindet.

Beispiel: Testfall 3:

Der gleiche Dialog soll für den Administrator getestet werden.

1. Schritt:
   Die Analyse der einbezogenen Abhängigkeiten ergibt die gleiche Abhängigkeitenliste A-B-D-E (Anwendung gestartet, Anwender angemeldet, Modul 2 geladen, Dialog geöffnet) wie für Testfall 2, wobei die Charakteristische Variablen in Abhängigkeit B den Wert ’Administrator’ hat.
2. Schritt:
   Abgleich der Abhängigkeitenliste mit dem Abhängigkeitenstapel: Die benötigte Abhängigkeit B unterscheidet sich auf Grund der Charakteristischen Variablen ’anwendername’ mit dem Wert ’Administrator’ von der auf dem Abhängigkeitenstapel gespeicherten mit dem Wert ’Standard’. Dies bedeutet, dass der Abhängigkeitenstapel bis einschließlich Abhängigkeit B (Anwender abmelden) abgebaut wird, beginnend mit Aufräumen für Abhängigkeit E (Dialog schließen), dann Aufräumen für Abhängigkeit D (Modul 2 beenden), dann Aufräumen für Abhängigkeit B (Anwender abmelden). Die Variable ’anwendername’ erhält hierbei den über die Charakteristische Variablen gespeicherten Wert ’Standard’.)
3. Schritt:
   Ausführung der Vorbereitung Knoten, beginnend mit A (Es wird erkannt, dass die Anwendung bereits läuft und der Rest der Vorbereitung übersprungen.), dann B (Administrator anmelden), dann D (Modul 2 laden), dann E (Dialog öffnen)
4. Schritt:
   Ausführung des Testfall Knotens 3

Im Protokoll sieht man die den Wert der Charakteristischen Variablen hinter der jeweiligen Abhängigkeit:

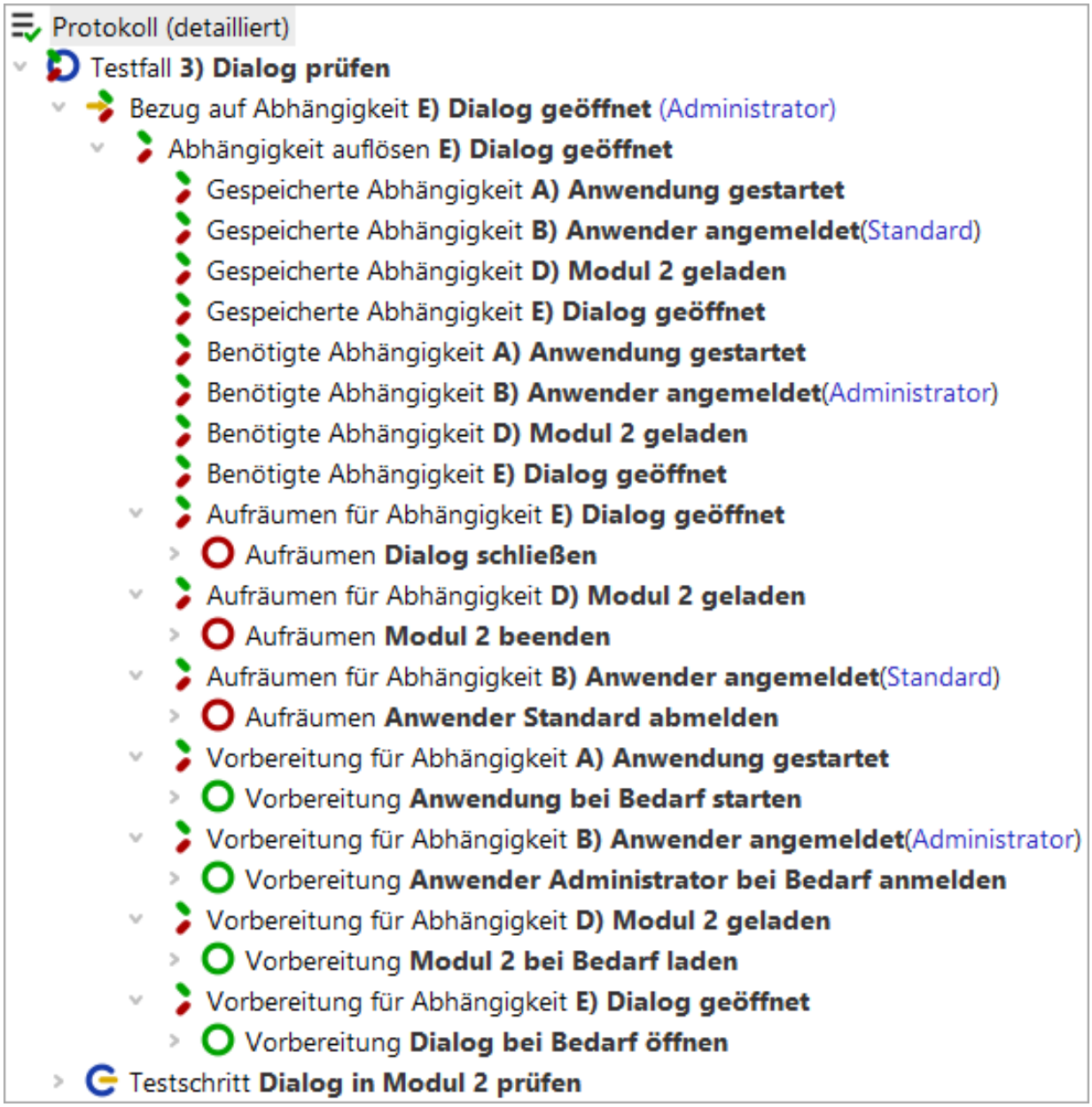


Abbildung 8.8: Charakteristische Variablen

Weitere typische Beispiele für Charakteristischen Variablen sind die JDK-Version, wenn das SUT mit verschiedenen JDK-Versionen getestet werden soll, oder der Browsername im Falle einer Web-Anwendung. Diese würden in unserem Beispiel als Charakteristische Variablen für Abhängigkeit A (Anwendung gestartet) definiert.

8.6.5 Aufräumen erzwingen

In manchen Anwendungsfällen kann es nötig sein, dass der Aufräumen Knoten einer Abhängigkeit nach jedem Testfall ausgeführt wird. In diesem Fall kann das Attribut Aufräumen erzwingen gesetzt werden.

Falls bei einem der Abhängigkeit Knoten auf dem Stapel das Attribut Aufräumen erzwingen gesetzt ist, werden von diesem und eventuell nachgeordneten Abhängigkeit Knoten die Aufräumen Knoten ausgeführt.

Beispiel:

In diesem Beispiel verlangt die Testlogik, dass das Anwendungsmodul 2 nach Testausführung immer beendet werden muss. Somit wird im Abhängigkeit Knoten D das Attribut Aufräumen erzwingen gesetzt.

Nun würden in Schritt 2 unserer Beispiele immer die Aufräumen Knoten der Abhängigkeit E (Dialog schließen) und der Abhängigkeit D (Modul 2 beenden) ausgeführt.

8.6.6 Abhängigkeiten abbauen

QF-Test räumt Abhängigkeiten je nach Bedarf der Testfälle auf.

Sie können den Stapel der Abhängigkeiten auf zwei Arten explizit leeren:

- Über den Menüeintrag Wiedergabe→Abhängigkeiten auflösen wird der Stapel "sauber" abgebaut indem alle Aufräumen Knoten in umgekehrter Reihenfolge der Abhängigkeitsvorbereitungen ausgeführt werden.
- Mittels Wiedergabe→Abhängigkeiten zurücksetzen wird der Stapel dagegen einfach gelöscht, ohne dass irgendein Knoten ausgeführt wird.

Bei einem Testfall, der selbst keine Abhängigkeiten nutzt, bleibt der Stapel der Abhängigkeiten unberührt, d.h. es werden keine Aufräumen Knoten ausgeführt. Erst beim nächsten Testfall mit Abhängigkeiten wird der Abhängigkeitenstapel wieder berücksichtigt.

8.6.7 Eskalation von Fehlern

Eine weitere hervorragende Eigenschaft von Abhängigkeiten ist die Möglichkeit, Fehler ohne weiteren Aufwand zu eskalieren. Betrachten wir wieder das Beispiel aus dem vorhergehenden Abschnitt nachdem der erste Abhängigkeitenstapel A-B-C (Anwendung gestartet, Anwender angemeldet, Modul 1 geladen) aufgebaut und alle zugehörigen

Vorbereitung Knoten ausgeführt wurden. Was passiert, wenn bei der Ausführung des eigentlichen Testfall Knotens das SUT auf einen wirklich schweren Fehler stößt wie z.B. ein Deadlock und nicht mehr auf Benutzereingaben reagiert?

Schlägt beim Abbau des Stapels von Abhängigkeiten die Ausführung eines Aufräumen Knotens fehl, baut QF-Test die nächstzugrunde liegende Abhängigkeit ab, bei erneuten Problemen eine weitere und so fort. Analog dazu führt beim Aufbau des Stapels ein Fehler in einem Vorbereitung Knoten dazu, dass zunächst eine weitere Abhängigkeit abgebaut wird und dann noch einmal die Vorbereitung Knoten ausgeführt werden.

Beispiel:

Im obigen Beispiel Testfall 1 geht das SUT zum Beispiel in ein Deadlock und reagiert nicht mehr auf Eingaben. Zunächst wird eine Exception geworfen, die Ausführung von Testfall 1 abgebrochen und mit Testfall 2 fortgefahren.

1. Schritt:
   Analyse der benötigten Abhängigkeiten und Erstellen der Abhängigkeitenliste A-B-D-E (Anwendung gestartet, Anwender angemeldet, Modul 2 geladen, Dialog geöffnet)
2. Schritt:
   Der Abgleich der benötigten Abhängigkeiten mit dem von Testfall 1 aufgebauten Abhängigkeitenstapel A-B-C (Anwendung gestartet, Anwender angemeldet, Modul 1 geladen) bewirkt, dass der Aufräumen Knoten der Abhängigkeit C (Modul 1 beenden) ausgeführt wird.
   Dabei kommt es erneut zur Exception. Nun führt QF-Test den Aufräumen Knoten der zu Grunde liegenden Abhängigkeit B (Anwender abmelden) aus. Dies scheitert erneut, so dass wiederum die zu Grunde liegende Abhängigkeit A aufgeräumt wird. Diese schließt erfolgreich die Anwendung.
3. Schritt:
   Ausführung der Vorbereitung Knoten, beginnend mit A (Anwendung starten), dann B (Anwender anmelden), dann D (Modul 1 laden) und schließlich E (Dialog öffnen).
4. Schritt:
   Testfall 2 kann trotz des Deadlocks in Testfall 1 ausgeführt werden.

Dies lässt sich wieder genau im Protokoll nachverfolgen:

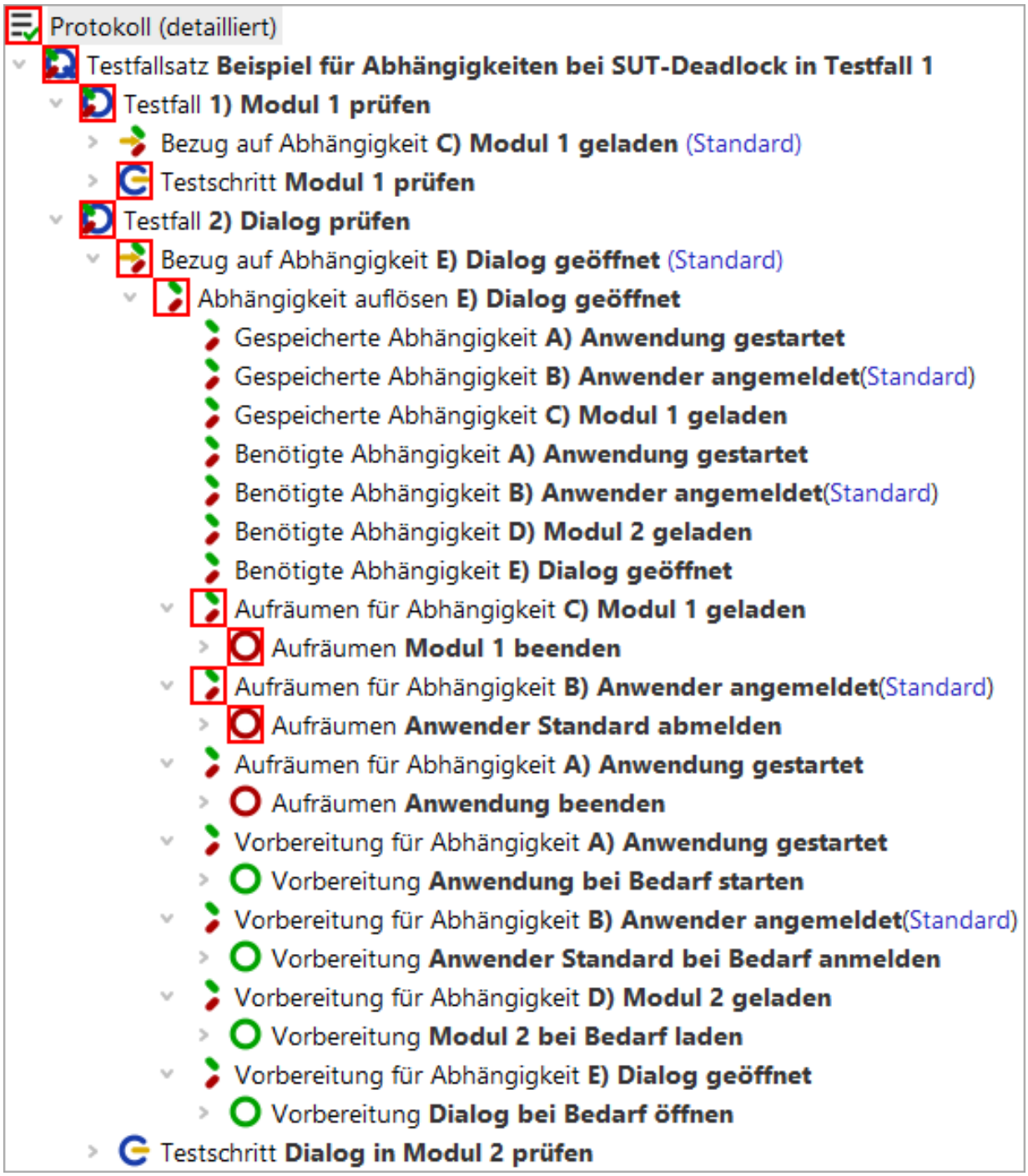


Abbildung 8.9: Fehlereskalation in Abhängigkeiten

Damit die Fehlereskalation zuverlässig funktioniert, ist es sehr wichtig, Aufräumen Knoten so zu implementieren, dass entweder der gewünschte Zustand erreicht oder eine

Exception geworfen wird. Letzteres führt nur zum Erfolg, wenn in einer der grundlegenderen Abhängigkeiten ein umfassender Aufräumen Knoten vorhanden ist, der das SUT in einen definierten Zustand bringen oder beenden kann. Dabei ist es wichtig, dass zum Beispiel der Aufräumen Knoten, der das SUT beenden soll, nicht nur versucht, das SUT etwa über das Datei->Beenden Menü sauber herunterzufahren, sondern auch abprüft, ob die Aktion erfolgt reich war und gegebenenfalls drastischere Maßnahmen durchführt wie zum Beispiel den Prozess des SUT "abzuschießen". Ein solcher Aufräumen Knoten sollte also einen Try/Catch Block verwenden und zusätzlich in einem Finally Knoten sicherstellen, dass das SUT auf jeden Fall beendet ist.

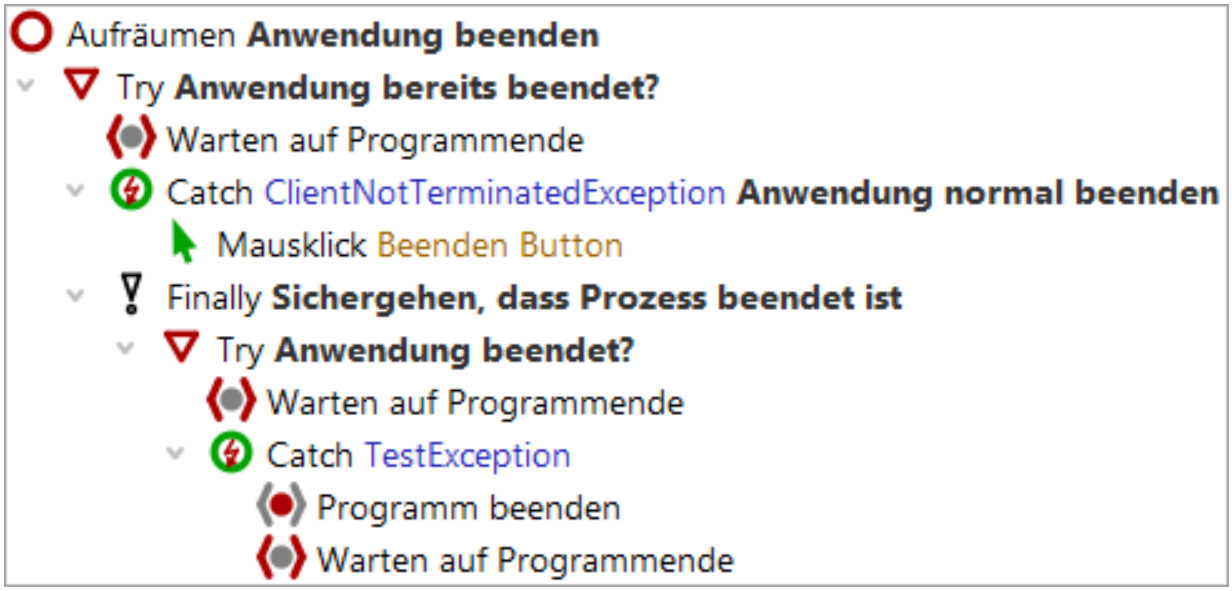


Abbildung 8.10: Typischer Aufräumen Knoten

Bei sauber implementierter Fehlerbehandlung werden sich Testfälle selbst bei schweren Fehlern kaum gegenseitig beeinflussen. Dies schützt vor dem Verlust aller Testergebnisse eines lang dauernden nächtlichen Testlaufs nur aufgrund eines einzelnen Fehlers zu einem frühen Zeitpunkt.

8.6.8 Fehlerbehandlung und Reaktion auf unerwartetes Verhalten

Neben der automatischen Eskalation von Fehlern unterstützt ein Abhängigkeit Knoten auch die explizite Behandlung von Fehlern oder Exceptions, die bei der Ausführung eines Testfall Knotens auftreten. Hierzu können Catch Knoten am Ende des Abhängigkeit Knotens eingefügt werden. Diese fangen im Testfall auftretende Exceptions und können gezielt darauf reagieren. So könnte zum Beispiel eine `DeadlockTimeoutException`(960) mit umgehender Terminierung des SUT Prozesses

quittiert werden. Eine derart gefangene Exception wird in Protokoll und Report weiterhin als Exception aufgeführt.

Des Weiteren kann ein Fehlerbehandlung Knoten in der Abhängigkeit zwischen dem Aufräumen Knoten und dem ersten Catch Knoten eingefügt werden. Dieser wird ausgeführt, wenn der Testfall mit dem Ergebnis "Fehler" beendet wird. Im Fall einer Exception wird der Fehlerbehandlung Knoten nicht ausgeführt, da das in der Regel eher noch mehr Probleme verursachen würde und sogar mit der Behandlung von Exceptions kollidieren könnte. Um im Fall von Fehlern und Exceptions die selben Konsequenzen zu ziehen, können Sie dafür eine Prozedur implementieren und diese aus Fehlerbehandlung und Catch Knoten heraus aufrufen. Fehlerbehandlung Knoten sind nützlich, um Informationen zu ermitteln und zu speichern, die QF-Test nicht bereits automatisch zur Verfügung stellt.

So könnten Sie beispielsweise eine Kopie von temporären Dateien oder Protokollen erstellen, die von Ihrem SUT angelegt werden und eventuell Aufschlüsse über die Fehlerursache liefern könnten. Es wird jeweils nur der oberste Fehlerbehandlung Knoten auf dem Stapel von Abhängigkeiten ausgeführt. Wenn also bei einem Stapel der Form A,B,C,D] sowohl A als auch C einen Fehlerbehandlung Knoten enthalten, wird nur der Knoten von C ausgeführt. Andernfalls wäre es schwierig, in der spezialisierten Abhängigkeit C die Fehlerbehandlung der allgemeineren Abhängigkeit A zu modifizieren. Um die Fehlerbehandlung von A in C wiederzuverwenden, implementieren Sie diese in einer Prozedur.

8.6.9 Namensräume für Abhängigkeiten

Hinweis

Dieser Absatz ist für Sie nur interessant, wenn Sie mehrere Applikationen haben, die Sie gleichzeitig testen wollen und wenn der Abhängigkeit Knoten eines Testfalls keine Auswirkung auf den bestehenden Abhängigkeitenstapel haben soll.

Ein typischer Anwendungsfall ist der Test ganzer Prozessketten, die mehrere Applikationen umfassen.

Als Beispiel soll hier folgende Situation dienen: Der Außendienst erfasst die Angebotsdaten über eine Web-Applikation und schickt diese an ein Datenbanksystem in der Zentrale, wo die Angebote vervollständigt, gedruckt und versandt werden. Eine Kopie der versandten Aufträge wird in einem separaten Dokumentenmanagementsystem (DMS) abgespeichert.

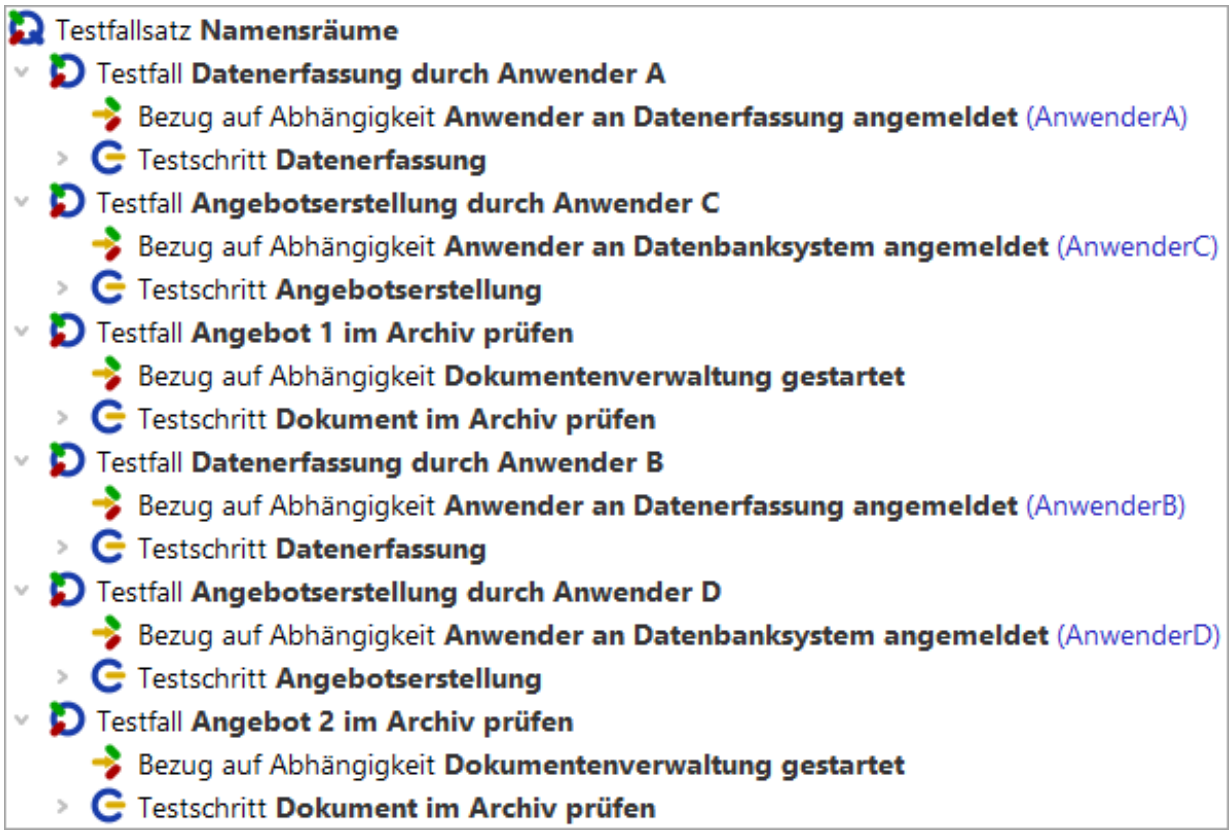


Abbildung 8.11: Beispiel Testfallsatz für Namensräume

In obigem Beispiel erfassen zwei unterschiedliche Mitarbeiter (AnwenderA und AnwenderB) die Daten für das Angebot, zwei weitere (AnwenderC und AnwenderD) erstellen die Angebote und die Archivierung wird im Dokumentenmanagementsystem geprüft. Damit sich die Anhängigkeiten nicht gegenseitig beeinflussen, wird im Bezug auf Abhängigkeit Knoten jeweils ein passender Name im Feld Namensraum für Abhängigkeiten(636) eingetragen.

Wenn man den Testfallsatz ausführt, sieht man im Protokoll, dass für den ersten Testfall ein Abhängigkeitenstapel im Namensraum ”Datenerfassung” angelegt wird:

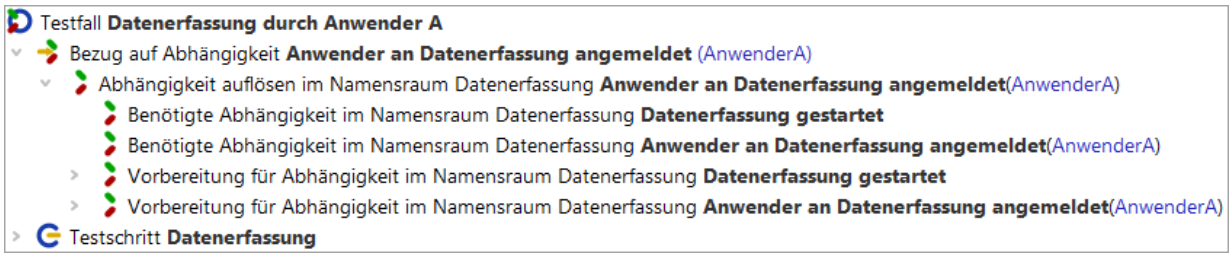


Abbildung 8.12: Abhängigkeitenbehandlung für Testfall Datenerfassung durch Anwender A

Für den zweiten Testfall wird ein Abhängigkeitenstapel im Namensraum ”Datenbanksystem” angelegt. Der Abhängigkeitenstapel im Namensraum ”Datenerfassung” wird nicht betrachtet. Es wird also das Datenbanksystem gestartet. Die Anwendung zur Datenerfassung bleibt unverändert.

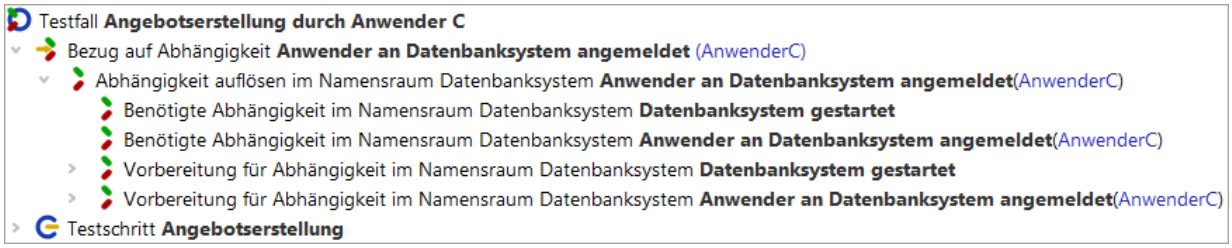


Abbildung 8.13: Abhängigkeitenbehandlung für Testfall Angebotserstellung durch Anwender C

Für den dritten Testfall wird ein Abhängigkeitenstapel im Namensraum ”Dokumentenverwaltung” angelegt. Die Abhängigkeitenstapel in den Namensräumen ”Datenerfassung” und ”Datenbanksystem” werden nicht betrachtet. Es wird also das Dokumentenmanagementsystem gestartet. Die beiden anderen Anwendungen bleiben unverändert.

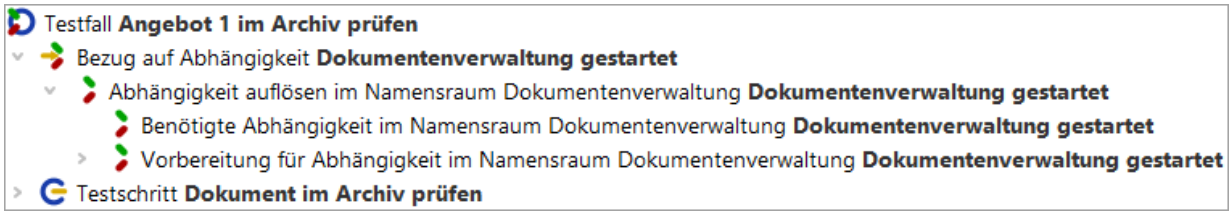


Abbildung 8.14: Abhängigkeitenbehandlung für Testfall Angebot 1 im Archiv prüfen

Im vierten Testfall werden die benötigten Abhängigkeiten mit dem vorhandenen Stapel im Namensraum ”Datenerfassung” aus dem ersten Testfall abgeglichen. Die Abhängig-

keitenstapel in den beiden anderen Namensräumen werden nicht betrachtet. Es wird also Anwender A abgemeldet und Anwender B an der Datenerfassung neu angemeldet. Die beiden anderen Anwendungen bleiben unverändert.

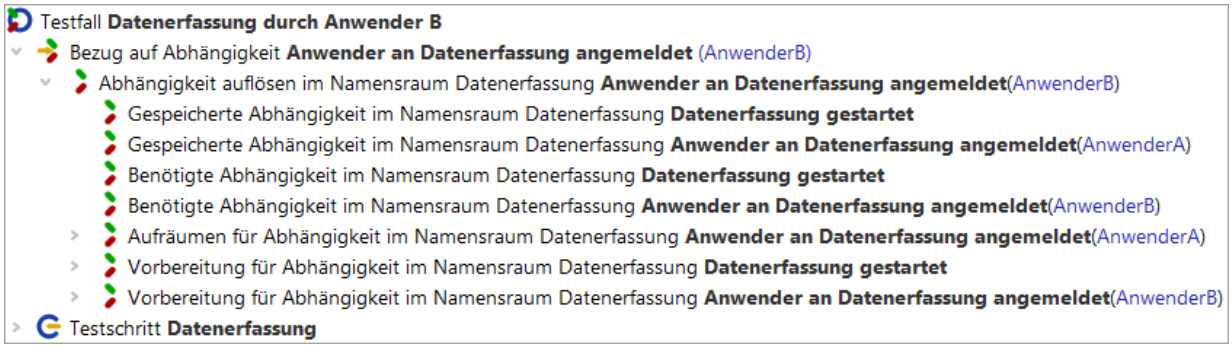


Abbildung 8.15: Abhängigkeitenbehandlung für Testfall Datenerfassung durch Anwender B

Im fünften Testfall werden die benötigten Abhängigkeiten mit dem vorhandenen Stapel im Namensraum ”Datenbanksystem” aus dem zweiten Testfall abgeglichen. Die Abhängigkeitenstapel in den beiden anderen Namensräumen werden nicht betrachtet. Es wird also Anwender C abgemeldet und Anwender D im Datenbanksystem neu angemeldet. Die beiden anderen Anwendungen bleiben wiederum unverändert.

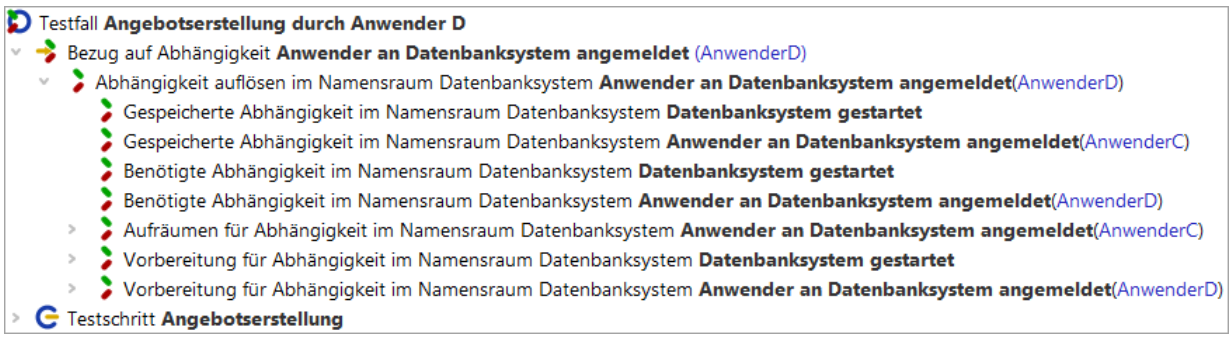


Abbildung 8.16: Abhängigkeitenbehandlung für Testfall Angebotserstellung durch Anwender D

Im letzten Testfall wird die benötigte Abhängigkeit mit dem vorhandenen Stapel im Namensraum ”Dokumentenverwaltung” aus dem dritten Testfall abgeglichen. Die Abhängigkeitenstapel in den beiden anderen Namensräumen werden nicht betrachtet. Es ergibt sich kein Handlungsbedarf bezüglich Aufräumarbeiten. Die beiden anderen Anwendungen bleiben wiederum unverändert.

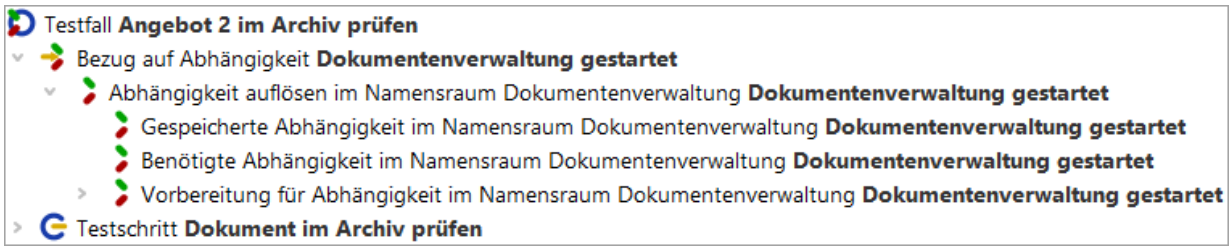


Abbildung 8.17: Abhängigkeitenbehandlung für Testfall Angebot 2 im Archiv prüfen

8.7 Dokumentieren von Testsuiten

Wie jede programmierähnliche Tätigkeit benötigt Testautomatisierung eine gute Dokumentation um langfristig erfolgreich sein zu können. Andernfalls besteht die Gefahr, den Überblick zu verlieren und Dinge unnötigerweise mehrfach zu implementieren oder Tests zu übersehen, die automatisiert werden sollten. Eine gute Dokumentation ist von unschätzbarem Wert, wenn Sie sich auf der Suche nach der Ursache für einen fehlgeschlagenen Test durch ein Protokoll arbeiten. Außerdem trägt sie wesentlich zur Lesbarkeit von Reports bei.

Eine gute Möglichkeit, die Tests lesbar zu gestalten und gleichzeitig zu dokumentieren sind die Sequenz(618) und Testschritt(621) Knoten, mit deren Hilfe Sie die aufgenommenen Knoten zusammenfassen und beschriften können.

Es steht auch ein Kommentar(851) Knoten zur Verfügung um Erläuterungen einzufügen.

Wenn Sie eine Dokumentation erstellen wollen, die auch außerhalb von QF-Test verfügbar sein soll, so können Sie basierend auf den Bemerkung(613) Attributen von Testfallsatz(606), Testfall(599), Package(680) und Prozedur(672) Knoten einen Satz von umfassenden HTML Dokumenten erstellen, welche die benötigten Informationen schnell auffindbar machen. Die verschiedenen Arten von Dokumenten und die Mittel zu ihrer Erstellung werden ausführlich in Kapitel 24(330) beschrieben.

Kapitel 9

Projekte

3.5+

Projekte schaffen einen besseren Überblick, vereinfachen die Navigation zwischen Testsuiten und bieten einen erweiterten Rahmen für Suchen und Ersetzen. Außerdem kümmert sich QF-Test automatisch um die durch Include-Beziehungen oder absolute Referenzen entstehenden Abhängigkeiten zwischen Testsuiten im gleichen Projekt (vgl. Abschnitt 26.1(359)). Viele weitere Funktionen sind bereits implementiert oder in Entwicklung.

Technisch gesehen ist ein Projekt in QF-Test eine Sammlung von Testsuiten mit einem gemeinsamen Wurzelverzeichnis. Es gibt eine 1:1 Beziehung zwischen dem Projekt und seinem Verzeichnis und der Name des Verzeichnisses wird automatisch zum Namen des Projekts.

Ein neues Projekt erstellen Sie über das Menü Datei→Neues Projekt... . Wählen Sie anschließend das Verzeichnis aus. QF-Test erstellt darin eine Datei namens `qftest.qpj`, die das Verzeichnis als Projekt kennzeichnet. Alle Testsuiten unterhalb dieses Verzeichnisses gehören automatisch zum Projekt, mit Ausnahme derer, die durch die Option Auszuschließende Projektdateien und -verzeichnisse(487) ausgeschlossen werden. Abschnitt 41.1.1(487) führt einige Optionen für Projekte auf, darunter auch die Ausschlussliste.

Ein Unterprojekt ist ein Unterverzeichnis eines Projekts, welches selbst ein Projekt ist. Testsuiten innerhalb eines Unterprojekts gehören auch zu allen Projekten darüber. *Das* Projekt einer Testsuite ist das innerste Projekt, zu dem die Suite gehört. Die Automatische Auflösung von Abhängigkeiten umfasst immer das gesamte äußerste Projekt einer Testsuite inklusive aller Unterprojekte.

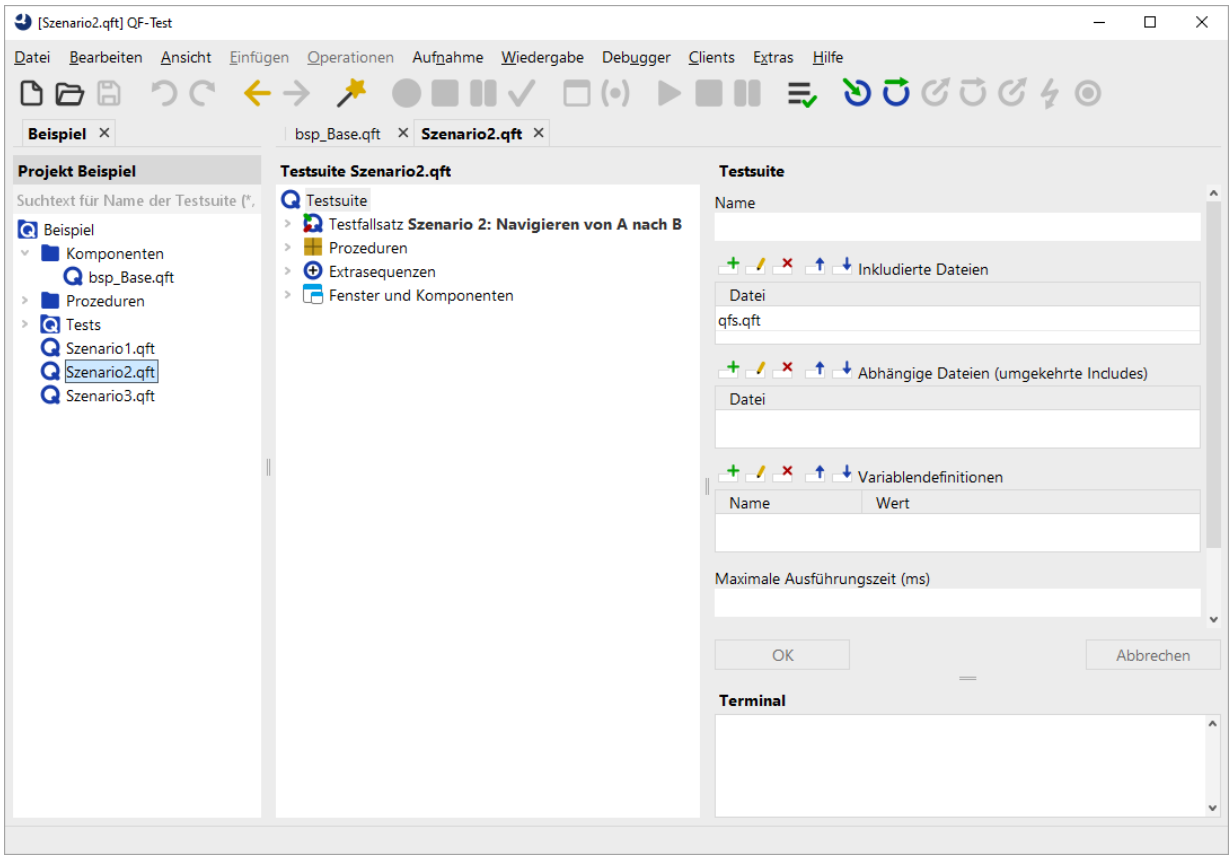


Abbildung 9.1: Die Projektansicht

Die Projektansicht mit ein oder mehreren Projekten wird über das Menü Ansicht→Projekte anzeigen ein oder ausgeschaltet. Der Projektbaum zeigt die Verzeichnishierarchie eines Projekts mit den Testsuiten, beginnend vom Wurzelverzeichnis. Die Hierarchie kann mit Hilfe des Filters oben am Baum eingeschränkt werden, der sich auf die Namen von Testsuiten bezieht. Ein Doppelklick auf eine Testsuite öffnet diese, ebenso wie ein Druck auf die Eingabe Taste. Es können auch mehrere Dateien auf einmal selektiert, oder alle Dateien unterhalb eines Verzeichnisses geladen werden.

Die Hierarchie wird regelmäßig komplett aktualisiert, den Zeitabstand hierfür definiert die Option Zeitabstand für automatisches Auffrischen von Projekten (s)[487]. Sie können ein Verzeichnis inklusive seiner gesamten Hierarchie jederzeit manuell über das Kontextmenü oder durch Drücken von F5 aktualisieren. Um das Verzeichnis komplett neu einzulesen ohne sich auf die Änderungszeiten der Verzeichnisse zu verlassen, was bei großen Projekten deutlich länger dauern kann, drücken Sie stattdessen Umschalt-F5.

Mit der F6 Taste wechseln Sie den Fokus zwischen Testsuite und Projektansicht hin und her. Um schnell von der aktuellen Testsuite zu deren Knoten im Projektbaum zu ge-

langen, drücken Sie Umschalt-F6. Falls nötig werden Projektansicht und Projektbaum für diese Aktion automatisch geöffnet.

Kapitel 10

Standardbibliothek

Die Standardbibliothek `qfs.qft`, eine Testsuite mit vielen hilfreichen Prozeduren für unterschiedliche Aufgaben, ist Teil der QF-Test Distribution.

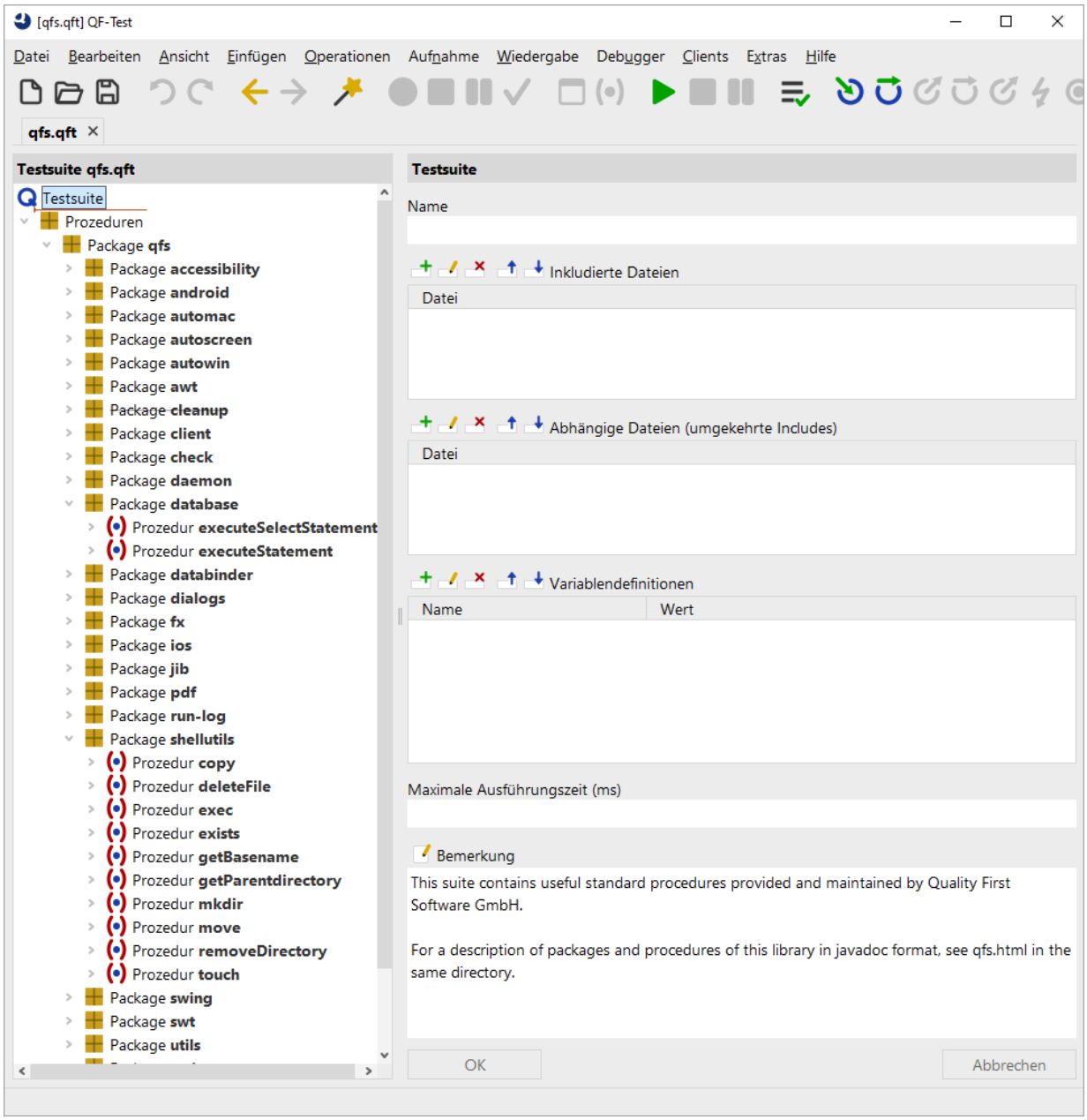


Abbildung 10.1: Standardbibliothek `qfs.qft`

Unter anderem gibt es Prozeduren zum generischen Ansteuern und Prüfen von Komponenten (AWT, Swing, JavaFX, SWT, Web), für den Zugriff auf Dateisysteme und Datenbanken, das Schreiben von Text oder Bildschirmabbildern in Protokoll und Report sowie Aufräummechanismen.

Die vollständige Beschreibung aller Packages und Prozeduren, inklusive Parameter und Rückgabewerte, finden Sie in der HTML Dokumentation, die auch über das Hilfe Menü

in QF-Test erreichbar ist. Zusätzlich ist die jeweils neueste Version online verfügbar.

`qfs.qft` wird standardmäßig in jeder neu erzeugten Testsuite inkludiert. Da das Verzeichnis der Suite immer im Bibliothekspfad[503] enthalten ist, reicht `qfs.qft` ohne weitere Pfadangabe in den Inkludierte Dateien[596] des Testsuite Knotens aus.

Hinweis Alle Prozeduren, die sich auf ein SUT beziehen, nutzen die generische Variable `$(client)` als impliziten Parameter. Stellen Sie sicher, dass diese Variable korrekt gesetzt wird, entweder global, lokal oder als expliziter Parameter im Prozeduraufruf.

Kapitel 11

Skripting

Video

Das Video

'Skripting in QF-Test' (Grundlagen)
https://www.qftest.com/de/yt/skripting-basics-45.html

behandelt die Grundlagen des Skriptens.

Das Video

'Skripting in QF-Test' (Fortgeschritten)
https://www.qftest.com/de/yt/skripting-advanced-47.html

zeigt weitere Möglichkeiten des Skriptens.

Es ist einer der großen Vorteile von QF-Test, dass komplexe Tests erstellt werden können, ohne eine einzige Zeile Code zu schreiben. Allerdings gibt es Dinge, die sich mit einer GUI alleine nicht bewerkstelligen lassen. Für ein Programm, das Daten in eine Datenbank schreibt, könnte es z.B. sinnvoll sein, zu überprüfen, ob die Daten korrekt geschrieben wurden. Oder man könnte Testdaten aus einer Datenbank oder einer Datei lesen und mit diesen einen Test ausführen. All das und mehr wird mithilfe der mächtigen Skriptsprachen Jython, Groovy und JavaScript ermöglicht.

4.2+

Jython ist von Anfang an dabei, Groovy seit QF-Test Version 3. Ab Version 4.2 kann man auch JavaScript als Skriptsprache verwenden. Es ist eine Frage des Geschmacks, welcher dieser Sprachen man den Vorzug gibt. Wer jedoch bereits mit Java vertraut ist, wird sich wahrscheinlich eher mit Groovy denn mit Jython anfreunden. Web-Entwickler werden vermutlich JavaScript verwenden.

In diesem Kapitel werden zunächst die Grundlagen der Skriptintegration und die in allen Skriptsprachen zur Verfügung stehenden Module beschrieben. Auf die Besonderheiten der Sprachen Groovy-Skripting(208), Jython-Skripting(199) und JavaScript-Skripting(211) wird in den jeweiligen Abschnitten eingegangen.

3.0+

Die Skriptsprache eines Knotens wird mit dem Attribut Skriptsprache(719) eines

Server-Skript(717)- oder SUT-Skript(720)-Knotens festgelegt. Somit können alle drei Sprachen innerhalb einer Testsuite parallel verwendet werden. Welche Sprache als Standard verwendet werden soll, kann über die Optionen Voreingestellte Sprache für Skript-Knoten(485) und Voreingestellte Sprache für Bedingungen(485) eingestellt werden.

11.1 Allgemeines

Beim Skripting ist die Herangehensweise von QF-Test genau umgekehrt zu der anderer GUI Testprogramme. Anstatt den gesamten Test durch ein Skript zu steuern, bettet QF-Test kleine Skripte in die Testsuite ein. Dies geschieht mithilfe der Knoten Server-Skript(717) und SUT-Skript(720).

Beiden Knoten gemeinsam ist das Attribut Skript(718) für den eigentlichen Programmcode.

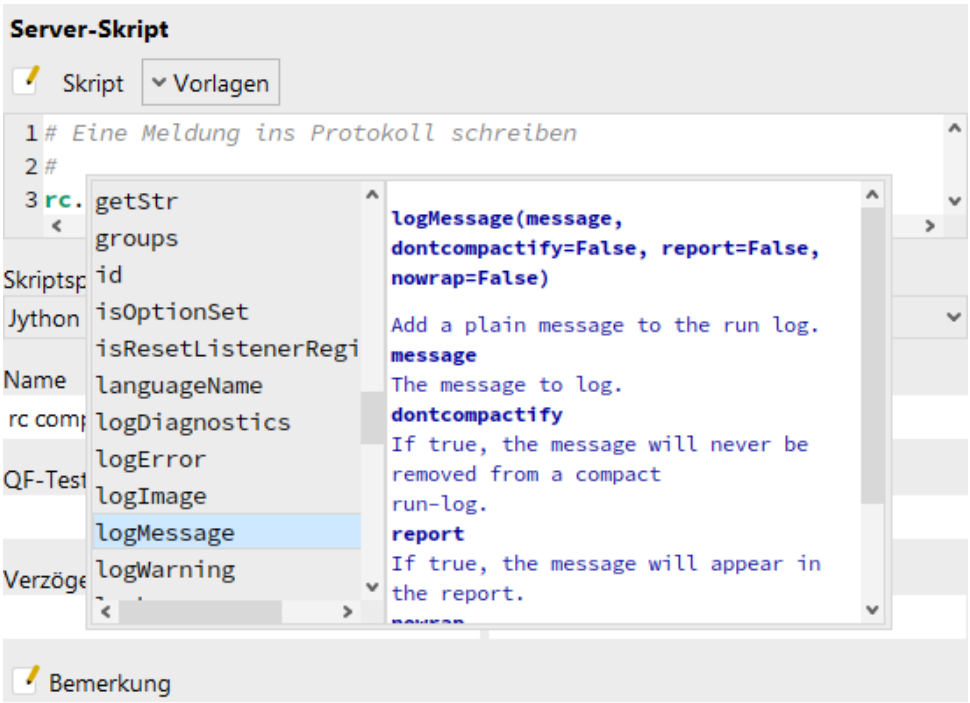


Abbildung 11.1: Detailansicht eines Server-Skript-Knotens mit Hilfefenster für `rc`-Methoden

3.0+ Der in QF-Test integrierte Skripteditor verfügt über ein paar nützliche Eigenschaften, die das Eingeben des Codes erleichtern. Reservierte Schlüsselwörter, eingebaute Funktionen, Standard-Objekttypen, Literale und Kommentare werden farblich hervorgehoben.

Innerhalb von Blöcken werden Codezeilen automatisch eingerückt und am Blockende wieder ausgerückt. Mit Hilfe von TAB können auch mehrere markierte Zeilen von Hand ein- oder ausgerückt (Shift-TAB) werden.

Das vielleicht - zumindest für den QF-Test Neuling - wichtigste Feature des integrierten Editors ist jedoch die Eingabehilfe für viele eingebaute Methoden. Gibt man beispielsweise `rc.` ein (und ggf. zusätzlich einen oder mehrere Anfangsbuchstaben eines Methodennamens) und drückt dann Ctrl-Leertaste, so erscheint ein Popup-Fenster mit den passenden *Runcontext* Methoden und ihrer Beschreibung (vgl. Kapitel 50(1028)). Nach Auswahl einer Methode und anschließender Bestätigung mit Eingabe wird die gewählte Methode in den Skriptcode eingefügt. Drückt man Ctrl-Leertaste nach einem Leerzeichen, wird eine Liste aller Objekte angezeigt, für die Hilfe zur Verfügung steht.

Server-Skripte sind für Dinge wie das Berechnen von Variablenwerten oder das Einlesen und Parsen von Testdaten nützlich. SUT-Skripte öffnen dagegen den unbeschränkten Zugang zu den Komponenten des SUT und zu allen anderen Java-Schnittstellen, die das SUT bietet. Ein SUT-Skript könnte z.B. zum Auslesen oder Überprüfen von Werten im SUT verwendet werden, auf die QF-Test keinen Zugriff hat. Im SUT-Skript-Knoten muss das Attribut Client(721) auf den Namen des SUT Clients gesetzt sein, in dem es ausgeführt werden soll.

Server-Skripte werden in jeder Skriptsprache jeweils in einem Interpreter ausgeführt, der in QF-Test selbst integriert ist, während SUT-Skripte in jeweils einem im SUT integrierten Interpreter laufen. Diese Interpreter sind voneinander unabhängig und haben keine gemeinsamen Zustände. QF-Test nutzt die RMI Verbindung zum SUT für eine nahtlose Integration der SUT-Skripte in die Testausführung.

Über die Menüeinträge Extras→Jython-Konsole, Extras→Groovy-Konsole etc. können Sie ein Fenster mit einer interaktiven Kommandozeile für die in QF-Test eingebetteten Interpreter öffnen. Darin können Sie mit der jeweiligen Skriptsprache experimentieren, um ein Gefühl für die Sprache zu entwickeln, aber auch komplexe Dinge ausprobieren wie z.B. das Herstellen der Verbindung zu einer Datenbank. Mittels Strg-Hoch und Strg-Runter können Sie frühere Eingaben wieder verwenden. Außerdem können Sie beliebige Zeilen in der Konsole bearbeiten oder eine Region markieren und mittels Return an den Interpreter schicken. Dabei filtert QF-Test die vom Interpreter stammenden '»>' und '...' Markierungen heraus.

Entsprechende Konsolen gibt es auch für SUT-Clients. Diese Konsolen sind über das Clients-Menü zugänglich.

Hinweis Wenn Sie in einer SUT-Skripting-Konsole arbeiten, müssen Sie eines beachten: Die Kommandos werden vom Interpreter nicht im Event Dispatch Thread ausgeführt, im Gegensatz zu Kommandos, die in einem SUT-Skripten-Knoten ausgeführt werden. Das sagt Ihnen möglicherweise nichts und meistens stellt es auch kein Problem dar, aber wenn Sie auf Swing- oder SWT-Komponenten zugreifen oder deren Methoden aufrufen, besteht die Gefahr, dass die gesamte Applikation einfriert. Um das zu

verhindern stellt QF-Test die globale Funktion `runAWT` (bzw. `runSWT`, `runFX`, `runWeb`, `runWin`, `runAndroid` und `runIOS`) zur Verfügung, mit deren Hilfe Sie beliebigen Code im Dispatch Thread ausführen können. Um zum Beispiel die Anzahl der sichtbaren Knoten einer `JTree` Komponente namens `tree` zu ermitteln, verwenden Sie `runAWT("tree.getRowCount()")` (bzw. `runAWT { tree.getRowCount() }` in Groovy) um ganz sicherzugehen.

11.2 Skriptausdrücke

Manchmal ist es hilfreich, in Knotenattributen kleinere Berechnungen oder Textmanipulationen direkt auszuführen. In QF-Test ist dies an allen Stellen möglich, an denen auch QF-Test Variablen durch ihre Werte ersetzt werden. Hierzu steht eine besondere Syntax zur Verfügung, über die einzeilige Skriptausdrücke ausgewerten werden können:

- `$[`*`Jython-Ausdruck`*`]` wertet den angegebenen Ausdruck im Jython-Interpreter aus. Alternativ kann `${`*`jython`*`:`*`Jython-Ausdruck`*`}` verwendet werden. Es sind alle Ausdrücke zulässig, deren Syntax für die Jython Methode `eval` gültig ist.
- Um einen Groovy-Ausdruck zu verarbeiten, verwenden Sie `${`*`groovy`*`:`*`Groovy-Ausdruck`*`}`,
- Für einen JavaScript-Ausdruck `${`*`javascript`*`:`*`JavaScript-Ausdruck`*`}`.

Die Auswertung der Skriptausdrücke erfolgt nach den gleichen Regeln wie für die Skriptknoten, siehe Jython-Skripting(199), Groovy-Skripting(208) beziehungsweise JavaScript-Skripting(211).

Hinweis Auch der Zugriff auf QF-Test Variablen in `${`*`Skriptsprache`*`:`*`Ausdruck`*`}` beziehungsweise `$[...]`-Ausdrücken folgt denselben Regeln wie in den Skripten für die entsprechende Sprache. Vor der Ausführung des Codes werden QF-Test Variablen im Ausdruck ausgewertet, d.h. die standard QF-Test Syntax `$(...)` und `${...:...}` kann für numerische und Boolesche Werte verwendet werden. Auf Zeichenketten sollte mittels `rc.getStr` zugegriffen werden (vgl. Abschnitt 11.3.3(191)).

Beispiel: In einem Schleife(684)-Knoten ist der Ausdruck `$[$(IndexLetzteZeile) + 1]` in Anzahl Wiederholungen hilfreich, wenn zuvor die Variable `IndexLetzteZeile` über den Knoten Index auslesen(843) mit dem Index `&-1` für die letzte Zeile (zum Beispiel `#List:&-1`) gesetzt wurde.

Skriptausdrücke können auch gekapselte Objekte zurückgeben. Beispiel: Die Variable `myList` erhält den Wert `$[ ["Affe", "Biber", "Chincilla"] ]`. In einem Schleife(684)-Knoten kann man nun `$[len(rc.vars.myList)]` in Anzahl Wiederholungen angeben.

In den Bedingungsfeldern von If[693], Testfall[599] und Testfallsatz[606] ist diese spezielle Syntax nicht erforderlich. Hier können Skriptausdrücke direkt eingegeben werden.

11.3 Der Runcontext rc

Zur Ausführung von Server-Skripten und SUT-Skripten stellt QF-Test eine spezielle Umgebung zur Verfügung, zu der u.a. das *Runcontext* Objekt gehört, das den aktuellen Zustand der Ausführung eines Tests repräsentiert. Auf dieses Objekt kann über die Variable ”rc”, welche in allen Sprachen verfügbar ist, zugegriffen werden. Es bietet Schnittstellen (vollständig dokumentiert in Abschnitt 50.5[1030]) für den Zugriff auf QF-Test Variablen, zum Aufruf von QF-Test Prozeduren und um Meldungen in das Protokoll zu schreiben. Ein SUT-Skript kann mit seiner Hilfe außerdem auf die echten Java-Komponenten des GUI im SUT zugreifen.

Für Fälle, in denen kein Runcontext verfügbar ist, z.B. Resolver, TestRunListener, Code der in einem Hintergrund-Thread ausgeführt wird etc. bietet QF-Test ein Modul namens `qf` mit hilfreichen generischen Methoden zum Logging und für andere Zwecke an. Detaillierte Informationen hierzu finden Sie in Abschnitt 50.6[1059].

11.3.1 Meldungen ausgeben

Ein Einsatzgebiet des Runcontexts ist die Ausgabe beliebiger Meldungen im Protokoll, das QF-Test für jeden Testlauf erstellt. Diese Meldungen können auch als Warnungen oder Fehler markiert werden.

```
rc.logMessage("This is a plain message")
rc.logWarning("This is a warning")
rc.logError("This is an error")
```

Beispiel 11.1: Meldungen aus Skripten ausgeben

Wird mit kompakten Protokollen gearbeitet (vgl. die Option Kompakte Protokolle erstellen[590]), werden Knoten, die aller Wahrscheinlichkeit nach nicht für eine Fehleranalyse benötigt werden, eventuell aus dem Protokoll entfernt, um Speicher zu sparen. Dies betrifft nicht die Fehlermeldung (`rc.logError`). Hier wird immer die Meldung selbst und etwa 100 vorhergehende Knoten im Protokoll aufgehoben. Bei einer Warnung `rc.logWarning` wird auf jeden Fall die Warnung behalten, jedoch keine vorhergehenden Knoten. Normale Meldungen (`rc.logMessage`) werden gegebenenfalls entfernt. Wenn Sie eine normale Meldung zwingend im Protokoll behalten wollen, können Sie dies über den optionalen zweiten Parameter (`dontcompactify`) erreichen:

```
rc.logMessage("This message will not be removed", dontcompactify=true)
rc.logMessage("This message will not be removed", 1)
```

Beispiel 11.2: Meldungen, die nicht aus kompakten Protokollen entfernt werden

11.3.2 Checks durchführen

Die Ausgabe einer Meldung ist meist an eine Bedingung geknüpft. Außerdem ist es oft wünschenswert, im XML- oder HTML-Report ein Ergebnis analog zu einem Check-Knoten zu erhalten. Hierzu dienen die Methoden `rc.check` und `rc.checkEqual`:

```
x = 0
rc.check(x == 0, "Value of x is 0")
userlang = rc.getStr("system", "user.language")
rc.checkEqual(userlang, "en", "English locale required",
              rc.EXCEPTION)
```

Beispiel 11.3: Checks durchführen

Das optionale letzte Argument legt die Fehlerstufe fest. Hierbei können `rc.EXCEPTION`, `rc.ERROR`, `rc.OK` bzw. `rc.WARNING` verwendet werden.

11.3.3 Variablen

In QF-Test gibt es verschiedene Arten von Variablen. Es wird zunächst unterschieden zwischen QF-Test Variablen, siehe Kapitel 6(116), und Variablen der Skriptsprachen. Die Variablen der Skriptsprachen wiederum werden unterteilt in Server- und SUT-seitige Variablen des jeweiligen Interpreters. Die folgende Grafik verdeutlicht die Sichtbarkeit der jeweiligen Variablenarten:

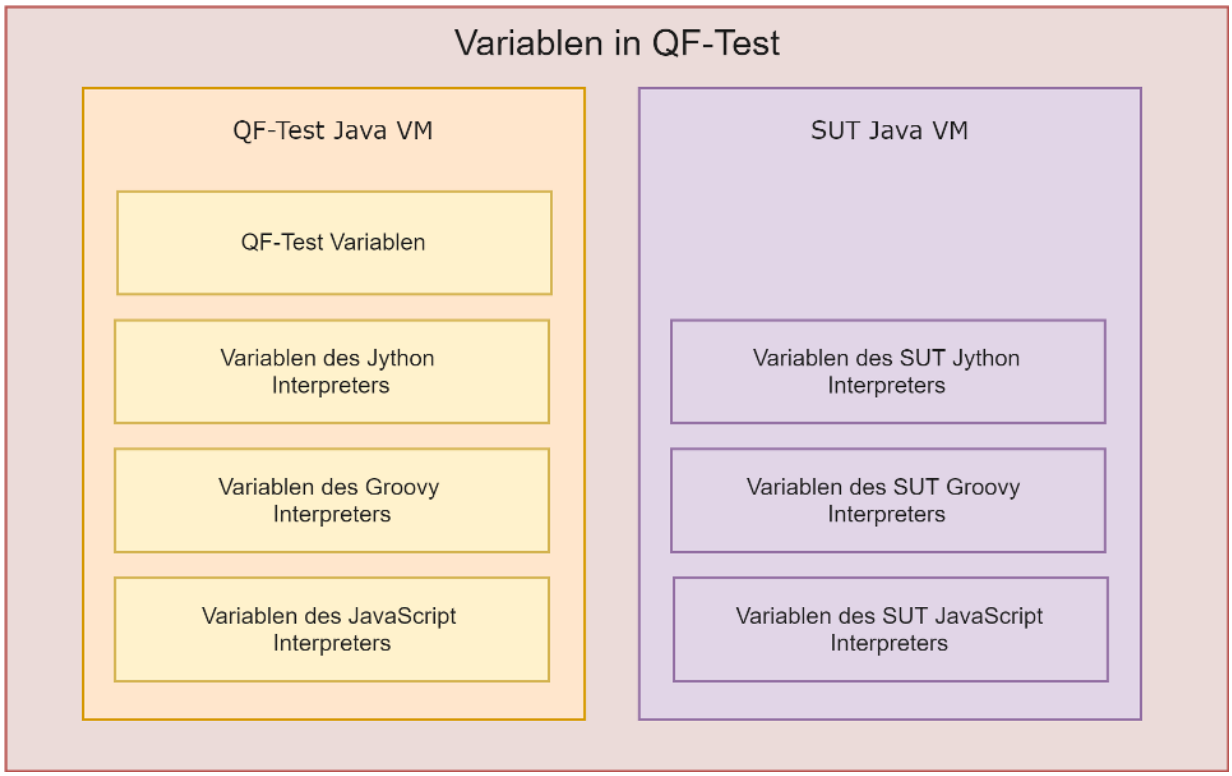


Abbildung 11.2: Übersicht über die verschiedenen Variablen in QF-Test

Um in den Skripten mit diesen unterschiedlichen Variablen zu arbeiten und dieses auszutauschen, stellt der Runcontext spezielle Methoden zur Verfügung. Diese Methoden werden in den nachfolgenden Abschnitten erläutert.

Zugriff auf Variablen

Auf Variablen von QF-Test in einem Skript zuzugreifen ist nicht weiter schwierig. Auf Textvariablen können Sie zum Beispiel mittels der Runcontext-Methode `getStr`, auf boolsche Werte mittels `getBool`, auf ganze Zahlen mit `getInt`, auf numerische Werte mit `getNum` und auf Datenobjekte mit `getObj` zugreifen (siehe Abschnitt 50.5(1030) für eine vollständige API-Beschreibung).

```
# access a simple variable
text = rc.getStr("someText")
# access a property or resource
version = rc.getStr("qftest", "version")
```

Beispiel 11.4: Zugriff auf Textvariablen mittels `rc.getStr`

Variablen setzen

Um die Ergebnisse eines Skripts für die weitere Ausführung eines Tests bekannt zu machen, können Werte in globalen oder lokalen QF-Test Variablen abgelegt werden. Der Effekt entspricht der Ausführung eines Variable setzen(871)-Knotens. Die entsprechenden Methoden im Runcontext sind `rc.setGlobal` und `rc.setLocal`.

```
# Test if the file /tmp/somefile exists
from java.io import File
rc.setGlobal("fileExists", File("/tmp/somefile").exists())
```

Beispiel 11.5: Verwendung von `rc.setGlobal`

Nach Ausführung des obigen Skripts wird `$(fileExists)` in einem Knoten von QF-Test zu `True` expandieren, wenn die Datei `/tmp/somefile` existiert und zu 'false', wenn sie nicht existiert.

Um eine Variable zu löschen, setzen Sie deren Wert auf `None` in Jython bzw. `null` in Groovy und JavaScript. Mittels `rc.clearGlobals()` aus einem Server-Skript können alle globalen Variablen gelöscht werden.

Globale Skript-Variablen

Manchmal ist es hilfreich, eine Skript-Variable in verschiedenen Skriptknoten zur Verfügung zu haben. Falls der Wert der Variablen kein simpler String oder Integer ist, genügt es eventuell nicht, diese mit `setGlobal` als globale QF-Test Variable zu definieren, da der Wert beim Zugriff von/einem SUT-Skripte serialisiert werden muss. In solchen Fällen können Sie die Variable zum Beispiel in Jython als `global` deklarieren, um auf sie aus verschiedenen Knoten oder Prüfausdrücken derselben Skriptsprache zuzugreifen, wie es das folgende Beispiel zeigt.

```
global globalVar
globalVar = 10000
```

Beispiel 11.6: Globale Jython-Variable

`globalVar` steht nun in allen folgenden Jython-Skriptknoten zur Verfügung (in allen Jython-Server-Skripte oder in allen Jython-SUT-Skripte desselben Clients). Um den Wert von `globalVar` in einem anderen Jython-Skriptknoten zu verändern, ist erneut eine Deklaration mit dem Schlüsselwort `global` notwendig. Andernfalls wird eine neue lokale Variable mit gleichem Namen erzeugt. Um eine globale Jython Variable zu entfernen, kann die `del` Anweisung verwendet werden:

```
global globalVar
del globalVar
```

Beispiel 11.7: Löschen einer globalen Jython Variable

In Groovy und JavaScript werden globale Variablen noch einfacher erzeugt als in Jython. Die Regel lautet, dass undeklarierte Variablen im Binding des Skripts erwartet werden. Sind sie dort nicht zu finden, werden sie automatisch hinzugefügt.

```
myGlobal = 'global'
```

Beispiel 11.8: Definieren von globalen Variablen in Groovy bzw. JavaScript

```
assert myGlobal == 'global'
def globals = binding.variables
assert globals['myGlobal'] == 'global'
globals.remove('myGlobal')
assert globals.find { it == 'myGlobal' } == null
```

Beispiel 11.9: Verwenden und entfernen einer globalen Groovy Variablen

Austausch von Variablen zwischen verschiedenen Prozessen

Es kommt vor, dass Variablen, die in einem Prozess definiert wurden, später in einem anderen Prozess benötigt werden. So könnte zum Beispiel eine Liste von Werten, die mit Hilfe eines SUT-Skripts aus einer Tabelle gelesen werden, in einem Server-Skript weiterverwendet werden, um darüber zu iterieren.

Die einfachste Möglichkeit ist nun, die Liste mit Hilfe von `rc.setGlobal` oder `rc.setLocal` in einer QF-Test Variable zu speichern und später den Inhalt mit `rc.getObj` abzufragen. Dies funktioniert, sofern der gespeicherte Inhalt serialisierbar ist und die Objekte im anderen Prozess wieder hergestellt werden können. Andernfalls liefert `rc.getObj` automatisch den String-Wert des Objektes zurück - vergleichbar mit `rc.getStr`.

Als Alternative stellt der Runcontext einen symmetrischen Satz von Methoden zum Zugriff auf und zur Modifikation von Skript-Variablen in einem anderen Prozess bereit. Für SUT-Skripte sind dies die Methoden `toServer` und `fromServer`. Die entsprechenden Methoden für Server-Skripte heißen `toSUT` und `fromSUT`. Dabei müssen die Skript-Knoten jeweils die gleiche Skriptsprache verwenden.

Das folgende Jython Beispiel zeigt, wie ein SUT-Skript direkt eine globale Variable im Jython-Interpreter von QF-Test setzen kann:

```
cellValues = []
table = rc.getStr("idOfTable")
for i in range(table.getRowCount()):
    cellValues.append(table.getValueAt(i, 0))
rc.toServer(tableCells=cellValues)
```

Beispiel 11.10: Setzen einer Server-Variable aus einem SUT-Skript heraus

Nach Ausführung des obigen Skripts enthält die globale Variable namens ”tableCells” in QF-Test’s Jython-Interpreter das Array der Werte aus der Tabelle.

Hinweis

Die Tabellenwerte im obigen Beispiel sind nicht notwendigerweise Strings. Sie könnten Zahlen sein, Datumswerte, was auch immer. Leider ist der *pickle* Mechanismus von Jython nicht mächtig genug, um Instanzen von Java-Klassen zu transportieren (nicht einmal von serialisierbaren), sodass dieser Austauschmechanismus auf primitive Typen wie Strings und Zahlen sowie auf Jython Objekte und Strukturen wie Arrays und Dictionaries beschränkt ist.

11.3.4 Zugriff auf die GUI-Komponenten des SUT

Für SUT-Skripte bietet der Runcontext eine äußerst nützliche Methode. Durch den Aufruf von `rc.getComponent(”componentId”)` werden die Informationen aus dem Komponente(930)-Knoten mit der QF-Test ID(931) ”componentId” aus der Testsuite geholt und an den Mechanismus zur Wiedererkennung von Komponenten gereicht. Dieser arbeitet genau wie bei der Simulation eines Events, das heißt, er wirft auch die entsprechenden Exceptions, falls die Komponente nicht gefunden werden kann.

Im Erfolgsfall wird die Komponente an das Skript zurückgegeben und zwar nicht in Form von abstrakten Daten, sondern das konkrete Objekt. Alle Methoden, die die Java-API der Klasse dieser Komponente zur Verfügung stellt, können ausgeführt werden, um Informationen auszulesen oder um Effekte zu erzielen, die durch das GUI nicht möglich sind. Um eine Liste der Methoden einer Komponente anzuzeigen, siehe Abschnitt 5.12(107).

```
# get the custom password field
field = rc.getComponent("tfPassword")
# read its crypted value
passwd = field.getCryptedText()
rc.setGlobal("passwd", passwd)
# get the table component
table = rc.getComponent("tabAddresses")
# get the number of rows
rows = table.getRowCount()
rc.setGlobal("tableRows", rows)
```

Beispiel 11.11: Zugriff auf Komponenten mit `rc.getComponent`

Sie können auf diesem Weg auch auf Unterelemente zugreifen. Wenn der Parameter `componentId` ein Element referenziert, liefert `getComponent` ein Paar zurück, bestehend aus der Komponente und dem Index des Elements. Der Index kann dazu verwendet werden, den eigentlichen Wert zu ermitteln. Das folgende Beispiel zeigt, wie Sie den Wert einer Tabellenzelle auslesen. Beachten Sie dabei auch die praktische Methode mit der Jython das Auspacken von Sequenzen bei Zuweisungen unterstützt.

```
# first get the table and index
table, (row,column) = rc.getComponent("tableAddresses@Name@Greg")
# then get the value of the table cell
cell = table.getValueAt(row, column)
```

Beispiel 11.12: Zugriff auf Unterelemente mit `rc.getComponent`

11.3.5 Aufruf von Prozeduren

Der Runcontext kann auch dazu verwendet werden, Prozeduren(672) in QF-Test auszuführen.

```
rc.callProcedure("text.clearField",
         {"component" : "nameField", "message" : "nameField cleared"})
```

Beispiel 11.13: Einfacher Prozeduraufruf in Jython

In obigem Beispiel wird die Prozedur namens ”clearField” im Package(680) namens ”text” aufgerufen. Die Parameter für den Aufruf sind ”component” mit dem Wert ”nameField” und ”message” mit dem Wert ”nameField cleared”.

Dasselbe Beispiel mit der veränderten Groovy Syntax:

```
rc.callProcedure("text.clearField",
         ["component" : "nameField", "message" : "nameField cleared"])
```

Beispiel 11.14: Einfacher Prozeduraufruf in Groovy

Und in JavaScript:

```
rc.callProcedure("text.clearField",
         {"component" : "nameField", "message" : "nameField cleared"})
```

Beispiel 11.15: Einfacher Prozeduraufruf in JavaScript

Der Rückgabewert einer Prozedur, der mittels eines Return(678)-Knotens festgelegt werden kann, ist gleichzeitig der Rückgabewert des `rc.callProcedure` Aufrufs.

Hinweis

In einem SUT-Skript-Knoten sollte `rc.callProcedure` nur mit großer Vorsicht verwendet werden. Rufen Sie nur Prozeduren mit kurzer Laufzeit auf, die keine allzu komplexen Operationen im SUT auslösen. Andernfalls könnte eine `DeadlockTimeoutException`(960) verursacht werden. Wenn Daten für datengetriebene Tests zwingend im SUT ermittelt werden müssen, speichern Sie diese mittels `rc.setLocal` in einer Variable bzw. transferieren Sie diese mittels `rc.toServer` zu QF-Test's Interpreter und treiben Sie die Tests dann aus einem Server-Skript-Knoten, für den es keine derartigen Einschränkungen gibt.

11.3.6 Setzen von Optionen

3.1+

Viele der in Kapitel 41(482) beschriebenen Optionen können auch zur Laufzeit via `rc.setOption` gesetzt werden. Konstanten für die Namen dieser Optionen sind in der Klasse `Options` definiert, welche in den Skriptsprachen automatisch verfügbar ist.

Ein reelles Beispiel, bei dem es sinnvoll ist, eine Option temporär zu setzen, ist die Wiedergabe eines Events auf eine deaktivierte Komponente. Für diesen Sonderfall muss die Überprüfung durch QF-Test auf den enabled/disabled Zustand verhindert werden. Zum Setzen mit umgehenden Zurücksetzen gibt es die Variante `pushOption` / `popOption`, bei der vorhergehende `setOption` Aufrufe nicht verloren gehen:

```
rc.pushOption(Options.OPT_PLAY_THROW_DISABLED_EXCEPTION, false)
```

Beispiel 11.16: pushOption

Nach Abspielen des speziellen Events kann der vorhergehende Wert der Option wiederhergestellt werden, wie in folgendem Beispiel gezeigt:

```
rc.popOption(Options.OPT_PLAY_THROW_DISABLED_EXCEPTION)
```

Beispiel 11.17: popOption

Möchte man ganz sicher gehen, dass der Wert korrekt zurückgesetzt wird, sollten die beiden Skript-Knoten in einer Try[704] / Finally[711] Kombination verbaut werden. Andernfalls würde z.B. eine `ComponentNotFoundException` beim Abspielen des Events das Zurücksetzen verhindern.

Hinweis

Achten Sie darauf, dass Sie QF-Test Optionen immer in einem Server-Skript-Knoten und SUT-Optionen in einem SUT-Skript-Knoten setzen, andernfalls hat die Aktion keinen Effekt. Einige Optionen - speziell für SmartIDs - haben Effekte sowohl auf QF-Test, als auch auf SUT Seite. Diese müssen in einem Server-Skript-Knoten geändert werden. QF-Test leitet diese Änderung automatisch an die SUT-Clients weiter. Die Dokumentation der Optionen in Kapitel 41[482] führt für jede Option die betroffene Seite - Server und/oder SUT - auf.

11.3.7 Komponenten bei Bedarf explizit setzen

Es können Fälle auftreten, in denen Sie eine bestimmte Komponente nicht über einen Komponente[930]-Knoten definieren können, sondern auf Skript-Ebene suchen müssen, um mit dieser arbeiten zu können, sei es, um aus Performance-Gründen dieselbe Komponente mehrfach zu nutzen oder für Spezialfälle in denen die normale Erkennung zu kompliziert oder ineffektiv ist. Für solche Fälle können Sie die Methode `rc.overrideElement` verwenden, um die gefundene Komponente einer QF-Test ID oder SmartID zuzuordnen. Anschließend können Sie mit den gewohnten QF-Test Knoten mit dieser ID arbeiten.

Hinweis

Das folgende Beispiel wäre zwar heute mit SmartID sehr einfach zu lösen, ist aber immer noch illustrativ. Für komplexere Fälle bleibt `overrideElement` weiterhin relevant.

Stellen Sie sich vor, wir möchten immer mit dem ersten Textfeld eines Panels arbeiten. Jedoch könnte das einfache Aufzeichnen der Textfelder nicht möglich sein, da sich der Inhalt zu stark ändert. Nun können wir ein Skript implementieren, welches das erste Textfeld sucht. Dann können wir dieses gefundene Textfeld einer Komponente `PriorityAwtSwingComponent` aus der Standardbibliothek `qfs.qft` zuordnen. Nachdem wir das Skript ausgeführt haben, können mit der Angabe der QF-Test ID `PriorityAwtSwingComponent` alle gewohnten QF-Test Knoten benutzen um mit dem gefundenen Textfeld zu arbeiten.

```
panel = rc.getComponent("myPanel")
for component in panel.getComponents():
    if qf.isInstance(component, "javax.swing.JTextField"):
        rc.overrideElement("PriorityAwtSwingComponent", component)
        break
```

Beispiel 11.18: Jython rc.overrideElement

Dieses Konzept ist sehr nützlich, wenn Sie einen Algorithmus kennen, um ihre Zielkomponenten für bestimmte Testschritte zu suchen.

Sie können solche (veraltete, s.u.) Priority-Komponenten für alle unterstützten Engines in der Standardbibliothek `qfs.qft` finden. Ein Beispiel finden Sie auch in Ihrer QF-Test Installation in der mitgelieferten Testsuite `carconfigSwing_advanced_de.qft` im Verzeichnis `demo/carconfigSwing`.

7.0+ Vor der Einführung von SmartIDs musste die QF-Test ID[931] eines Komponente[930]-Knotens als `id` Parameter angegeben werden. Bei Verwendung einer SmartID entfällt diese Anforderung. Sie können eine SmartID beliebig wählen, sie muss nur mit # beginnen. Diese Funktion basiert auf einem einfachen String-Vergleich, eventuell gesetzte Scopes gehen auf dieser Ebene nicht ein! Ebenso neu in QF-Test 7.0 ist die Möglichkeit, überladene Elemente via `rc.getOverrideElement` auszulesen. Im folgenden Beispiel werden beide Möglichkeiten kombiniert.

```
if not rc.getOverrideElement("#FirstTextField"):
    panel = rc.getComponent("myPanel")
    for component in panel.getComponents():
if qf.isInstance(component, "javax.swing.JTextField"):
    rc.overrideElement("#FirstTextField", component)
    break
```

Beispiel 11.19: Nutzung von rc.overrideElement mit SmartID und Bedingung

11.4 Jython-Skripting

Hinweis Jython basiert auf Python 2 und nicht Python 3. Wenn also in diesem Handbuch nur von ”Python” ohne genauere Angabe die Rede ist, ist immer Python 2 gemeint.

Python ist eine vielseitige, objektorientierte Skriptsprache, die von Guido van Rossum entworfen und in C implementiert wurde. Hilfreiche Informationen zu Python gibt es unter http://www.python.org. Python ist eine standardisierte Sprache und seit vielen Jahren etabliert. Umfassende Dokumentation dazu ist frei verfügbar, daher beschränkt sich dieses Handbuch darauf, die Integration von Jython in QF-Test zu erklären. Die Sprache selbst ist sehr natürlich und intuitiv. Ihre größte Stärke ist die Verständlichkeit und Les-

barkeit von Python-Skripten. Daher sollten Sie keine Probleme haben, die folgenden Beispiele zu verstehen.

Jython (früher JPython genannt) ist eine Implementierung von Version 2 der Programmiersprache Python in Java. Jython hat dieselbe Syntax wie Python und verfügt über beinahe identische Features. Die Objektsysteme von Java und Jython haben vieles gemeinsam und Jython kann nahtlos in Anwendungen wie QF-Test integriert werden. Das macht es zu einem äußerst nützlichen Werkzeug für Java-Skripting. Jython hat seine eigene Homepage unter http://www.jython.org. Dort gibt es unter anderem auch ein ausführliches Tutorial zum Einstieg.

QF-Test verwendet die Jython Version 2.7, die einen Großteil der Standard Python 2 Bibliothek unterstützt.

Die Skriptsprache Jython wird in QF-Test nicht nur in Server-Skript(717) und SUT-Skript(720) Knoten verwendet, sondern auch in `$[...]`-Ausdrücken und standardmäßig zur Auswertung von Bedingungen wie im Attribut Bedingung(694) von If(693)-Knoten.

11.4.1 Jython-Variablen

Hinweis

In Jython-Skripten werden QF-Test Variablenreferenzen der Form `$(var)` oder `${Gruppe:Name}` vor Ausführung des Skripts expandiert. Dies kann zu unerwünschten Effekten führen, insbesondere wenn die Werte dieser Variablen Zeilenumbrüche oder Rückstriche (\) enthalten. Es sollten stattdessen die Methoden `rc.getStr()` und `rc.getObj()` etc. (vgl. Abschnitt 11.3.3(192)) oder `rc.vars` und `rc.groups` (vgl. Abschnitt 6.1.3(117)) verwendet werden, die erst während der Ausführung des Skripts ohne Risiko evaluiert werden.

11.4.2 Module

Module für Jython in QF-Test sind nichts anderes als gewöhnliche Python-Module. Sie können Module in QF-Test importieren und deren Methoden aufrufen, was die Entwicklung komplexer Skripte stark vereinfacht und außerdem die Wartbarkeit Ihrer Tests erhöht, da Module testsuiteübergreifend verfügbar sind.

Module, die Sie für mehrere Testsuiten zur Verfügung stellen wollen, sollten Sie im `jython` Verzeichnis unter QF-Tests Wurzelverzeichnis ablegen. Module, die speziell für eine Testsuite geschrieben sind, können auch direkt im selben Verzeichnis wie die Testsuite liegen. Das versionsspezifische Verzeichnis `qftest-9.0.0/jython/Lib` ist für mitgelieferte Module reserviert. Jython-Module haben die Endung `.py`.

Das folgende Beispiel zeigt ein Jython-Modul, das eine Prozedur zur Verfügung stellt, die eine Liste von Zahlen sortiert:

```
def insertionSort(alist):
    for index in range(1,len(alist)):
        currentvalue = alist[index]
        position = index
        while position>0 and alist[position-1]>currentvalue:
            alist[position]=alist[position-1]
            position = position-1
        alist[position]=currentvalue
```

Beispiel 11.20: Das Jython-Modul `pysort.py`

Das folgende Jython-Skript ruft die im Modul definierte Prozedur auf.

```
import pysort
alist = [54,26,93,17,77,31,44,55,20]
pysort.insertionSort(alist)
print(alist)
```

Beispiel 11.21: Jython-Skript mit Verwendung eines Moduls

11.4.3 Post-mortem Fehleranalyse von Jython-Skripten

In Python gibt es einen einfachen zeilenorientierten Debugger namens `pdb`. Zu seinen nützlichen Features gehört die Möglichkeit zu analysieren, warum ein Skript mit einer Exception fehlgeschlagen ist. In Python können Sie hierzu einfach nach einer Exception das `pdb` Modul importieren und `pdb.pm()` ausführen. Damit gelangen Sie in eine Debugger-Umgebung in der Sie die Werte der Variablen zum Zeitpunkt des Fehlers betrachten und auch den Call-Stack hinauf navigieren können um dort weitere Variablen zu analysieren. Das Ganze ist vergleichbar mit der Analyse eines Core-Dump einer C-Anwendung.

Obwohl Jython den `pdb` Debugger grundsätzlich unterstützt, funktioniert er aus verschiedenen Gründen in QF-Test nicht besonders gut, aber immerhin ist die post-mortem Analyse von Skripts über die Jython-Konsolen möglich. Nach einem fehlgeschlagenen Server-Skript(717)-Knoten öffnen Sie QF-Test's Jython-Konsole, für ein gescheitertes SUT-Skript(720) die Jython-Konsole des entsprechenden SUT, und geben dort einfach `debug()` ein. Dies sollte denselben Effekt wie das oben beschriebene `pdb.pm()` haben. Weitere Informationen zum Python-Debugger entnehmen Sie bitte der Python-Dokumentation.

Eine Schritt-für-Schritt-Anleitung, wie Jython-Scripte in QF-Test mit externen Werkzeugen debugged werden können, finden Sie in unserem Blogartikel Wie kann man Jython Scripts in QF-Test debuggen? .

11.4.4 Boolean-Typ

Jython hat einen echten Boolean-Typ mit den Werten `True` und `False`. In älteren Versionen dienten die Integer-Werte 0 und 1 als Boolean-Werte. Dies kann zu Problemen führen, wenn das Ergebnis eines Aufrufs wie `file.exists()` einer QF-Test Variable zugewiesen wird, z.B. "fileExists", und später in einem Bedingung(694)-Attribut in der Form `$(fileExists) == 1` ausgewertet wird. Derartige Bedingungen sollten grundsätzlich in der einfachen Form `$(fileExists)` bzw. `rc.getBool("fileExists")` geschrieben werden, die mit allen Jython-Versionen funktioniert.

11.4.5 Jython Strings und Zeichenkodierung

Zusammenfassung und Hinweise

5.3+ Zeichen in Jython Literalen wie `"abc"` waren auf 8 Bit limitiert, was zu Problemen bei Verwendung von internationalen Zeichen führte.

QF-Test Version 5.3 ermöglicht die Verwendung von internationalen Zeichen in Jython Skripten und Bedingung(694) Attributen basierend auf der Option Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java)(485).

Falls Sie QF-Test erst seit Version 5.3. oder höher verwenden, ist diese Option standardmäßig aktiv.

Ein kleiner Teil von bestehenden Skripten muss beim Umschalten auf Unicode Literale angepasst werden. Daher bleibt die Option zunächst deaktiviert, falls QF-Test eine bestehende ältere Systemkonfiguration antrifft. Es wird wärmstens empfohlen, diese Option zu aktivieren. Der Abschnitt "Problembehandlung" weiter unten erklärt, was im Fall von dadurch auftretenden Problemen zu tun ist.

Wenn Jython Unicode Literale aktiviert sind, sollte für maximale Flexibilität die Option Standard-Zeichenkodierung für Jython(486) auf "utf-8" gesetzt werden.

Unabhängig von den eingestellten Option sollte vor allen Dingen die Expansion von QF-Test Variablen in Literalen verhindert werden. Ausdrücke der Form `"$(somevar)"` können zu Syntaxfehlern oder unerwarteten Ergebnissen führen, wenn der Wert der Variable Zeilenumbrüche oder Rückstriche ('\') enthält. Verwenden Sie stattdessen `rc.getStr("somevar")`.

Hintergründe und Werdegang von Jython in QF-Test

Alle Java-Strings sind Sequenzen von 16-Bit Zeichen. Jython kennt hingegen zwei Arten von Strings: 8-Bit Byte-Strings (type <str>) und 16-Bit Unicode-Strings (type <unicode>). Der überwiegende Anteil von Strings in QF-Test Jython Skripten sind entweder Konstante Zeichenketten wie `"abc"`, genannt Literale, oder Java-Strings, die nach Jython konvertiert werden, wie das Ergebnis von `rc.getStr("varname")`. Die Konvertierung aus Java führt immer zu 16-Bit Unicode-Strings. Für Literale hängt das Ergebnis

von der Option Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java)[485] ab.

Wenn Unicode und Byte-Strings verglichen oder zusammengefügt werden, muss Jython eine Form in die andere konvertieren. Die Konvertierung von Unicode zu Byte-Strings heißt Enkodierung, die umgekehrte Richtung Dekodierung. Es gibt viele verschiedene Wege, 16-Bit Strings in 8-Bit Sequenzen zu kodieren und die Regeln dafür heißen Zeichenkodierung. Typische Beispiele hiefür sind ”utf-8” oder ”latin-1”. Die Option Standard-Zeichenkodierung für Jython[486] legt fest, welche Kodierung Jython verwenden soll, wenn keine explizite angegeben ist. Aus Kompatibilitätsgründen war vor QF-Test 5.3 der Standardwert ”latin-1”. Inzwischen ist er ”utf-8”, weil diese Kodierung flexibler ist und alle internationalen Zeichensätze unterstützt.

Jython in QF-Test basiert auf Python Version 2. In früheren Python Versionen bestanden Strings stets aus 8-Bit Zeichen. Später kamen Unicode-Strings mit 16-Bit Zeichen hinzu. In Python 2 sind Literale wie `”abc”` 8-Bit Byte-Strings, das Voransetzen von ’u’, also `u”abc”` macht daraus Unicode-Strings. In Python 3 sind Literale bereits Unicode-Strings und können durch Voransetzen von ’b’, also `b”abc”` zu Byte-Strings gemacht werden.

In Jython 2.2 wurden Java-Strings in 8-Bit Python-Strings konvertiert, basierend auf der Standard-Zeichenkodierung der Java-VM, in der westlichen Hemisphäre üblicherweise ISO-8859-1 (auch als latin-1 bekannt). Seit Jython 2.5 werden Java Strings grundsätzlich als Unicode Jython Strings interpretiert. Zusammen mit 8-Bit String-Literalen führt dies zu viel implizierter Konvertierung zwischen Byte-Strings und Unicode-Strings, z.B. wenn ein - nun als Unicode interpretierter - Java-String und ein Literal verknüpft werden, wie in `rc.getStr(”path”) + ”/file”`.

5.3+ Vor QF-Test Version 5.3 hatten Jython Skripte durch die Art, wie der Code von QF-Test an den Jython Compiler übergeben wurde, weitere Probleme mit Zeichen außerhalb des 8-Bit Bereichs. Im Zuge der Behebung dieser Probleme stellte es sich heraus, dass der beste Weg zur Behebung der Problem mit Jython String-Literalen die Adaption eines bereits in Python 2 vorhanden Features ist, nämlich `from future import unicode_literals`, um Jython Literale in QF-Test generell als Unicode-Strings zu behandeln. Dadurch sind String-Literale nun in allen Skriptsprachen von QF-Test einheitlich und voll kompatibel mit Java-Strings, so dass die Interaktion zwischen Jython und allem anderen in QF-Test viel natürlicher wird. Die neue Option Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java)[485] bestimmt, ob String-Literale in Jython als Unicode-Strings behandelt werden. Aus Kompatibilitätsgründen bleibt es bei 8-Bit Byte-Strings, falls QF-Test beim Start auf eine ältere bestehende Systemkonfiguration trifft, andernfalls sind Unicode Literale nun der Standard.

Die empfohlenen Einstellungen für die Jython Optionen sind aktiviert für Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java)[485] und ”utf-8” für Standard-Zeichenkodierung für Jython[486].

Behandlung von Problemen mit Jython und Zeichenkodierungen

Wie in den vorherigen Abschnitten beschrieben, verfügt Jython über zwei Arten von Strings, `<type 'str'>` für 8-Bit Byte-Strings und `<type 'unicode'>` für 16-Bit Unicode-Strings. Literale kann ein 'b' vorangestellt werden (`b"abc"`) um Byte-Strings zu erhalten und ein 'u' (`u"abc"`) für Unicode-Strings. Nicht näher gekennzeichnete Literale (`"abc"`) sind Unicode, falls die Option Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java)[485] aktiviert ist, andernfalls Byte-Strings. Java-Strings aus einem Java-Funktionsaufruf wie `rc.getStr("somevar")` sind immer Unicode-Strings.

Die folgenden Hinweise sollten Ihnen dabei helfen, Probleme mit Jython und Zeichenkodierungen zu minimieren:

- Schalten Sie die Option Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java)[485] ein und setzen Sie die Option Standard-Zeichenkodierung für Jython[486] auf "utf-8".

- String-Literale mit $()-Expansion wie `"$(varname)"` waren immer schon problematisch und sollten durch `rc.getStr("varname")` oder `rc.vars.varname` ersetzt werden.

- Strings mit Windows-Dateinamen brauchen wegen der enthaltenen Rückstriche ('\') spezielle Behandlung. In 8-Bit-Strings werden Rückstriche beibehalten, wenn sie keine Sonderfunktion wie '\t' für Tab oder '\n' für einen Zeilenumbruch haben. In 16-Bit-Strings gibt es wesentlich mehr sogenannte Escape-Sequenzen mit besonderer Bedeutung, die zu Syntaxfehlern oder unerwarteten Ergebnissen führen können. Probleme können durch Verwendung von `rc.getStr("filename")` oder `rc.vars.filename` (siehe oben) und Voranstellen von 'r' (für "raw string") bei Angabe von Literalen, z.B. `qftestDir = r"C:\Program Files\QFS\QF-Test"`, vermieden werden.

- Verwenden Sie grundsätzlich `qf.println` anstelle von `print ...`, da letzteres durch einen 8-Bit-Stream mit der Standardkodierung von Java (und im Falle eines SUT-Skript[720]-Knotens zusätzlich mit der des Betriebssystems) durchgeschleift wird und dadurch internationale Zeichen leicht verloren gehen.

- Die Konvertierung von Objekten in Strings wurde in Jython traditionell via `str(some_object)` vorgenommen. Da `str` der Typ von Byte-Strings ist, erzeugt dies immer einen Byte-String und erzwingt damit Enkodierung. Wenn Sie nicht ausdrücklich einen Byte-String benötigen, sollten Sie stattdessen `unicode(some_object)` verwenden.

- Das Jython-Modul `types` beinhaltet die Konstanten `types.StringType` und `types.UnicodeType` sowie die Liste `types.StringTypes` mit beiden Typen. Letztere ist sehr hilfreich, um zu prüfen, ob ein Objekt von irgendeinem String-Typ

ist, egal ob 8-Bit oder 16-Bit. Statt
`if type(some_object) == types.StringType`
sollte lieber
`if type(some_object) in types.StringTypes`
verwendet werden.

- In den wenigen Fällen, in denen Sie wirklich ein 8-Bit Byte-String Literal benötigen, setzen Sie ein 'b' voran, z.B.
`array.array(b'i', [1, 2, 3])`

Und natürlich ist unser Support immer für Sie da.

11.4.6 Den Namen einer Java-Klasse ermitteln

Diese einfache Operation ist in Jython überraschend schwierig. Bei einem gegebenen Java Objekt würde man den Namen der Klasse einfach mittels `obj.getClass().getName()` bestimmen. Für manche Objekte funktioniert das auch in Jython, für andere scheitert es mit einer kryptischen Fehlermeldung, was recht frustrierend sein kann. Es geht immer dann schief, wenn die Klasse selbst auch eine `getName` Methode implementiert. Dies ist für AWT `Component` der Fall, so dass es für alle AWT/Swing Komponenten schwierig ist, den Namen ihrer Klasse zu ermitteln.

Die einzige Lösung, die zuverlässig funktioniert ist:

```
from java.lang import Class
Class.getName(obj.getClass())
```

Da der Code nicht gerade intuitiv ist, haben wir ein neues Modul namens `qf` mit praktischen Methoden initiiert. Es ist automatisch verfügbar, so dass Sie nun einfach folgendes schreiben können:

```
qf.getClassName(obj).
```

11.4.7 Ein komplexes Beispiel

Wir schließen diesen Abschnitt mit einem komplexen Beispiel ab, das Features von Jython und QF-Test kombiniert, um einen datengetriebenen Test durchzuführen. Wir gehen für dieses Beispiel von einer einfachen Tabelle mit den drei Spalten "Name", "Age" und "Address" aus, die mit Werten gefüllt werden soll, die aus einer Datei gelesen werden. Die Datei soll dabei im "Comma-Separated-Values" Format vorliegen, mit '|' als Trennzeichen, eine Zeile pro Tabellenzeile, z.B.:

```
John Smith|45|Some street, some town
Julia Black|35|Another street, same town
```

Das Beispiel testet die Funktionalität des SUT neue Tabellenzeilen zu erstellen. Dabei kommt eine QF-Test Prozedur zum Einsatz, die 3 Parameter erwartet - ”name”, ”age” und ”address” - und mit diesen eine neue Tabellenzelle anlegt und füllt. Im Jython-SUT-Skript wird die Datei mit den Werten eingelesen und geparst. In einer Schleife wird über die Datensätze iteriert und für jede zu erstellende Tabellenzeile die Prozedur aufgerufen. Der Name für die Datei wird in der QF-Test Variable namens ”filename” übergeben. Wenn das Füllen der Tabelle abgeschlossen ist, wird der Endzustand der Tabelle mit den eingelesenen Werten verglichen, um sicher zu gehen, dass alles geklappt hat.

```
import string
data = []
# read the data from the file
fd = open(rc.getStr("filename"), "r")
line = fd.readline()
while line:
    # remove whitespace
    line = string.strip(line)
    # split the line into separate fields
    # and add them to the data array
    if len(line) > 0:
        data.append(string.split(line, "|"))
    line = fd.readline()
# now iterate over the rows
for row in data:
    # call a qftest procedure to create
    # one new table row
    rc.callProcedure("table.createRow",
                     {"name": row[0], "age": row[1],
                      "address": row[2]})
# verify that the table-rows have been filled correctly
table = rc.getComponent("tabAddresses")
# check the number of rows
if table.getRowCount() != len(data):
    rc.logError("Row count mismatch")
else:
    # check each row
    for i in range(len(data)):
        if str(table.getValueAt(i, 0)) != data[i][0]:
            rc.logError("Name mismatch in row " + str(i))
        if str(table.getValueAt(i, 1)) != data[i][1]:
            rc.logError("Age mismatch in row " + str(i))
        if str(table.getValueAt(i, 2)) != data[i][2]:
            rc.logError("Address mismatch in row " + str(i))
```

Beispiel 11.22: Ein datengetriebener Test

Natürlich dient obiges Beispiel nur zur Anschauung. Es ist viel zu komplex, um halbwegs komfortabel in QF-Test editiert werden zu können. Außerdem sind zu viele Dinge fest verdrahtet, so dass es mit der Wiederverwendbarkeit nicht weit her ist. Für eine echte Anwendung würde man den Code zum Einlesen und Parsen der Datei parametrisieren und in ein Modul auslagern, ebenso den Code zur Verifikation der Tabelle.

Dies geschieht im folgenden Jython-Skript mit den Methoden `loadTable` zum Lesen der Daten aus der Datei und `verifyTable` zum Überprüfen der Tabelle. Es wird in einem Modul namens `csvtable.py` abgespeichert. Ein Beispiel dafür finden Sie in `qftest-9.0.0/doc/tutorial/csvtable.py`. Zur Erläuterung genügt folgende vereinfachte Version:

```
import string
def loadTable(file, separator="|"):
    data = []
    fd = open(file, "r")
    line = fd.readline()
    while line:
        line = string.strip(line)
        if len(line) > 0:
            data.append(string.split(line,separator))
        line = fd.readline()
    return data
def verifyTable(rc, table, data):
    ret = 1
    # check the number of rows
    if table.getRowCount() != len(data):
        if rc:
            rc.logError("Row count mismatch")
        return 0
    # check each row
    for i in range(len(data)):
        row = data[i]
        # check the number of columns
        if table.getModel().getColumnCount() != len(row):
            if rc:
                rc.logError("Column count mismatch " +
                            "in row " + str(i))
            ret = 0
        else:
            # check each cell
            for j in range(len(row)):
                val = table.getModel().getValueAt(i, j)
                if str(val) != row[j]:
                    if rc:
                        rc.logError("Mismatch in row " +
                                    str(i) + " column " +
                                    str(j))
                    ret = 0
    return ret
```

Beispiel 11.23: Schreiben eines Moduls

Der obige Code sollte Ihnen bekannt vorkommen. Er ist eine verbesserte Version von Teilen von Beispiel 11.22(206). Ist dieses Modul installiert, vereinfacht sich der Code, der in QF-Test geschrieben werden muss, wie folgt:

```
import csvtable
# load the data
data = csvtable.loadTable(rc.getStr("filename"))
# now iterate over the rows
for row in data:
    # call a qftest procedure to create
    # one new table row
    rc.callProcedure("table.createRow",
                     {"name": row[0], "age": row[1],
                      "address": row[2]})
# verify that the table-rows have been filled correctly
table = rc.getComponent("tabAddresses")
csvtable.verifyTable(rc, table, data)
```

Beispiel 11.24: Aufruf von Methoden in einem Modul

11.5 Groovy-Skripting

Groovy ist eine weitere etablierte Skriptsprache für die Java-Platform. Sie wurde von James Strachan and Bob McWhirter im Jahre 2003 entwickelt. Im Grunde ist alles was man für Groovy braucht, eine Java-Laufzeitumgebung (JRE) und die Datei `groovy-all.jar`. Diese Bibliothek enthält sowohl einen Compiler, um Java .class Dateien zu erstellen, wie auch die entsprechende Laufzeitumgebung, um diese Klassen in der Java Virtual Machine (JVM) auszuführen. Man kann sagen, Groovy ist Java mit einer zusätzlichen `.jar` Datei. Im Gegensatz zu Java ist Groovy allerdings eine dynamische Sprache, was bedeutet, dass das Verhalten von Objekten erst zur Laufzeit ermittelt wird. Außerdem können Klassen auch direkt aus dem Skriptcode geladen werden, ohne erst Class-Dateien erzeugen zu müssen. Schließlich lässt sich Groovy auch leicht in Java-Anwendungen wie QF-Test einbetten.

Die Groovy Syntax ist ähnlich der von Java, vielleicht ausdrucksstärker und leichter zu lesen. Wenn man von Java kommt, kann man sich dem Groovy Stil nach und nach annähern. Wir können hier natürlich nicht die Sprache Groovy in allen Details besprechen, dazu sei auf die Groovy Homepage http://groovy-lang.org/ oder das exzellente Buch ”Groovy in Aktion” von Dierk Koenig u.a. verwiesen. Vielleicht können aber die folgenden Hinweise einem Java-Programmierer beim Einstieg in Groovy helfen.

- Das Semikolon ist optional, solange eine Zeile nur ein Statement enthält.
- Klammern sind manchmal optional, zum Beispiel bedeutet `println 'hello qfs'` dasselbe wie `println('hello qfs')`.
- Anstelle von `for (int i = 0; i < len; i++) { ... }` verwende man `for (i in 0..<len) { ... }`.

- Die folgenden Importe werden bei Groovy standardmäßig vorgenommen: `java.lang.*`, `java.util.*`, `java.io.*`, `java.net.*`, `groovy.lang.*`, `groovy.util.*`, `java.math.BigInteger`, `java.math.BigDecimal`.
- Alles ist ein Objekt, sogar Integer oder Boolean Werte wie '1' oder 'true'.
- Anstelle von Getter- und Setter-Methoden wie `obj.getXxx()` kann man einfach `obj.xxx` verwenden.
- Der Operator == prüft auf Gleichheit statt auf Identität, so dass Sie `if (somevar == "somestring")` statt `if (somevar.equals("somestring"))` verwenden können. Um auf Identität zu prüfen, gibt es die Methode `is()`.
- Variablen haben einen dynamischen Typ, wenn sie mit dem Schlüsselwort `def` deklariert werden. `def x = 1` zum Beispiel erlaubt es, der Variablen `x` später auch einen `String` zuzuweisen.
- Arrays werden etwas anders als in Java definiert, z. B. `int[] a = [1, 2, 3]` oder `def a = [1, 2, 3] as int[]`. Mit `def a = [1, 2, 3]` wird in Groovy eine Liste definiert.
- Groovy erweitert die Java-Bibliothek indem für viele Klassen zusätzliche Methoden definiert werden. So kann in einem Groovy-Skript etwa die Methode `isInteger()` auf ein `String` Objekt angewendet werden. Diese Erweiterungen werden als *GDK* bezeichnet (analog zu *JDK* in Java). Eine Liste der GDK-Methoden für ein Objekt `obj` liefert der Ausdruck `obj.class.metaClass.metaMethods.name` oder - übersichtlicher - das folgende Beispiel:

```
import groovy.inspect.Inspector
def s = 'abc'
def inspector = new Inspector(s)
def mm = inspector.getMetaMethods().toList().sort() {
    it[Inspector.MEMBER_NAME_IDX] }
for (m in mm) {
    println(m[Inspector.MEMBER_TYPE_IDX] + ' ' +
            m[Inspector.MEMBER_NAME_IDX] +
            '(' + m[Inspector.MEMBER_PARAMS_IDX] + ')')
}
```

Beispiel 11.25: GDK-Methoden für ein `String` Objekt

- Innere Klassen werden nicht unterstützt. In den meisten Fällen können stattdessen *Closures* verwendet werden. Eine `Closure` ist ein Object, das einen

Code-Schnipsel repräsentiert. Sie kann Parameter haben und auch ein Wert zurückliefern. Genau wie ein Block wird eine `Closure` in geschweiften Klammern definiert. Blöcke gibt es nur im Zusammenhang mit `class`, `interface`, statischer oder Objekt-Initialisierung, Methodenrümpfen, `if`, `else`, `synchronized`, `for`, `while`, `switch`, `try`, `catch` und `finally`. Jedes andere Vorkommen von `{...}` ist eine `Closure`. Als Beispiel schauen wir uns die GDK-Methode `eachFileMatch` der Klasse `File` an. Sie hat zwei Parameter: einen Filter (z. B. ein `Pattern` Objekt) und eine `Closure`. Diese `Closure` hat selbst auch einen Parameter: ein `File` Object, das die gerade gefundene Datei repräsentiert.

```
def dir = rc.getStr('qftest', 'suite.dir')
def pattern = ~/.*\.qft/
def files = []
new File(dir).eachFileMatch(pattern) { file ->
    files.add(file.name)
}
files.each {
    // Auf ein einzelnes Closure-Argument kann mit "it" zugegriffen werden.
    rc.logMessage(it)
}
```

Beispiel 11.26: Closures

- Mit Listen (`List`) und Dictionaries (`Map`) lässt es sich in Groovy viel leichter arbeiten als in Java.

```
def myList = [1, 2, 3]
assert myList.size() == 3
assert myList[0] == 1
myList.add(4)
def myMap = [a:1, b:2, c:3]
assert myMap['a'] == 1
myMap.each {
    this.println it.value
}
```

Beispiel 11.27: Listen und Dictionaries

11.5.1 Groovy Packages

Genau wie Java-Klassen werden Groovy-Skriptdateien (`.groovy`) in Packages organisiert. Diejenigen, welche suiteübergreifend Anwendung finden, stellt man am

besten in den `groovy`-Ordner unterhalb des QF-Test Wurzelverzeichnisses. Dateien bzw. Packages, die speziell für eine Testsuite entwickelt worden sind, können auch im Verzeichnis der Testsuite abgelegt werden. Das versionsspezifische Verzeichnis `qftest-9.0.0/groovy` ist für Groovy-Dateien reserviert, die von Quality First Software GmbH bereitgestellt werden.

```
package my
class MyModule
{
    public static int add(int a, int b)
    {
        return a + b
    }
}
```

Beispiel 11.28: `MyModule.groovy`

Die Datei `MyModule.groovy` könnte etwa im Unterverzeichnis `my` unterhalb des Testsuite-Verzeichnisses abgespeichert werden. Die Methode `add` aus `MyModule` kann dann folgendermaßen aufgerufen werden:

```
import my.MyModule as MyLib
assert MyLib.add(2, 3) == 5
```

Beispiel 11.29: Using MyModule

Dieses Beispiel demonstriert gleichzeitig noch ein weiteres Groovy Feature: *Type Aliasing*. Indem `import` und `as` zusammen verwendet werden, kann man eine Klasse über einen Namen eigener Wahl referenzieren.

11.6 JavaScript-Skripting

JavaScript hat sich vor allem im Bereich der Webentwicklung durchgesetzt und ist dort eine sehr beliebte Programmiersprache. QF-Test unterstützt ECMAScript, das entwickelt wurde um einen Standard für JavaScript bereitzustellen.

Um JavaScript verwenden zu können muss QF-Test mindestens mit Java 8 ausgeführt werden.

Dabei muss der ECMAScript 6 Standard in den JavaScript-Skripten verwendet werden. QF-Test führt automatisch eine interne Übersetzung auf den ECMAScript 5 Standard durch. Im Fehlerfall wird der übersetzte Code im Protokoll im Skript-Knoten aufgeführt, falls dieser vom Original-Code abweicht.

Einige Besonderheiten von JavaScript gegenüber anderen Skriptsprachen.

- Es gibt zwei verschiedene null-Werte: `undefined` und `null`. Eine Variable ist `undefined`, wenn sie keinen Wert besitzt. `null` ist ein beabsichtigter Null-Wert der zugewiesen werden muss.
- Der Operator == prüft auf Gleichheit statt auf Identität, so dass `if (3 == "3")` "true" ergibt. Um auf Identität, dass heißt Gleichheit von Typ und Wert beider Operanten, zu prüfen, gibt es den === Operator.
- Variablen haben einen dynamischen Typ, wenn sie mit dem Schlüsselwort `let` deklariert werden. `let x = 1` zum Beispiel erlaubt es, der Variablen `x` später auch einen `String` zuzuweisen. Konstanten werden mit `const` definiert.

11.6.1 Module

Auch in JavaScript können häufig benötigte Funktionen in Module ausgelagert werden. Diese müssen analog zu Jython bzw. Groovy in das `javascript`-Verzeichnis im QF-Test Wurzelverzeichnis gelegt werden.

Im folgenden Beispiel werden die Funktionen des Moduls `moremath.js` ausgelagert. Zunächst der Aufbau des Moduls:

```
var fibonacci = function(n) {
    return n < 1 ? 0
                 : n <= 2 ? 1
                 : fibonacci(n - 1) + fibonacci(n - 2);
}
function sumDigits(number) {
    var str = number.toString();
    var sum = 0;
    for (var i = 0; i < str.length; i++) {
        sum += parseInt(str.charAt(i), 10);
    }
    return sum;
}
// Module exports (Node.js style)
exports.fibonacci = fibonacci;
exports.sumDigits = sumDigits;
```

Beispiel 11.30: Das Modul `moremath.js`

In dem Modul `moremath.js` sind zwei Funktionen definiert: `fibonacci` und `sumDigits`. `fibonacci` berechnet den Wert der Fibonacci-Zahl an der Stelle n und `sumDigits` bildet die Quersumme. Jede Funktion muss exportiert werden, damit sie

für den Import zur Verfügung steht. Dies geschieht mit der an Node.js angelehnten Funktion `exports`.

Im Skript-Knoten kann nun der folgende Code verwendet werden um auf die Funktionen des Moduls `moremath.js` zuzugreifen:

```
moremath = require('moremath');
              console.log(moremath.fibonacci(13));
              console.log(moremath.sumDigits(123));
```

Beispiel 11.31: `Verwendung des moremath.js-Moduls`

Module die von QF-Test bereitgestellt werden, können über die import-Funktion importiert werden.

```
import {Autowin} from 'autowin';
            Autowin.doClickHard(10, 10, true);
```

Beispiel 11.32: `Verwendung des autowin-Moduls`

Java-Klassen können ebenfalls über das import Statement importiert werden.

```
import {File} from 'java.io';
```

Beispiel 11.33: `Import von Java-Klassen`

Es ist auch möglich, mit der `require`-Funktion `npm`-Module zu importieren. Diese werden im nächsten Abschnitt beschrieben.

npm-Module

`npm` ist ein Paketmanager für JavaScript der über 350.000 Pakete zur Verfügung stellt. Unter der Webseite https://www.npmjs.com/ können die vorhandenen Pakete durchsucht werden. Es ist möglich, in einem QF-Test Skript, installierte `npm`-Module zu verwenden. Diese müssen im javascript-Verzeichnis des QF-Test Wurzelverzeichnisses installiert werden. Mit dem Kommando `npm install underscore` wird das `npm`-Modul `underscore` über die Konsole des Betriebssystems installiert. Dieses kann nun in den Skript-Knoten verwendet werden.

Es gibt `npm`-Module, die nicht mit Nashorn kompatibel sind. Da beispielsweise einige Funktionen verwendet werden, die nicht vom ECMAScript Standard spezifiziert werden

```
_ = require('underscore');
              func = function(num){ return num % 2 == 0; }
              let evens = _.filter([1, 2, 3, 4, 5, 6], func);
              console.log(evens);
```

Beispiel 11.34: Verwendung des underscore-Moduls

11.6.2 Ausgaben

Neben `console.log()` wurde für Ausgaben ins Terminal in QF-Test eine zusätzliche print-Methode definiert.

```
print([1,2,3,4]);
```

Beispiel 11.35: Ausgabe eines Arrays

11.6.3 Ausführung

Die JavaScript-Skripte werden auf Server- bzw. SUT-Seite nicht im Browser ausgeführt, sondern in der Nashorn-Engine. Dies ermöglicht die Ausführung von ECMAScript in der JVM.

Kapitel 12

Unit-Tests

Unit-Tests oder Komponententests dienen der Überprüfung von funktionalen Einheiten. Sie sollen gezielt die Funktionalität der Komponenten testen, isoliert von anderen Komponenten. Aus diesem Grund besitzen sie eine deutlich geringere Komplexität im Vergleich zu Integrations- bzw. Systemtests, welche erheblich mehr Entwicklungsaufwand erfordern.

Mit dem Unit-Test(896) Knoten können Unit-Tests mit Hilfe des JUnit-Frameworks als Teil eines QF-Test Testlaufs ausgeführt werden. Sowohl der Report als auch das Protokoll zeigen deren Ergebnisse an. Die Einbindung von QF-Test Testsuiten in bestehende JUnit-Tests wird in Abschnitt 29.5(407) beschrieben.

Der Unit-Test Knoten kann Tests aus zwei unterschiedlichen Quellen starten. Hierbei werden die Parameter des Knotens dynamisch an den Anwendungsfall angepasst. Es können Java-Klassen angegeben werden, die JUnit-Testfälle enthalten, oder Unit-Test-Skripte mit QF-Test geschrieben werden. Dieses Kapitel erläutert die unterschiedlichen Möglichkeiten.

Für die Ausführung der JUnit-Tests wird das JUnit 5 Framework verwendet. Dieses ermöglicht es einerseits JUnit 5 Tests mit Hilfe der JUnit Jupiter Engine auszuführen. Andererseits kann die JUnit Vintage Engine sowohl JUnit 4 als auch JUnit 3 Tests ausführen. Mit JUnit 5 ist es möglich parametrisierte Tests zu schreiben, Tests innerhalb von Klassen einzubetten und den Reportnamen des Tests zu ändern.

12.1 Java-Klassen als Quelle für Unit-Tests

Es ist möglich Unit-Tests aus geladenen Jar-Ordnern und Class-Dateien auszuführen. Es können aber auch Tests ausgeführt werden, die bereits mit dem Start des SUTs geladen wurden. QF-Test führt dabei die angegebenen Test-Klassen(900) als Testschritte aus. Das folgende Beispiel soll den Aufbau eines Unit-Tests mit Java-Klassen verdeutlichen.

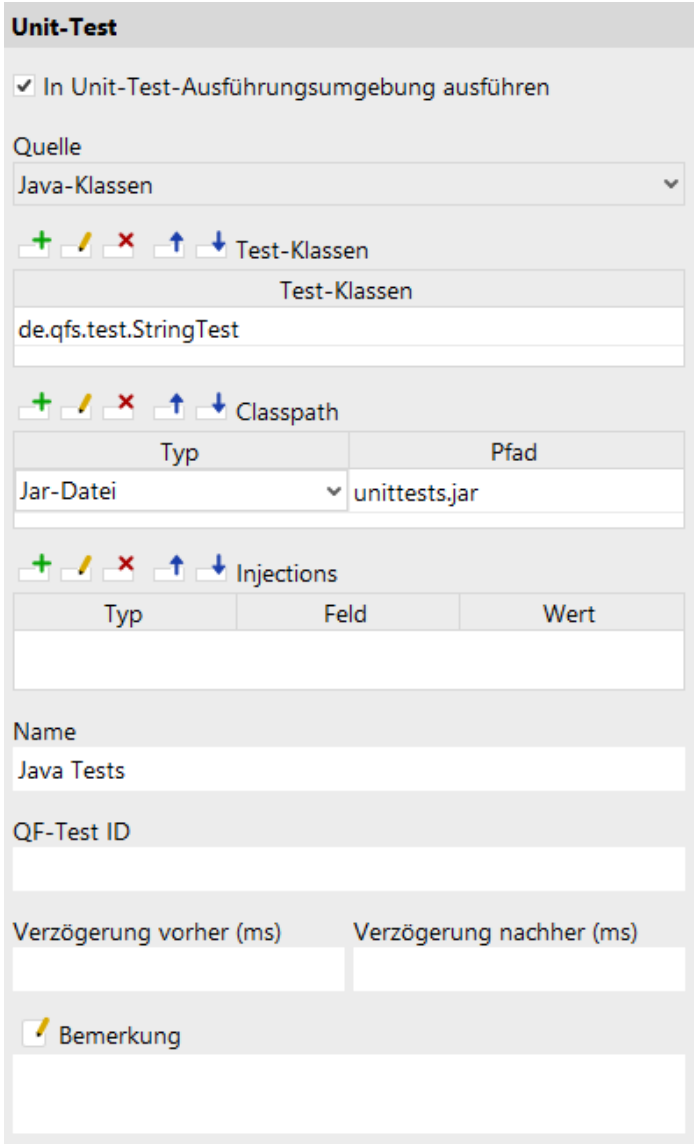


Abbildung 12.1: Unit-Test-Knoten mit Java-Klassen

```
package de.qfs.test;
import org.junit.Assert;
import org.junit.Test;
public class StringTest {
    @Test
    public void testubstring() {
        String s = new String("Langer Text");
        s = s.substring(7, 11);
        assert("Text".equals(s));
    }
    @Test
    public void testReplace() {
        String s = new String("Beispiel");
        s = s.replace('e', 'i');
        Assert.assertEquals("Biispiil", s);
    }
}
```

Beispiel 12.1: Code der Java-Unit-Test-Klasse

Die Klasse de.qfs.test.StringTest muss sich im unittests.jar befinden. Der Pfad zu dieser Jar-Datei wird unter Classpath(901) angegeben. Dieser Pfad wird relativ zum Verzeichnis der aktuellen Suite ermittelt. In diesem Beispiel liegt die jar-Datei direkt im Verzeichnis der Testsuite.

JUnit-Test-Klassen sind Java-Klassen, deren Methoden die `@Test`-Annotation besitzen. Der Unit-Test Knoten führt alle Klassen aus, die unter Test-Klassen(900) angegeben sind. Deshalb kann ein Unit-Test Knoten auch mehrere Test-Klassen ausführen.

12.2 Grundlagen der Test-Skripte

Die zweite Möglichkeit, die Unit-Test Knoten bieten, ist den Unit-Test direkt im Knoten zu kodieren. Hierfür können die in QF-Test vorhanden Skriptsprachen genutzt werden. Groovy ist dazu am besten geeignet, denn es ermöglicht die Verwendung von Java-Annotationen. Das Framework hierfür ist JUnit.

12.2.1 Groovy Unit-Tests

```
@BeforeClass
static void onbefore(){
    println("Vorbereitung")
}
@Test(expected=IndexOutOfBoundsException.class)
void indexOutOfBoundsAccess() {
    def numbers = [1,2,3,4]
    numbers.get(4)
}
@Test
void noFailure() {
    assert true
}
```

Beispiel 12.2: Unit-Test-Skript mit Groovy

In Groovy werden die für JUnit 4 erforderlichen Klassen automatisch importiert. Und wie in Java werden alle Tests mit der `@Test`-Annotation ausgeführt. Der `expected` Parameter der `@Test`-Annotation erlaubt es, erwartete Exceptions zu ignorieren. Die Methode, welche mit der `@BeforeClass`-Annotation gekennzeichnet ist, wird einmal vor der Ausführung der Test-Methoden ausgeführt.

12.2.2 Jython Unit-Tests

```
def setUp(self):
    print "Vorbereitung"
def testMathCeil(self):
    import math
    self.assertEqual(2, math.ceil(1.01))
    self.assertEqual(1, math.ceil(0.5))
    self.assertEqual(0, math.ceil(-0.5))
    self.assertEqual(-1, math.ceil(-1.1))
def testMultiplication(self):
    self.assertAlmostEqual(0.3, 0.1 * 3)
```

Beispiel 12.3: Unit-Test Skript mit Jython

Da in Jython keine Java-Annotationen verwendet werden können, werden die Skripte als JUnit-3-Tests ausgeführt. Alle Funktionen, die mit dem Schlüsselwort `test` beginnen, werden als Checks ausgeführt. Die Methoden müssen den Parameter `self` besitzen,

da diese von einer Klasse umschlossen werden. Die `setUp`-Methode wird vor Beginn der Tests ausgeführt.

12.2.3 JavaScript Unit-Test

```
setUp(){
    print("Vorbereitung");
}
tearDown() {
    print("Nachbereitung");
}
testUpperCase(){
    let s = "text";
    assertEquals("TEXT", s.toUpperCase());
}
testOk() {
    assertTrue(true);
}
```

Beispiel 12.4: Unit-Test Skript mit JavaScript

Da auch JavaScript keine Java-Annotationen unterstützt, können nur JUnit-3 Tests (vgl. Abschnitt 12.2.2(218)) ausgeführt werden. Wie in Jython werden alle Funktionen, die mit dem Schlüsselwort `test` beginnen, als Checks ausgeführt.

12.3 Injections

Es ist möglich, den Unit-Test(896) Knoten für sogenannte 'LiveTests' zu nutzen. QF-Test führt die Unit-Tests hierbei in einem laufenden SUT aus. Um in den JUnit-Test-Klassen Objekte wie Komponenten, QF-Test Variablen oder WebDriver-Objekte verwenden zu können, müssen diese in den Code 'injiziert' werden.

12.3.1 Komponenten in den Unit-Tests verwenden

```
import static org.junit.Assert.*;
import javax.swing.JComponent;
import org.junit.Test;
public class ComponentTest
{
    /** The component to test in this unit test */
    static JComponent component;
    /** Expected value */
    static String accessibleName;
    @Test
    public void accessibleNameIsCorrect()
    {
        /** component and accessible name are injected at run-time */
        final String currentName =
            component.getAccessibleContext().getAccessibleName();
        assertEquals(accessibleName,currentName);
    }
}
```

Beispiel 12.5: Java-Unit-Test-Klasse

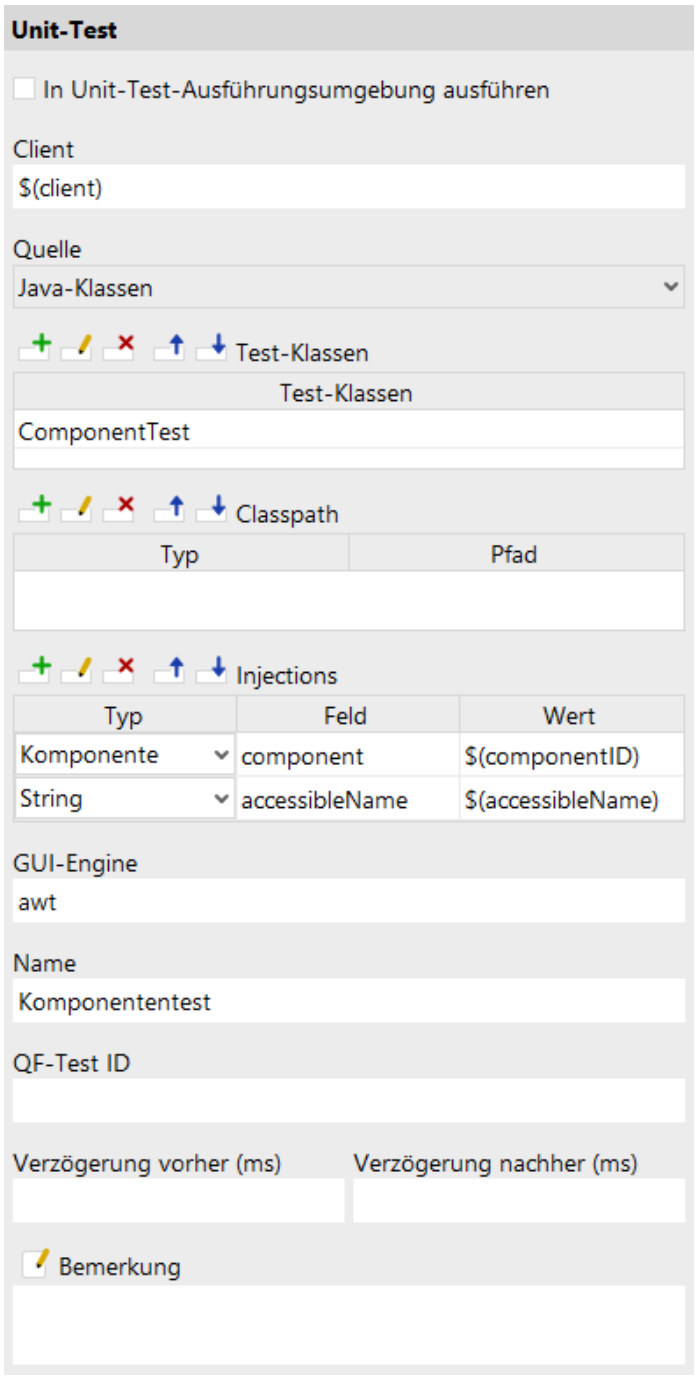


Abbildung 12.2: Beispiel eines Unit-Test Knotens mit Injections

In diesem Beispiel werden gleich zwei Objekte in die Unit-Tests übertragen. Eine Komponente und eine QF-Test Variable. Der Parameter 'Feld' der Injection entspricht hier dem Namen des Felds `static JComponent component;` der Java-Klasse.

Das Java-Feld muss hierfür `static` sein.

12.3.2 WebDriver-Injections

```
import static org.junit.Assert.*;
import org.junit.Test;
import org.openqa.selenium.WebDriver;
public class WebdriverTest
{
    /** The driver of the window currently opened by QF-Test. */
    static WebDriver driver;
    @Test
    public void urlIsCorrectedLoaded()
    {
        // driver is injected at run-time
        final String currentUrl = driver.getCurrentUrl();
        assertEquals("http://www.example.com", currentUrl);
    }
}
```

Beispiel 12.6: Java-Unit-Test mit WebDriver-Injections

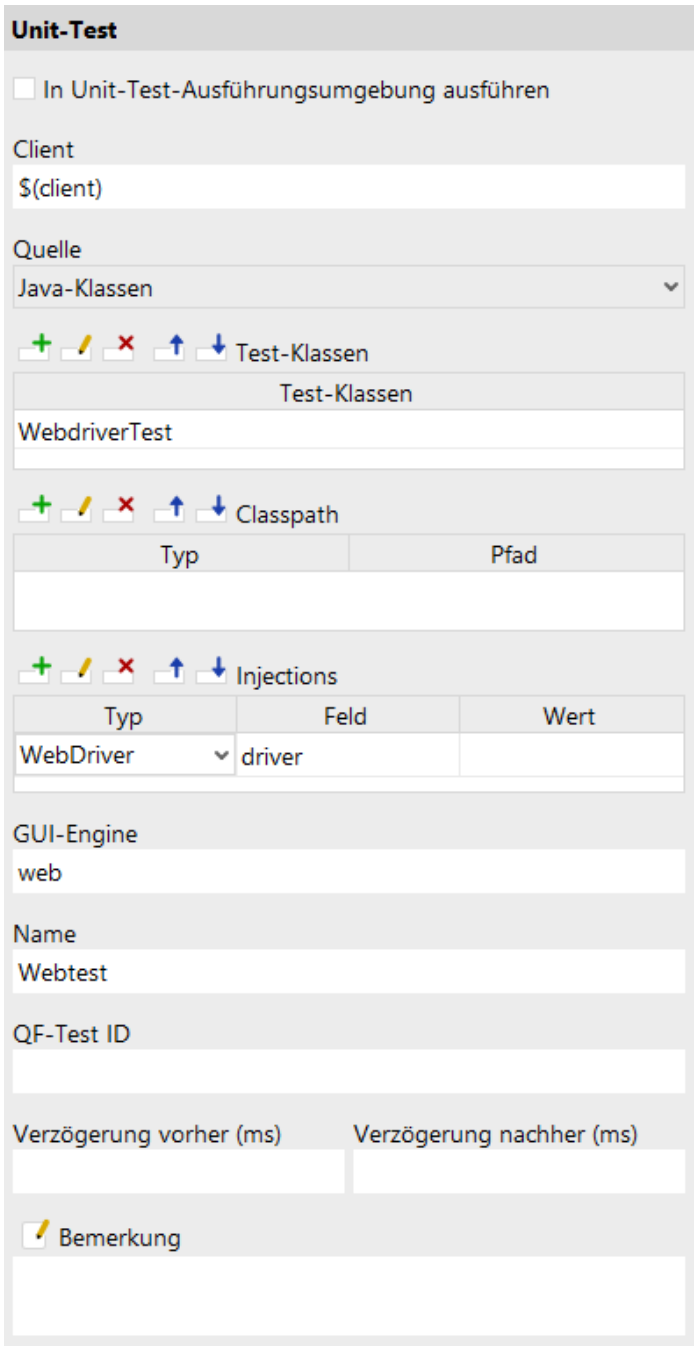


Abbildung 12.3: Beispiel eines 'Unit-Test'-Knotens mit WebDriver-Injections

Dieses Beispiel zeigt wie ein WebDriver-Objekt in die Java-Klasse übertragen wird. QF-Test ermittelt den Wert der WebDriver-Injection anhand des angegebenen Clients, falls kein expliziter Wert gesetzt ist.

12.4 Unit-Tests im Report

Der große Vorteil des Unit-Test(896) Knotens ist, dass dieser im HTML-Report erscheint. Die Unit-Tests werden als Testfälle interpretiert. Damit diese korrekt im Report erscheinen, muss ein Testfall ausgeführt werden, in welchem der Unit-Test-Knoten enthalten ist.

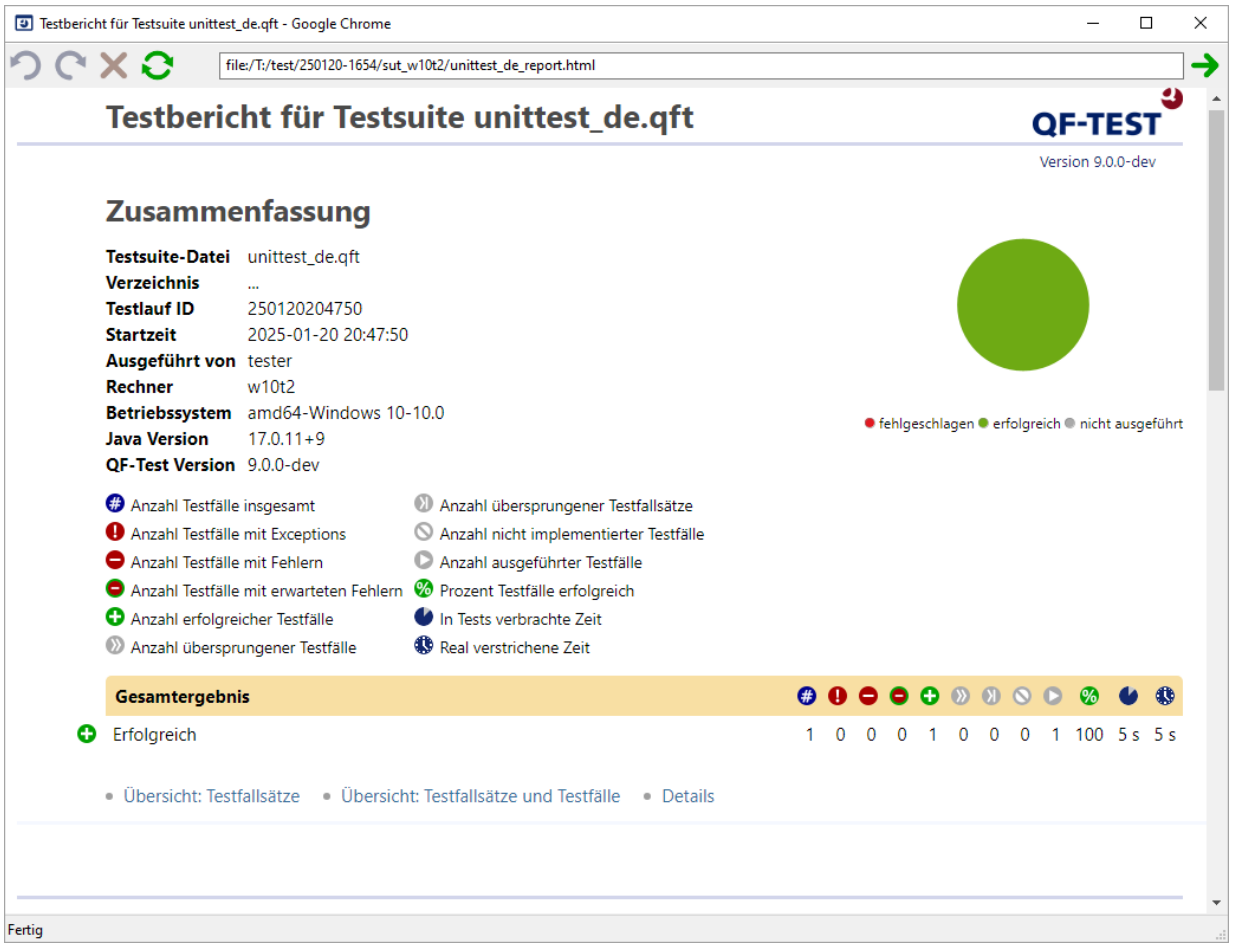


Abbildung 12.4: Unit-Test Report

Kapitel 13

Testen von Java Desktop-Anwendungen

Der Ursprung von QF-Test liegt im Testen von Java-basierten Desktop-Anwendungen und seit dem Start 1999 konnten wir die Unterstützung in zahlreichen Projekten perfektionieren für die bekannten Java GUI-Toolkits, als da wären:

- Java Swing - das GUI Toolkit von Sun/Oracle
- SWT (Standard Widget Toolkit) - das Tookit hinter Eclipse, entwickelt von IBM
- JavaFX - der gedachte Nachfolger von Java Swing von Oracle

Es gibt für diese Toolkits auch Erweiterungen/Bibliotheken, die spezielle Komponenten oder Framework-Funktionalitäten bereitstellen:

- Rich Client Platform (RCP)
- Eclipse Plug-Ins
- Netbeans Platform
- JFace GUI toolkit (auf SWT basierende Bibliothek)
- JIDE Common Layer Komponenten
- ULC (UltraLightClient) und RIA (Rich Internet Application)
- Java WebStart
- ...

Desktop-Anwendungen, die auf den genannten Technologien basieren, können mit QF-Test in einfacher und effizienter Weise getestet werden. Es entstehen robuste und verlässliche Testfälle mit geringem Wartungsaufwand, die einen hohen Wert für die Software-Qualitätssicherung darstellen. Alle allgemeinen Techniken, die in diesem Handbuch beschrieben sind, können auf das Testen von Java Desktop-Programmen angewendet werden.

Es gibt auch hybride Systeme, wie Java Desktop-Anwendungen mit einer eingebetteten Browser-Komponente oder solche, die in einem Browser als angezeigt/gerendert werden. Auch diese Art von Systemen werden von QF-Test perfekt unterstützt. Weitere Details findet man in den Kapiteln Testen von Webseiten(227) und Testen von Java Desktop-Anwendungen im Browser mit Webswing oder JPro(307).

Video Es gibt kurze Einführungsvideos zu

Java Swing Testen
https://www.qftest.com/de/yt/java-swing-testing.html

und

JavaFX Testen
https://www.qftest.com/de/yt/javafx-testing.html

auf unserem QF-Test YouTube Kanal.

Kapitel 14

Testen von Webseiten

QF-Test erlaubt das intuitive Testen von Webseiten im Browser aus Sicht des Benutzers. Wie auch bei den anderen unterstützten GUI-Technologien können Aktionen und Checks direkt aufgenommen, nachbearbeitet, strukturiert und wiedergegeben werden.

Video

Zum Einstieg gibt es auch ein kurzes

Überblicksvideo zu Web-Testen
https://www.qftest.com/de/yt/web-testen.html

auf unserem QF-Test YouTube Kanal.

14.1 Unterstützte Browser

QF-Test unterstützt die Automatisierung von Tests die folgenden Browser:

- Google Chrome (auch im Headless-Modus, siehe Abschnitt 14.7[232])
- Mozilla Firefox (auch im Headless-Modus, siehe Abschnitt 14.7[232])
- Microsoft Edge (auch im Headless-Modus, siehe Abschnitt 14.7[232])
- Safari
- Opera
- JxBrowser eingebettet in Swing, JavaFX oder SWT
- WebView eingebettet in JavaFX
- Internet Explorer oder Webkit eingebettet in SWT

Details zu den unterstützten Browser Versionen finden Sie in Abschnitt 1.1.3[4].

14.2 Allgemeine Vorgehensweise

Eine Schritt-für-Schritt Anleitung zum Einstieg das Web-Testen mit QF-Test finden Sie im entsprechende zweiten Teil des Tutorials.

Video

Es gibt Teile des Tutorials auch als Videos. Nur zur Erzeugung einer passenden Web-Startsequenz empfehlen wir das Video

Der Schnellstart Assistent Web
https://www.qftest.com/de/yt/schnellstart-assistent-web-42.html

auf unserem QF-Test YouTube Kanal.

Die Vorgehensweise für die Testerstellung und -ausführung bei Web-Anwendungen unterscheidet sich nicht wesentlich von der anderer GUI-Technologien, wie dies allgemein ab Kapitel 2(15) dieses Handbuchs beschrieben wird. Jedoch sollte ein besonderer Augenmerk auf die Komponentenerkennung gelegt werden, die stark abhängig von der konkreten Implementierung der Web-Anwendung ist. Um herauszufinden, wie gut die direkte Wiedererkennung funktioniert, empfiehlt es sich zu Beginn testweise Aktionen auf unterschiedliche Komponenten und Masken der Web-Anwendung aufzunehmen und das korrekte Abspielen zu überprüfen. Weitere Hinweise zur Komponentenerkennung und Möglichkeiten der Optimierung finden Sie in Abschnitt 14.4(229).

14.3 Die Verbindung zum Browser

Im ersten Schritt muss der gewünschte Browser von QF-Test gestartet und eine Verbindung hergestellt werden. Sobald die per URL angegebene Web-Anwendung geladen ist, kann mit der Aufnahme und Testerstellung begonnen werden.

QF-Test verwendet unterschiedliche Methoden, so genannte Driver, um Zugriff auf den Browser zu erlangen und eine Verbindung herzustellen: den **QF-Driver**, den **CDP-Driver** und den **WebDriver**.

Hinweis

Je nach Browser stehen unterschiedliche Verbindungsmodi zur Verfügung. QF-Test versucht, automatisch den besten Modus zu wählen. Sie können die Kontrolle darüber mit Hilfe des Attributs Verbindungsmodus für den Browser(738) im Web-Engine starten(737) Knoten übernehmen. Details hierzu finden Sie in Abschnitt 51.3(1128).

Beim **QF-Driver** wird der auf dem Rechner des Anwenders installierte Browser in ein so genanntes Wrapper-Fenster eingebunden. Man spricht bei diesem Ansatz auch von Embedding. QF-Test bindet in dieses Fenster den lokal installierten Web-Browser nativ ein und erlangt somit Zugriff auf die Automatisierungs-Interfaces des jeweiligen Browsers. Über diese kann QF-Test dann die benötigten Events vom Browser abgreifen aber auch umgekehrt Events in den Browser einspeisen.

Die Einbettung des Browsers in ein separates Fenster funktioniert für neuere Browser leider nicht mehr zufriedenstellend oder gar nicht, weshalb alternative Mechanismen zur Verfügung stehen.

5.3+ Der **CDP-Driver** Mechanismus greift zur Kommunikation auf die in Chromium (und davon abgeleiteten Browsern) eingebaute Debugging-Schnittstelle zurück. Dazu verwendet QF-Test das Chrome DevTools Protokoll. Da dies die gleiche Schnittstelle ist, welche auch von den Entwicklungstools im Browser verwendet wird, ist eine enge und effiziente Testausführung möglich. Leider steht das Protokoll noch nicht für alle von QF-Test unterstützen Browser zur Verfügung.

4.1+ Der **WebDriver** Mechanismus verwendet als Bindeglied zwischen dem Browser und QF-Test den Selenium WebDriver, der sich zu einem W3C-Standard für die Steuerung von Web-Browsern entwickelt hat (http://www.w3.org/TR/webdriver/).

Hinweis Der WebDriver Modus ist systembedingt leider in Bezug auf Performance und Funktionsumfang im Vergleich zum QF-Driver etwas eingeschränkt (siehe Abschnitt 51.3.4(1130)). Wenn möglich sollte die Testerstellung mittels QF-Driver oder CDP-Driver, also z.B. mit dem Chrome Browser durchgeführt werden. Die reine Ausführung kann dann auch mittels WebDriver erfolgen - leider nicht ganz so performant.

Hinweis Um die Aufnahmefunktionalitäten von QF-Test auch im WebDriver Modus verwenden zu können, muss QF-Test die Inhalte des Browser-Fensters kontinuierlich analysieren und auf Änderungen überwachen. Dies kann in einigen Browsern eine Warnung hervorrufen, die sich auf Mixed-Mode-Darstellung bzw. die Verwendung eines unbekannten Zertifikats bezieht. Sollte diese Warnung bzw. Fehlermeldung nur während des Tests Ihrer Website mit QF-Test auftreten, so kann sie ignoriert werden.

14.4 Erkennung von Web-Komponenten und Toolkits

Bei Web-Applikationen haben Entwickler viele Freiheitsgrade, wie sie graphische Objekte in HTML entwickeln. Dies resultiert in einer Vielzahl von Implementierungen für funktionale GUI-Komponenten wie Buttons, Textfelder, Tabellen etc. Hier ein paar Beispiele für die Umsetzung eines Okay Buttons:

1. `<button id="ok1">OK</button>`
2. `<div class="toolkit-btn">OK</div>`
3. `<span class="btn" name="ok2">OK</span>`
4. `<div role="button">OK</div>`

Zunächst nimmt QF-Test die GUI-Elemente mit den HTML-Tags als Klassen sowie die Basiswiedererkennungsmerkmale auf.

Im ersten Beispiel würde somit eine QF-Test Komponente mit der Klasse `BUTTON`, dem Namen `ok1` und den für die Anzeige gültigen Struktur- und Geometrieinformationen angelegt. Beim Abspielen der Tests sollte die Komponente somit zuverlässig erkannt werden.

Darüber hinaus prüft QF-Test, ob es sich bei den HTML-Angaben um Quasi-Standards handelt, die einer generischen QF-Test Klasse zugeordnet werden können. Dies wäre in diesem Beispiel der Fall und es würde `Button` zugewiesen.

Der Vorteil der generischen Klassen ist, dass klassenabhängig zusätzliche Wiedererkennungsmerkmale aufgenommen werden. So wird einem Button zum Beispiel die Beschriftung als Merkmal Attribut zugewiesen. Des weiteren stehen bei der Aufnahme von Checks zusätzliche klassenspezifische Checks zur Verfügung. Bei Tabellen wird zum Beispiel die Aufnahme eines Checks für eine ganze Zeile oder Spalte angeboten. Weitere Vorteile sind in der Einleitung zu Generische Klassen(1329) beschrieben. Detaillierte Informationen zu den klassenspezifischen Erweiterungen finden Sie in Generische Klassen(1329).

Auf dem Markt existieren eine Vielzahl von Komponentenbibliotheken, wie Angular Material oder Vaadin, die die Erstellung von Webseiten massiv erleichtern. Jede dieser Bibliotheken hat ihre eigenen Implementierungen für GUI-Objekte.

Im zweiten Beispiel verwendet das Toolkit immer die css-Klasse `toolkit-btn` für Buttons.

Für eine Reihe von Toolkits ist in QF-Test die Zuordnung der GUI-Elemente zu Komponenten einer bestimmten generischen Klassen bereits hinterlegt. In diesen Fällen können Sie mit der gewohnt stabilen Komponentenerkennung von QF-Test arbeiten. Weitere Informationen zu den unterstützten Web-Toolkits finden Sie in Besondere Unterstützung für verschiedene Web-Komponentenbibliotheken(1122). Im Normalfall erkennt QF-Test selbständig, ob und mit welchem Toolkit die Web-Anwendung erstellt wurde. Alternativ können Sie manuell aus den verfügbaren Web-Toolkits auswählen.

Das dritte und vierte Beispiel entsprechen keinerlei Standards. Die Wiedererkennung im dritten Beispiel würde über das `name` Attribut vermutlich ausreichend stabil sein. Im vierten Beispiel bietet das GUI-Objekt von sich aus aber gar keine guten Merkmale für die Wiedererkennung. In beiden Fällen kann jedoch das kennzeichnende HTML-Attribut der generischen Klasse `Button` zugewiesen werden. Details hierzu finden Sie in Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077).

Hinweis

Es ist sinnvoll vor dem Erstellen der Tests zu prüfen, zu welcher dieser Kategorien die GUI-Objekte der zu testenden Applikation gehören. Falls die Standarderkennung nicht ausreichend ist, sollte durch Zuordnung von GUI-Objekten zu generischen QF-Test Klassen eine Optimierung erfolgen. Siehe Abschnitt 51.1(1077).

Allgemeine Informationen Komponenten in QF-Test finden Sie in Kapitel 5(47).

Hinweis Der Umgang mit mehreren Browser-Fenstern wird in FAQ 25 genauer erläutert.

14.5 Cross-Browser Tests

Browser-übergreifende Tests sind einfach realisierbar. Sie erstellen Testfälle für einen Browser und können diese dann auf anderen Browsern abspielen. Im Wesentlichen muss nur in einem Datentreiber(646) die Variable `$(browser)`, die den Browser definiert, passend gesetzt werden. Falls Sie es probieren wollen, fügen Sie einfach in der mitgelieferten Demo-Testsuite für Web im Testfallsatz "CarConfiguratorWeb Demo" einen Datentreiber wie folgt ein: (Um die Demo-Testsuite zu öffnen, wählen Sie den Menüpunkt Hilfe→Beispiel-Testsuiten erkunden... und klicken den "öffnen"-Link hinter "Web CarConfig Suite".)

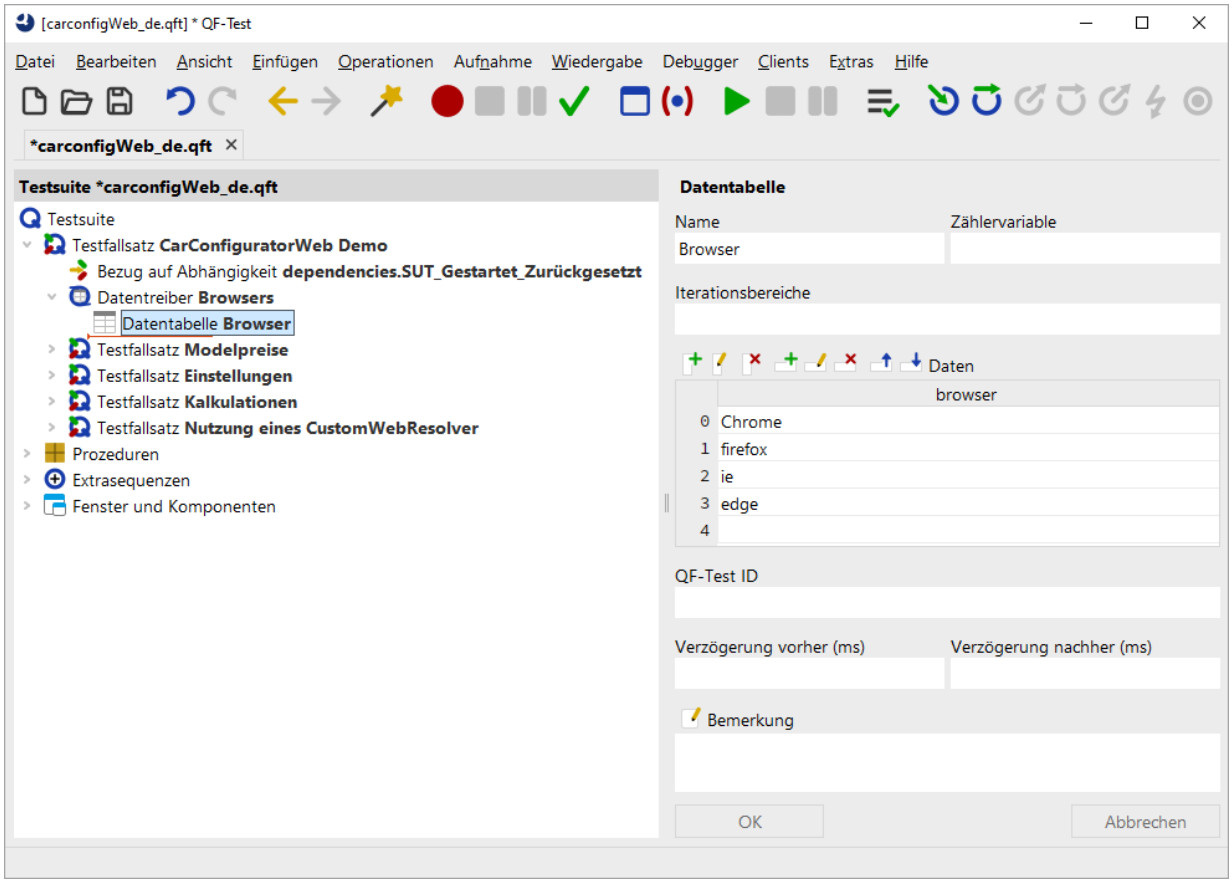


Abbildung 14.1: Cross-Browser Tests

Dann werden die vier Testfallsätze, die gemeinsam mit dem Datentreiber im Testfallsatz

”CarConfiguratorWeb Demo” liegen, für jeden Browser-Typ einmal durchlaufen.

14.6 Testen von mobilen Webseiten

4.2.1+ Beim Testen von Webseiten ist häufig das Benutzererleben auf mobilen Geräten wie Smartphones oder Tablets relevant, da die Webseite aufgrund einer unterschiedlichen Browser-Kennung (”User Agent”) und spezifischen Display-Größen anders dargestellt wird als im Desktop-Browser (”Responsive Design”).

QF-Test unterstützt solche Szenarien über die Emulation mobiler Browser. Dabei wird ein Desktop-Browser in einem Modus gestartet, in welchem die Fenstergrößen und die Browserkennung den Browser eines Mobilgerätes emuliert.

Insbesondere Google Chrome ist in der Lage, spezifische Charakteristika der mobilen Variante, wie ein angepasstes Pixel-Verhältnis bzw. eine automatische Skalierung nicht-responsiver Webseiten nachzubilden.

Um mit so einem Szenario zu beginnen, können Sie im Schnellstart-Assistenten die Kategorie ”Eine Web-Anwendung in einem emulierten mobilen Browser” nutzen und dann das gewünschte Mobilgerät auswählen (siehe Kapitel 3(32)).

Ein Demo finden Sie in der Testsuite ”carconfigWeb_advanced_de.qft”. Dort die Testsuite ”Emulation von mobilen Geräten”. Dies können einfach öffnen über das Menü Hilfe→Beispiel-Testsuiten erkunden... und dann Auswahl des letzten Beispiels ”CarConfig Web Testprojekt”.

14.7 Web-Testen im Headless-Modus

4.2+ Mit Hilfe der CDP-Driver und WebDriver Ansätze ist es auch möglich, Chrome, Firefox und Microsoft Edge im sogenannten Headless-Modus anzusteuern. Dabei wird der Browser im Hintergrund gestartet, ohne ein sichtbares Fenster auf dem Bildschirm anzuzeigen. Alle Interaktionen mit der Webseite erfolgen im ”unsichtbaren” Fenster.

Anwendungsfälle für Headless-Browser können Web-Lasttests sein (Abschnitt 33.5(449)) oder Tests, die parallel zur Testentwicklung auf dem gleichen Rechner im Hintergrund laufen sollen.

Um einen bestehenden Web-Test im Headless-Modus auszuführen, muss lediglich der Typ des Browsers im Knoten Web-Engine starten(737) von `chrome` auf `headless-chrome` bzw. von `firefox` auf `headless-firefox` oder von `edge` auf `headless-edge`geändert werden.

14.8 Einbindung vorhandener Selenium Web-Tests

QF-Test erlaubt bei Verwendung des WebDriver Modus die Kombination mit bereits bestehenden Selenium Skripten.

Ein Weg ist die direkte Nutzung der WebDriver Java-APIs in SUT-Skript(720) Knoten (vgl. Abschnitt 54.11(1269)).

Eine andere Möglichkeit ist Einbettung von Selenium Skripten als Unit-Tests, wie in Kapitel 12(215) beschrieben. Als schönen Nebeneffekt erhält man ein integriertes Reporting, bei dem auch die ausgeführten Unit-Tests enthalten sind.

Ein Demo finden Sie in der Testsuite "carconfigWeb_advanced_de.qft" - dort die Testsuite "Integration von Selenium Tests". (Einfach zu öffnen über das Menü Hilfe→Beispiel-Testsuiten erkunden... und dann Auswahl des letzten Beispiels "CarConfig Web Testprojekt".)

14.9 Auswahl der Browser Installation

Bei der Verwendung des CDP-Driver oder WebDriver Modus kann man im Web-Engine starten(737) auch für andere Browser als Firefox explizit das Verzeichnis der Browser-Installation(738) setzen. Ist kein Verzeichnis angegeben, so wird versucht, einen Standard-Browser des entsprechenden Typs zu starten.

Kapitel 15

Testen nativer Windows-Anwendungen

5.0+

15.1 Einstieg

Video

Video über das Testen von Native Windows Desktop Applicationen:

'QF-Test Version 5.0 - Windows Anwendungen testen'
https://www.qftest.com/de/yt/version-50-windows-anwendungen-testen-50.html

Dieses Kapitel behandelt das automatisierte Testen von Windows-Desktop-Anwendungen, insbesondere

- klassische Win32-Anwendungen,
- .NET-Anwendungen, welche auf den Frameworks Windows Presentation Foundation (WPF) oder Windows Forms basieren und
- Universal Windows Platform (UWP) Anwendungen, welche XAML-Steuerelemente verwenden.

Die genannten Anwendungstypen unterstützen die Microsoft UI Automation oder die Microsoft Active Accessibility (MSAA) Schnittstellen. Weiterführende Informationen zu diesen, unter dem Namen Windows Automation API zusammengefassten Schnittstellen, wurden im Abschnitt 15.2(235) zusammengetragen.

Für die Testausführung benötigt QF-Test eine Verbindung zu dem Prozess der zu testenden Anwendung. Zur Erstellung einer Vorbereitung, welche eine solche Verbindung herstellt, bietet sich der Erzeugung der Startsequenz - Schnellstart-Assistent(33) an, welcher über das Menü Extras aufgerufen werden kann. Wählen Sie 'Eine native Windows-Anwendung' als Anwendungstyp. Weiterführende

Informationen zur Benutzung des Schnellstart-Assistenten können Sie Abschnitt 15.3[236] und Abschnitt 3.1[33] entnehmen.

Um sich mit einer bereits laufenden Anwendung zu verbinden, können Sie den Windows-Anwendung verbinden Knoten verwenden. Hierbei reicht es den Titel des (Haupt-)Fensters anzugeben. Hierbei können Sie auch reguläre Ausdrücke verwenden. Falls dies der Fall ist, müssen Sie zusätzlich die CheckBox Als Regexp aktivieren. Für den Windows Editor (Notepad) wäre dies zum Beispiel `.*- Editor`. Hierbei sollte allerdings darauf geachtet werden, dass der angegebene reguläre Ausdruck nicht zufällig mit dem Titel eines anderes Fenster übereinstimmt. Wenn die Anwendung durch QF-Test gestartet werden soll, verwenden Sie bitte den Windows-Anwendung starten Knoten und geben Sie die ausführbare Datei (.exe) Ihrer Anwendung inklusive Pfad an, vgl. Start/Anbindung einer Applikation[236].

Wenn sich QF-Test zu der Anwendung (als GUI-Engine[722] `win`) verbunden hat, können Tests aufgenommen und abgespielt werden wie in Kapitel 4[39] beschrieben. Auf Grund der Eigenschaften der Microsoft UI Automation sind allerdings die in Abschnitt 15.4[237] aufgeführten Aufnahmeregeln zu beachten.

Die Installation von QF-Test liefert folgende Beispiel-Testsuiten mit:

- `qftest-9.0.0/demo/carconfigForms/winDemoForms_de.qft`
- `qftest-9.0.0/demo/carconfigWpf/winDemoWPF_de.qft`
- `qftest-9.0.0/demo/windows/Win10Calculator_de.qft`

Bitte beachten Sie auch die (aktuellen) Einschränkungen, Abschnitt 15.9[243], von denen zu erwarten ist, dass die meisten in zukünftigen QF-Test Releases behoben oder verbessert werden.

15.2 Technischer Hintergrund

Ein verbreitetes Framework für Windows basierte Applikationen ist die Windows Automation API, bestehend aus der Microsoft Active Accessibility und dessen Nachfolger, der Microsoft UI Automation. Dies ist das Kernstück der `win` Engine, wodurch QF-Test in der Lage ist, fast alle Arten von Windows Applikationen zu steuern.

Eine Windows-Anwendung muss sogenannte `Provider` zur Verfügung stellen, um die Regeln der UI Automation einzuhalten. Dies geschieht bei der Verwendung eines Frameworks wie WPF automatisch, bei Win32 Applikationen über Proxy Provider. Wie gut eine Applikation getestet werden kann, hängt somit von der Qualität der jeweiligen `Provider` ab, d.h. dem Framework, welches für die Entwicklung verwendet wurde. Da die UI Automation zusammen mit dem WPF Framework eingeführt wurde, sollten

Anwendungen, die damit entwickelt wurden, gut testbar sein. Bei Entwicklungsplattformen ohne Integration der UI-Automation sieht die Sache anders aus, wie zum Beispiel bei Java Swing. Aber dafür für Java-Anwendungen stellt QF-Test auch einen ziemlich guten anderen Verbindungsmodus bereit ...

Wenn ein Programm über UI Automation getestet werden soll, so stehen sogenannte `Automation Elements` zur Verfügung, die die eigentlichen UI Elemente der zu testenden Anwendung darstellen. Obwohl jedes `Automation Element` einen `Control Type` hat (`Button`, `MenuItem` etc.), wird seine tatsächliche Funktionalität - zum Beispiel das Setzen eines Wertes in einem Textfeld - über `Control Patterns` bestimmt, die über den jeweiligen `Provider` implementiert sind.

Zur Bedienung des UI Automation Frameworks startet QF-Test ein spezielles Java Programm, das als UI Automation Client Applikation dient. Dieses Programm kann alle UI Automation Elemente eines bestimmten Prozesses ansprechen und diese entsprechend der QF-Test Regeln bedienen (zum Beispiel zum Erstellen eines Abbildes eines Elements als Komponente(930)).

15.3 Start/Anbindung einer Applikation

Um eine native Windows-Anwendung zu testen ist es nicht zwingend notwendig, diese Anwendung über QF-Test zu starten. Man kann QF-Test auch mit einem bereits laufenden Prozess verbinden. Dadurch ist es auch möglich, Bereiche des Betriebssystems zu steuern, zum Beispiel die Windows Taskbar.

Die Anbindung an einen bereits laufenden Prozess erfolgt entweder über den Fenstertitel (wahlweise mit regulärem Ausdruck), die Prozess-Id oder den Klassennamen des Fensters, der über die UI Automation bereitgestellt wird. Die Anbindung ist für ein `Window` im Sinne eines UI Automation `Control Types` möglich, aber auch für ein `Pane` oder `Menu` Element. Unabhängig von der gewählten Verbindungsmethode, ermittelt QF-Test die jeweilige Prozess-Id und wird genau diesen Prozess als System Under Test (SUT) behandeln.

Der Windows-Anwendung verbinden(747) Knoten stellt die Verbindung her. Dabei muss für die Verbindung zu einer bereits laufenden Anwendung im Attribut Fenstertitel(747) einer der folgenden Werte eingetragen werden (wie bereits oben erwähnt):

- ein regulärer Ausdruck für den Fenstertitel
- -pid `<Prozess-Id>`
- -class `<class name>`

Beispiele:

`.*- Editor`: zur Anbindung an einen laufenden Windows Notepad Prozess,

`-class Shell_TrayWnd`: Anbindung der Windows Taskbar verbinden.

Für die Ermittlung des Fenstertitels, der Prozess-Id oder des `Class Name` des laufenden Programms steht die Prozedur `qfs.autowin.logUIAToplevels` in der Standardbibliothek(183) `qfs.qft` zur Verfügung.

Man kann die zu testende Anwendung aber auch direkt über den Windows-Anwendung starten(743) Knoten starten. In diesem Fall geben Sie den Dateinamen der ausführbaren Datei, inklusive Pfad, im Attribut Windows-Anwendung(744) an.

Sie können auch das Attribut Windows-Anwendung zusammen mit dem Fenstertitel Attribut angeben. Dies ist nützlich, wenn die Anwendung gestartet werden soll, falls sie nicht bereits läuft. QF-Test prüft dann zuerst, ob es sich an einen laufenden Prozess, der dem Fenstertitel Attribut entspricht, anbinden kann. Falls nicht, wird das angegebene Programm gestartet und über seine Prozess-Id verbunden. Es kann vorkommen, dass dieser Prozess einen weiteren Prozess startet, der das eigentliche Programm darstellt, wobei ersterer selbst aber keine (grafische) Benutzeroberfläche hat oder sich sogar beendet. In diesem Fall wird ein weiterer Versuch durchgeführt um eine Verbindung zu dem zweiten Prozess zu erlangen.

Wenn Sie in QF-Test einen `win` Client beenden (entweder mittels des Programm beenden(769) Knotens oder über das Clients Menü), wird der entsprechende UI Automation Client Prozess mitsamt seiner Unterprozesse gestoppt. Das heißt, dass die zu testende Anwendung nur dann beendet wird, wenn sie aus QF-Test heraus gestartet wurde, nicht aber, wenn sie bereits vor der Verbindung mit QF-Test gelaufen ist.

Wenn sie die zu testende Anwendung beenden, wird in jedem Fall auch der UI Automation Client beendet.

Um sich mit einem heraufgestuften Prozess verbinden zu können, muss QF-Test als Administrator gestartet werden.

15.4 Aufnahme

Nach erfolgreicher Verbindung von QF-Test mit der zu testenden Anwendung können Sie mit der Aufnahme von Aktionen (Abschnitt 4.1(39)), Checks (Abschnitt 4.3(43)) und Komponenten (Abschnitt 4.5(44)) beginnen.

Da die Kommunikation zwischen dem QF-Test UI Automation Client und der zu testenden Anwendung über Windows (konkret, den UI Automation Core) läuft, ist der Zugriff auf die Komponenten nicht ganz so schnell wie Sie dies vielleicht von der QF-Test Java-Automatisierung kennen. Außerdem werden Events im Gegensatz zu Java- oder Web-

Tests (QF-Driver) asynchron verarbeitet. Das bedeutet, dass Sie nicht davon ausgehen können, dass der Dispatch Thread der Applikation geblockt wird während QF-Test ein Ereignis bearbeitet.

Dies macht Aufnahmen schwieriger, wenn die Zielkomponente in Folge der aufzunehmenden Aktion verschwindet bevor QF-Test die Komponentenidentifizierung abgeschlossen hat. Zum Beispiel bei der Aufnahme eines Combobox-Eintrags, der die Liste nach dem Klick sofort schließt, oder eines Buttons, der das Fenster, in dem er liegt, beendet.

Daher ist es sinnvoll, wenn Sie folgende Vorgehensweise beachten:

- Aktivieren Sie den Aufnahmemodus und bewegen Sie die Maus zu dem Element, für das Sie eine Aktion aufnehmen wollen.
- Da QF-Test etwas Zeit benötigt um das Element unter dem Mauszeiger zu identifizieren, wird ein rotes Panel angezeigt bis der Vorgang abgeschlossen ist. In dem kleinen 'QF-Test Element Information' Fenster sehen Sie dann, welche Komponente erkannt wurde.
- Führen Sie nun die aufzunehmende Aktion aus.
- Wenn Sie einen Mausklick aufnehmen, der einen Dialog oder ein Fenster schließt (auch Popup-Listen), halten Sie die Maustaste ein wenig länger gedrückt, so dass QF-Test die Möglichkeit hat, die nötigen Informationen auszulesen bevor das Fenster verschwindet, was häufig fast zeitgleich mit dem Loslassen der Maustaste geschieht.
- Bei der Aufnahme von Checks oder Komponenten erscheint ein Rahmen um das Element sobald der Mauszeiger darüber fährt. Bitte warten Sie mit der Aufnahme bis der Rahmen wieder verschwunden ist.

Wenn Sie bei der Aufnahme einer Aktion Probleme haben, weil diese zum Beispiel das entsprechende Fenster schließt (Klick auf OK oder Abbrechen Button), kann es hilfreich sein, einen Check auf die entsprechende Komponente aufzunehmen und diesen dann in die gewünschte Aktion zu konvertieren. Manchmal kann das Drücken der Maustaste auch dazu führen, dass ein Element neu generiert wird (zum Beispiel bei der Zubehörtabelle im CarConfiguratorNet WPF Demo). Auch hier kann die Check-Aufnahme helfen. Im Check-Aufnahmemodus legt QF-Test ein (fast) unsichtbares Fenster über die zu testende Anwendung um so zu verhindern, dass Mausklicks eine Aktion in der Anwendung bewirken.

15.5 Komponenten

Ein UI Automation Element wird von QF-Test als Fenster[919] beziehungsweise Komponente[930] innerhalb des Fenster und Komponenten[942] Knotens aufgenommen. Ein manuelles Einfügen von Knoten ist natürlich auch möglich. Die (generische) QF-Test Klasse entspricht häufig dem `Type` des UI Automation Elements, zum Beispiel bei `Button`. Um den `Type` vom Klassennamen unterscheiden zu können, setzt QF-Test das Präfix `Uia.` vor den `Type`. Analog wird der Kurzname des UI Automation Frameworks als Präfix für die UI Automation `Class` des Elements verwendet. Zum Beispiel würde QF-Test bei einer Tabelle einer WPF-Anwendung in Weitere Merkmale des Komponente Knotens einen Eintrag mit `classname: WPF.DataGrid` anlegen.

QF-Test bildet die Hierarchie der UI Automation Elemente unter Umständen nicht eins zu eins im Komponentenbaum ab. Dies geschieht häufig bei Dialogen (wie zum Beispiel dem Schriftart-Dialog von Notepad), die üblicherweise in der Komponentenhierarchie der UI Automation unter dem Hauptfenster aufgeführt werden. Aus der Sicht von Win32, genauso wie das auch QF-Test Anwender erwarten würden, sind diese Dialoge top-level Windows und damit in Fenster und Komponenten parallel zum Hauptfenster angeordnet. Andererseits kann ein Kontextmenü, das ganz oben in der UI Automation Hierarchie angesiedelt ist, in QF-Test innerhalb eines Fensters zu finden sein.

15.6 Wiedergabe und Patterns

Die `win` Engine unterstützt neben den bereits aus anderen Engines bekannten Mausklicks auch verschiedene Spezialaktionen die nicht auf Mausklicks basieren.

Zum Beispiel kann die Aktion einer Schaltfläche mittels

```
+Auswahl: invoke [myButtonID]
```

ausgelöst werden. Die Wirkung sollte die gleiche sein wie bei

```
+Mausklick [myButtonID]
```

Bei einem Auswahl wird die Maus nicht verwendet. Stattdessen löst der UI Automation Prozess die Ausführung einer `Invoke()` Methode des `Providers` im SUT aus.

Der Auswahl Knoten unterstützt die folgenden Aktionen im Attribut Detail: `invoke`, `expand`, `collapse`, `select`, `toggle` und `scroll`. In der nachfolgenden Tabelle werden sie detailiert beschrieben:

Detail	Beschreibung	Pattern
`invoke`	Meist gleichwertig mit einem Mausklick.	InvokePattern
`expand, collapse`	Dies sollte eine Combobox, ein MenuItem oder ein TreeItem aus- beziehungsweise einklappen.	ExpandCollapsePattern
`select[:0|-1|1]`	Wählt einen Listeneintrag aus. Wenn als Detail -1 bzw. 1 angegeben wird, so wird eine Mehrfachauswahl erweitert oder verringert.	SelectionItemPattern
`toggle[:on|off]`	Ändert den Zustand einer CheckBox.	TogglePattern
`scroll:horiz%,vert%`	Erlaubt Werte zwischen 0 und 100, die die Scroll-Position in Prozent angeben. -1, wenn die Position (Horizontal oder vertikal) nicht verändert werden soll.	ScrollPattern

Tabelle 15.1: Unterstützte Details für Auswahl

Welche Aktionen tatsächlich auf einem Control abgespielt werden können, hängt davon ab, welche Pattern das entsprechende UI Automation Elements implementiert. Die verfügbaren Patterns werden unter Weitere Merkmale einer Komponente abgespeichert oder können in einem SUT-Skript mit dem Befehl `print rc.getComponent(id).getPatterns()` ermittelt werden.

Die genaue Bedeutung eines Patterns kann sich von Anwendung zu Anwendung unterscheiden. Wenn sowohl das `SelectionItem`-Pattern als auch das `Invoke`-Pattern unterstützt werden, sollte `invoke` bevorzugt werden, da

```
+Select [list@item]
```

eventuell nur das Element markiert, nicht jedoch die entsprechende Aktion auslöst. (Ein Beispiel hierfür ist die Schriftartenauswahl bei Notepad.)

Die formale Unterstützung eines Patterns bedeutet leider noch lange nicht, dass der Aufruf des entsprechenden Auswahl-Events irgendeine Wirkung zeigt. Dies ist zum Beispiel in der Rechner-Applikation von Windows der Fall, wenn man zu einem (nicht sichtbaren) Eintrag der Modus-Auswahlliste blättern möchte (`ScrollItem pattern`). Um dieses Problem zu umgehen, kann in diesem Fall ein `select` abgespielt werden, unabhängig davon, ob der Eintrag gerade sichtbar ist.

Weil das ”weiche” Abspielen von Aktionen häufig im `provider` nicht implementiert ist funktionieren diese Spezialaktionen nur bedingt.

Bei Tastaturevents kann ein Text nur dann direkt über einen Texteingabe Knoten gesetzt

werden, wenn das `Value pattern` unterstützt wird. Ansonsten muss der Text über einzelne Tastaturevents eingegeben werden.

15.7 Skripting

Intern stellt die `win` Engine ein UI Automation Element mit der Klasse `WinControl` dar. Um auf ein Element in einem Groovy SUT-Skript[720] Knoten zuzugreifen, führen Sie folgendes Skript mit der passenden QF-Test ID der Komponente aus:

```
def winc = rc.getComponent("myComponentID")
println winc
```

Beispiel 15.1: Zugriff auf ein `WinControl` in einem Groovy-SUT-Skript

Die Methoden der Klasse `WinControl` sind in Abschnitt 54.12.1[1271] detailliert beschrieben:

- `getUiaType(), getUiaClassName(), getFramework(), getUiaName(), getUiaId(), getUiaDescription(), getUiaHelp(), getHwnd(), getLocation(), getSize(), getLocationOnScreen(), getPatterns(), hasPattern()` um die UI Automation Eigenschaften eines Elements zu erhalten
- `getChildren(), getParent(), getChildrenOfType(), getAncestorOfType(), getElementsByClassName()` für die Elementhierarchie
- `getUiaControl()` um ein `AutomationBase` zu erhalten, welches dann mit der `uiauto`-Skriptbibliothek verwendet werden kann (Kapitel 52[1136]).

15.8 Optionen

Das Verhalten der `win` Engine kann über die QF-Test Optionen beeinflusst werden. Darüber hinaus können noch Properties gesetzt werden, die den nativen Teil des UI Automation Clients beeinflussen. Diese Optionen und Properties können in einem SUT-Skript Knoten gesetzt werden via:

```
rc.setOption(<name>, <value>)
```

bzw.

```
rc.engine.preferences().setPref(<name>, <value>)
```

Um die gesetzte Option zurückzusetzen genügt:

```
rc.unsetOption(<name>)
```

15.8.1 Windows Skalierung

Auf Grund stetig wachsender Bildschirmauflösungen in den letzten Jahren gibt es in Windows die Möglichkeit die Skalierungseinstellungen anzupassen. Windows-Anwendungen sowie ihre Komponenten und die darin dargestellte Schrift werden an diesen Skalierungsfaktor angepasst. UWP, WPF und Windows-Forms-Anwendungen skalieren normalerweise automatisch aber Win32-Anwendungen und deren Steuerelemente im Besonderen behalten teilweise ihre Größe oder skalieren auf andere Weise.

QF-Test arbeitet standardmäßig mit physischen Bildschirmkoordinaten. Angenommen die Skalierung ist auf 125% gesetzt und eine Schaltfläche würde bei 100% Skalierung an der Position (24,40) mit einer Größe von (100,20) existieren, so wäre diese bei dieser Skalierung an Position (30,50) und hätte eine Größe von (125,25). Daraus ergeben sich folgende Konsequenzen:

- unterschiedliche Geometrie wenn die Komponente erneut aufgenommen wird
- Unterschiede bei der Geometrie, wenn QF-Test versucht eine Komponente zu identifizieren, die mit 100% Skalierung aufgenommen wurde.
- Ein (harter) Mausklick auf einen bestimmten Bereich innerhalb eines Elementes kann fehlschlagen, weil der skalierte Bereich weiter von der oberen-rechten Ecke des Elementes entfernt ist.

Damit QF-Test mit logischen anstelle von physischen Koordinaten arbeitet, kann `Options.OPT_WIN_USE_SCALING` auf `true` gesetzt werden. QF-Test verwendet dann den Skalierungsfaktor des primären Monitors um die Geometrie für Komponenten und die Koordinaten für Maus-Events anzupassen. Bei dieser Anpassung kann es je nach eingestelltem Skalierungsfaktor zu Rundungsfehlern kommen, wenn die Koordinaten neu berechnet werden, da diese als Integerwerte gespeichert sind. Das kann dazu führen, dass Koordinaten um 1 Pixel abweichen.

15.8.2 Sichtbarkeit

Es kann vorkommen, dass ein Ereignis auf eine Komponente ausgelöst werden soll, die gerade nicht sichtbar ist (da sie gerade außerhalb des sichtbaren Bereiches gescrollt ist). Um auf diese Komponente ein `invoke` Event abzuspielen, muss vorher die Prüfung

auf Sichtbarkeit deaktiviert werden, die normalerweise ein Bestandteil der Objekterkennung ist.

Das kann durch das Setzen von `Options.OPT_WIN_TEST_VISIBILITY` auf `false` erreicht werden. Nach der Wiedergabe des jeweiligen Events sollte die Sichtbarkeitsprüfung wieder aktiviert werden.

15.8.3 Verbinden zu einem Fenster einer bestimmten Klasse

Wenn eine Anwendung via `-class <class name>` verbunden wird, ignoriert QF-Test standardmäßig alle Haupt-Fenster der Anwendung, die nicht dieser Klasse entsprechen. Auf diesem Weg kann man QF-Test zum Beispiel lediglich mit der Windows Taskbar verbinden und den Desktop und die darauf liegenden Icons komplett ignorieren, obwohl sie im gleichen Prozess laufen.

Um dennoch alle Haupt-Fenster (toplevels) eines Prozesses zugreifbar zu machen, kann die Präferenz "`windriver.restrict.tops.to.class`" auf "`false`" gesetzt werden.

15.8.4 Begrenzung der Anzahl von Kind-Elementen

Große Hierarchien von UI Automation Elementen können Aufnahme und Wiedergabe deutlich verlangsamen. Um dem entgegenzuwirken, begrenzt QF-Test die Anzahl der Kind-Elemente, wenn diese vom Client abgerufen werden.

Der vorgegebene Wert beträgt 100 und kann über `Options.OPT_WIN_MAX_CHILDREN` angepasst werden.

15.9 (Aktuelle) Einschränkungen

Die aktuelle Implementierung des Windows-Testens enthält noch eine Reihe von Einschränkungen. Wir werden versuchen, die Funktionalität laufend zu verbessern, doch mag die eine oder andere Einschränkung noch eine Weile bestehen.

Da die Unterstützung der UI Automation vom Framework abhängt, mit dem die Anwendung entwickelt wurde, ist die Aufnahme in QF-Test eventuell nicht konsistent. Zum Beispiel kann beim Öffnen eines Dialogs ein Warten auf Komponente aufgezeichnet werden oder auch nicht.

Das Testen von Anwendungen, die aus mehreren Prozessen bestehen, ist komplex und erfordert mehrere `win` Clients.

Weitere Einschränkungen und noch nicht implementierte Funktionalitäten (Stand Januar 2020) sind unter anderem:

- Die unterstützten Check-Arten sind so gut wie vollständig. Es fehlen aber noch einige speziellere Checks. Dies sollen in einer zukünftigen QF-Test Version vervollständigt werden.
- Elemente einer Titelleiste einer Windows App können nicht (einfach) angesprochen werden, weil diese in einem anderen Prozess liegen. Dies könnte in einer zukünftigen QF-Test Version behoben werden.
- Die Event-Weiterleitung vom Textelement einer Schaltfläche auf das Schaltflächenelement selbst erfolgt bei der Aufnahme eines Mausklicks, kann aber an anderer Stelle fehlen. Dies sollte in einer zukünftigen QF-Test Version behoben sein.

15.10 Links

Die Windows Automation API ist beschrieben unter: https://docs.microsoft.com/en-US/windows/desktop/WinAuto/windows-automation-api-portal.

Weitere Informationen zu Mark Humphreys ui-automation Java-Bibliothek finden Sie unter https://github.com/mmarquee.

Kapitel 16

Testen von Android-Anwendungen

6.0+

Dieses Kapitel behandelt die Testautomatisierung von nativen Android-Anwendungen.

Video Zum Einstieg gibt es auch ein kurzes

Überblicksvideo zu Android Testen
https://www.qftest.com/de/yt/android-testen.html

auf unserem QF-Test YouTube Kanal.

Im Juni 2022 fand ein Spezial-Webinar zum Thema Android Testen mit QF-Test statt, das nach etwas Theorie das detaillierte Arbeiten mit dem Emulator und einem echten Gerät zeigt.

Video Hier geht es zum

Videomitschnitt des Spezial-Webinars
https://www.qftest.com/de/yt/android-spezialwebinar.html

auf unserem QF-Test YouTube Kanal.

Hinweis Sollten Sie mobile Web-Anwendungen testen wollen, schauen Sie sich bitte die Möglichkeiten der mobilen Browser-Emulation des Chrome Desktop-Browsers an, welche im Abschnitt 14.6(232) beschrieben werden. Obwohl es möglich ist, einen Webbrowser auf Android-Geräten (vorausgesetzt er unterstützt die Accessibility Schnittstelle) zu nutzen, bietet die mobiles Browser-Emulation bessere Automatisierungsmöglichkeiten und weniger Aufwand für mobiles Web-Testen.

16.1 Voraussetzungen und bekannte Einschränkungen

16.1.1 Voraussetzungen

Für Android Tests mit QF-Test müssen folgende Voraussetzungen für den Testrechner erfüllt sein:

- Soll ein Android-Emulator genutzt werden, ist ein ausreichend performanter Rechner erforderlich (keine alte Mühle - :-). Zusätzlich kann es sinnvoll sein die Hardware-Beschleunigung (typischerweise im Bios) zu aktivieren, falls der Emulator trotzdem zu langsam ist. Weitere Details findet man unter https://developer.android.com/studio/run/emulator-acceleration.
- Die Android SDK Command-Line Tools müssen auf dem Rechner verfügbar sein, besser noch, eine Installation des Android Studios, wie in Abschnitt 16.3(247) beschrieben.
- Entweder: Ein echtes Android Mobilgerät, das (typischerweise per Kabel) mit dem Rechner verbunden ist, wobei USB Debugging auf dem Mobilgerät aktiviert sein muss (siehe Abschnitt 16.4(253)).
- Oder: Ein Android-Emulator ist installiert, welches am einfachsten über das Android Studio(247) geschieht, auf dem ein passendes Android Virtual Device (AVD) läuft (siehe Abschnitt 16.3.2(248)).
- Auf dem echten oder virtuellen Android-Gerät muss die Android-API 24 oder größer sein. Dies entspricht der Android Version 7 Nougat oder neuer (siehe Andriod Versionsliste auf Wikipedia).

16.1.2 Bekannte Einschränkungen

Hinweis

Es gibt folgende Einschränkungen in der dieser Version:

- Nur ein Android Client kann derzeit gleichzeitig verbunden und gesteuert werden. Es ist geplant, dies auf mehrere Clients zu erweitern, wie es auch für die anderen GUI Technologien unterstützt wird.

16.2 Emulator oder echtes Gerät

Wenn man mit Android Testing startet, stellt sich die Frage, wie man beginnen möchte: mit einem virtuellen oder echten Android-Gerät.

Es kann ein echtes Android-Gerät zum automatisierten Testen mit QF-Test verwendet werden. Dieses muss die Entwickleroption für USB Debugging aktiviert haben und per USB Kabel mit dem Rechner verbunden sein, auf dem QF-Test läuft. Mit Hilfe einer passenden Startsequenz verbindet sich QF-Test mit diesem Gerät und kann dieses dann kontrollieren. Es können Aktionen und Tests aufgezeichnet und abgespielt werden.

Ein virtuelles Android-Gerät (AVD) ist die Nachbildung eines echten Gerätes. Es läuft mittels einer Emulator-Software, die auf dem Computer die Hardware und das Verhalten des echten Gerätes emuliert, also nachbildet. Ein Android-Emulator ist also eine Software, um Android-Anwendungen auf einem Computer ausführen und testen zu können. Der Emulator kann hierbei verschiedene virtuelle Android-Geräte laden, die auf unterschiedlichen Android Versionen basieren oder auch Geräte spezifischer Hersteller abbilden.

Bei der Nutzung eines Emulators startet QF-Test diesen im Regelfall am Anfang des Tests, dann lädt und verbindet sich mit dem vom Nutzer definierten virtuellen Android Gerät. Als letztes wird die zu testende Android-Anwendung geöffnet. Dann können Aktionen und Tests aufgezeichnet und ausgeführt werden.

Der Vorteil des Emulators ist, dass man keine Abhängigkeit zu einem externen Gerät hat und flexibel mit unterschiedlichen virtuellen Geräten testen kann. Man hat aber etwas mehr Initialaufwand und ggf. auch mehr Last auf seinem Rechner.

Ein echtes Gerät erlaubt einen schnelleren Einstieg, erfordert aber die Aktivierung des USB Debuggings und ist weniger flexibel.

16.3 Installation des Android Studios, Emulators und virtueller Geräte

Der einfachste Weg zur Installation der erforderlichen Android SDK Command-Line tools und des Emulators sowie zur Konfiguration eines virtuellen Geräts ist das Android Studio. Es besteht auch die Möglichkeit, nur das Android SDK oder sogar nur die SDK Command line tools zu installieren. Dabei gibt es aber einige Fallstricke, weshalb wir uns entschlossen haben, hier den Weg über das Android Studio zu beschreiben.

Falls Sie nicht bereits das Android Studio installiert haben, sind die folgenden Schritte notwendig:

16.3.1 Android Studio installation

- Laden Sie das Android Studio von https://developer.android.com/studio herunter. Die Installation benötigt ca. 2.7 GB Plattenplatz.

- Führen Sie die Installation mit Standardeinstellungen durch, wodurch es im Anschluss auch gestartet wird.
- In dem Setup Assistenten können Sie die vorgeschlagenen Einstellungen übernehmen.
- Bei der Installation könnte nach einer Änderung der Kommandozeileneinstellungen gefragt werden, die Sie annehmen können.
- Schließen Sie die Installation ab.

16.3.2 Android Studio AVD Konfiguration

- Da wir das Android Studio nur zur Konfiguration eines virtuellen Geräts verwenden wollen, wählen Sie bitte Virtual Device Manager aus dem Menü More Actions (manchmal dargestellt durch drei vertikale Punkte ganz oben rechts). Ansonsten finden Sie den Device Manager auch im Tools Menü.

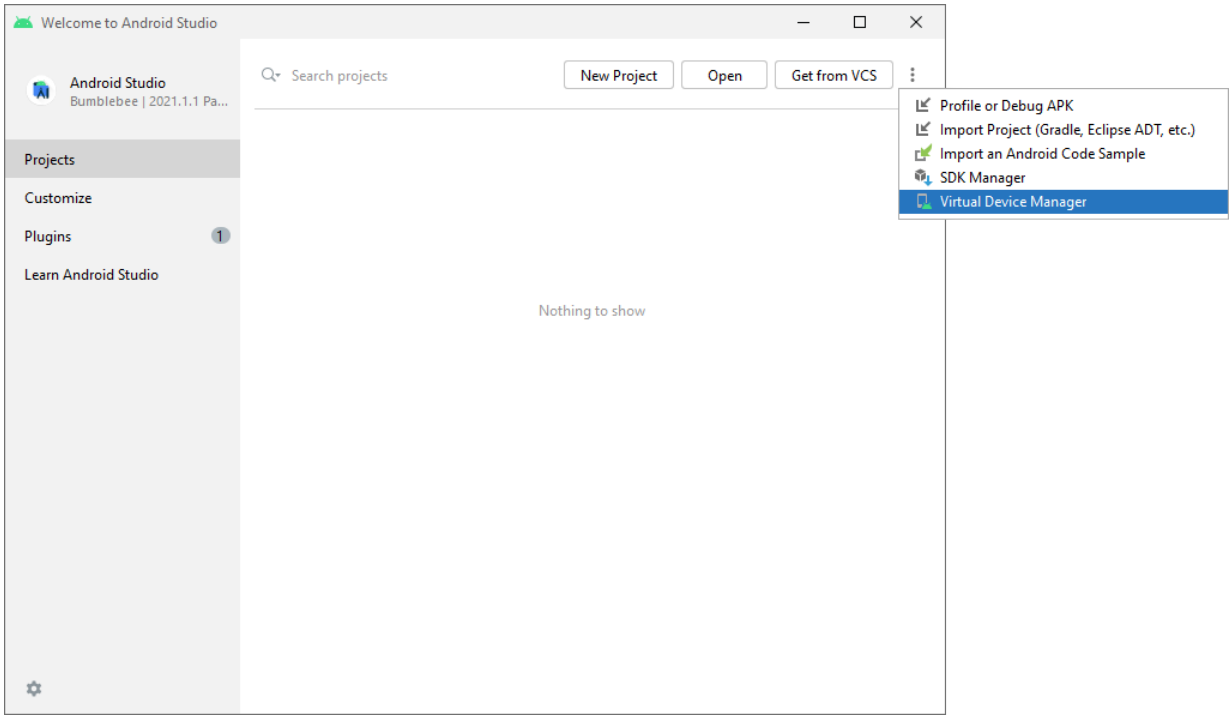


Abbildung 16.1: Android Studio Startfenster

- Wählen Sie Create device....

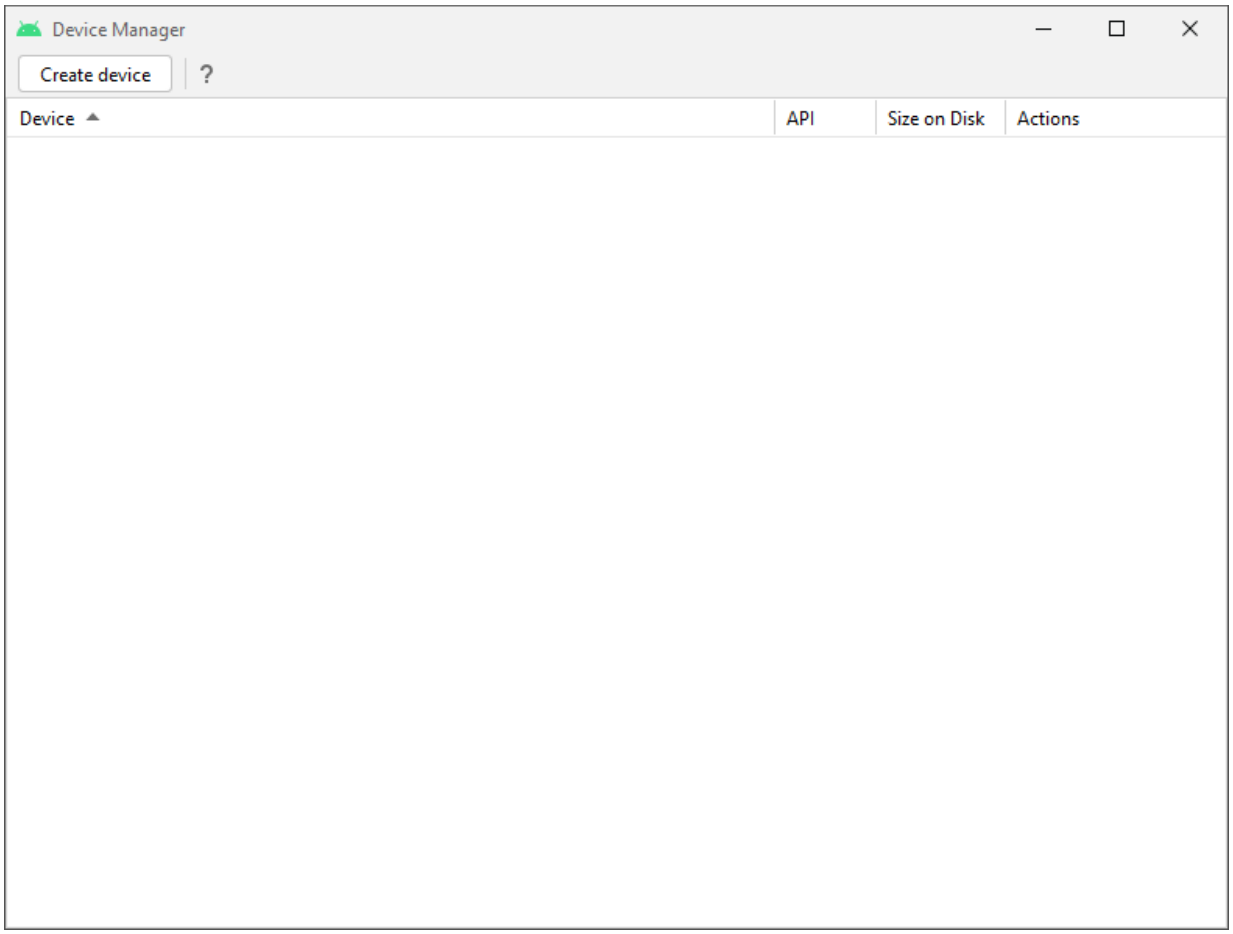


Abbildung 16.2: Android Studio Dialog zur Erzeugung eines virtuellen Gerätes

- Wählen Sie ein passendes virtuelles Gerät. Wir empfehlen eher solche mit kleineren Bildschirmgrößen, da dies sowohl Speicherplatz spart als auch die vollständige Anzeige des virtuellen Geräts auf Ihrem Bildschirm ermöglicht.

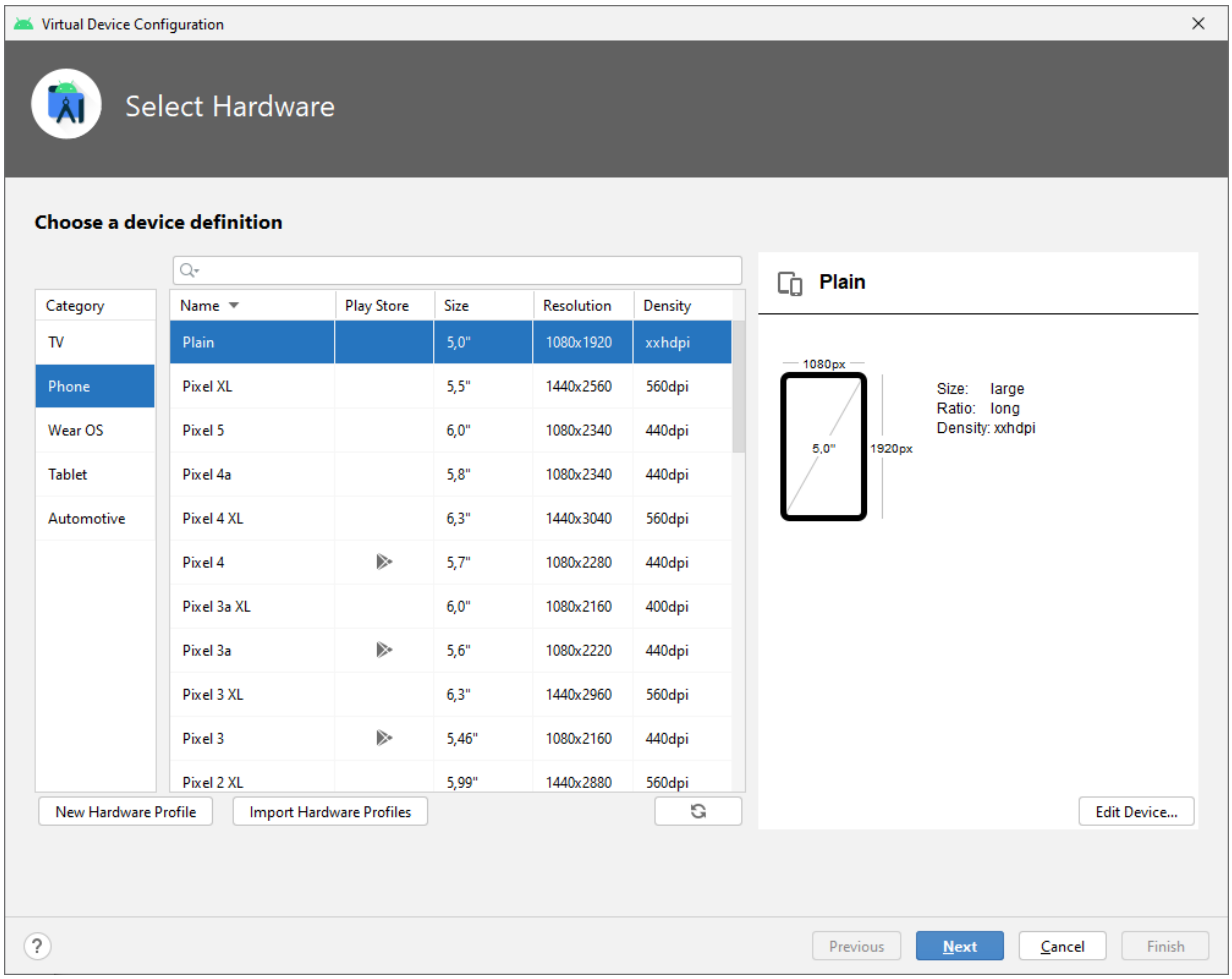


Abbildung 16.3: Android Studio Dialog zur Auswahl der Gerätedefinition

- Laden Sie nun das System Image für die gewünschte Android Version herunter. Klicken Sie den entsprechenden ”Download” Link um den ”Component Installer” zu starten und danach auf ”Finish”.

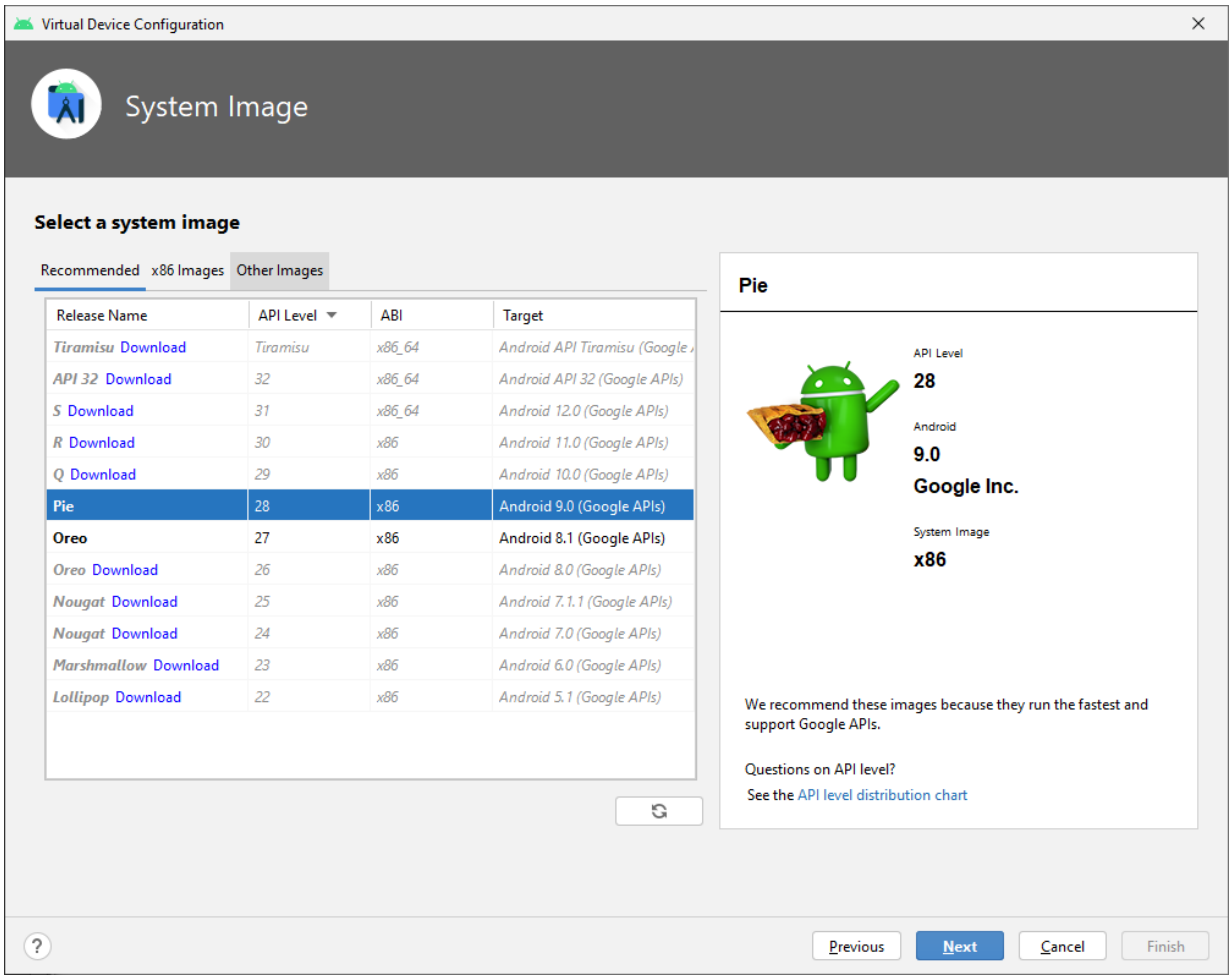


Abbildung 16.4: Android Studio Dialog für Download und Auswahl des System Images

- Um die Installation abzuschließen drücken Sie ”Finish” auf dem letzten Konfigurationsdialog.

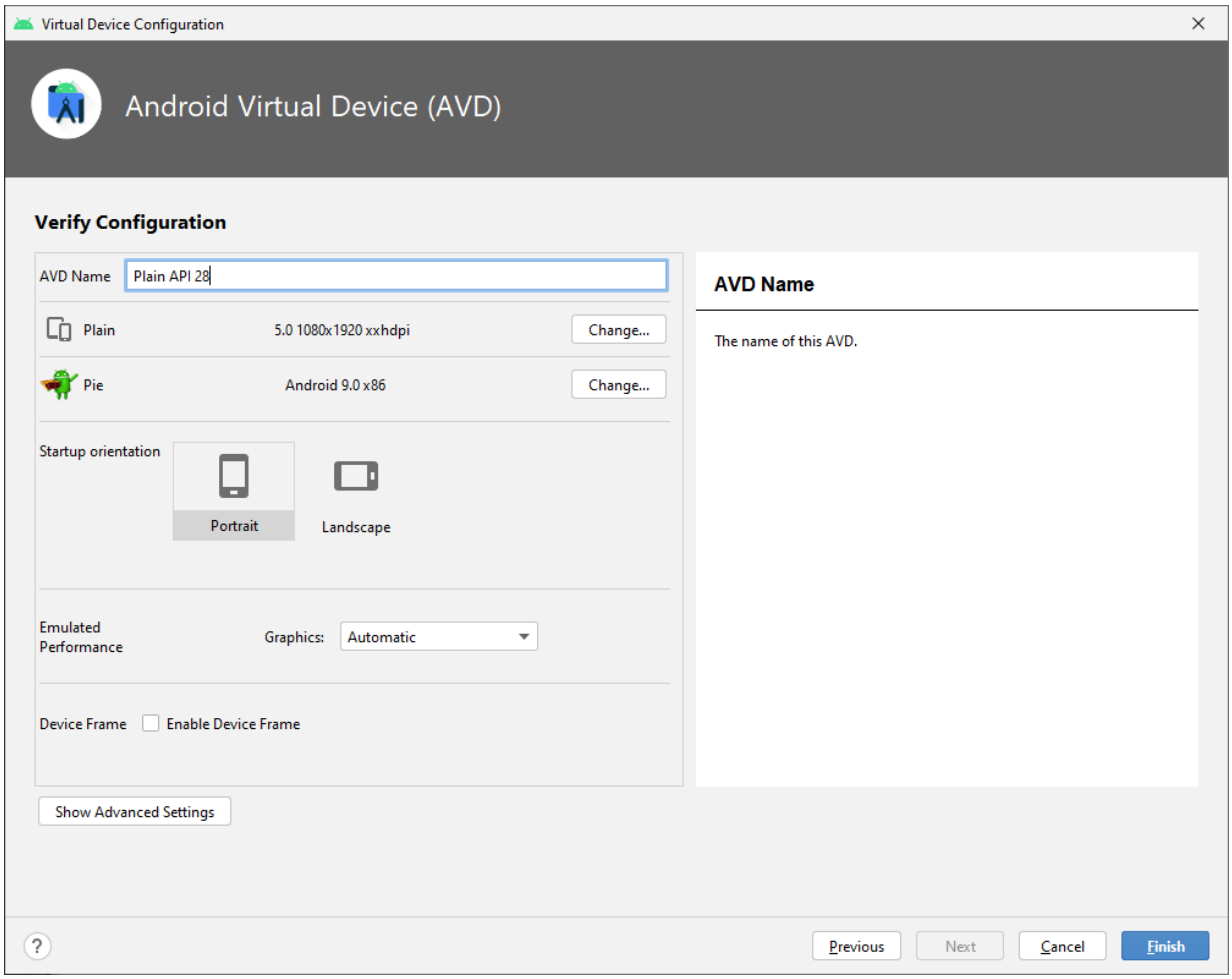


Abbildung 16.5: Android Studio Dialog zum Abschluss der AVD Konfiguration

- Nun steht das erste virtuelle Gerät (AVD) für QF-Test bereit.

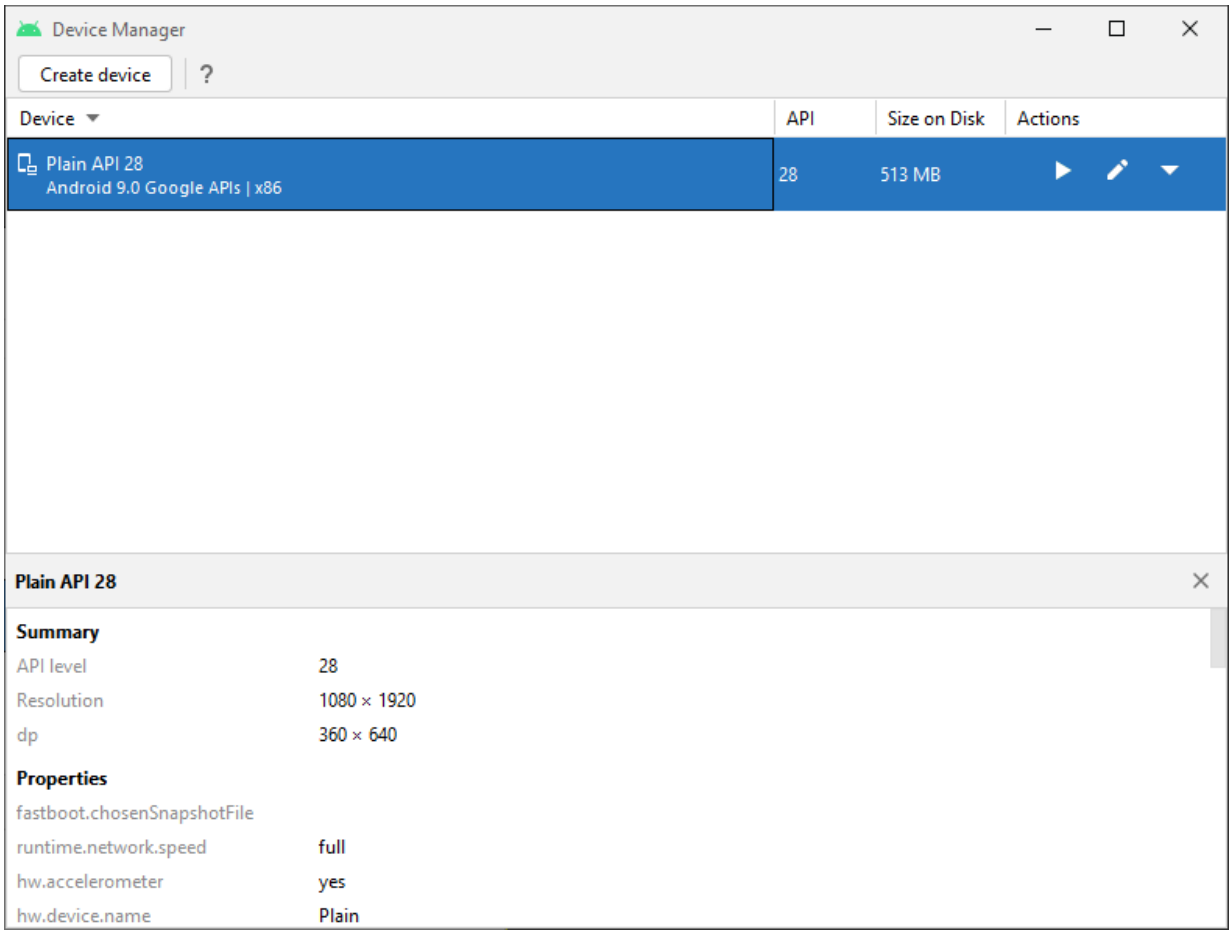


Abbildung 16.6: Android Studio Dialog zeigt verfügbare AVDs

16.4 Verbinden mit einem echten Android-Gerät

Für Tests auf einem echten Android-Gerät muss dieses über ein USB-Kabel mit dem Rechner, auf dem QF-Test läuft, verbunden sein. Außerdem muss USB Debugging aktiviert werden.

Zusätzlich wird ein Android SDK auf Ihrer Maschine benötigt. Obwohl es ausreichend sein kann, nur die SDK Kommandozeilewerkzeuge zu installieren, empfehlen wir das volle Android Studio zu nutzen, wie in Abschnitt 16.3(247) beschrieben.

USB Debugging aktivieren

Aktivieren Sie USB Debugging für Ihre Android-Gerät. Im Regelfall sind dafür folgende Schritte notwendig:

1. Öffnen Sie auf dem Gerät ”Einstellung” -> ”Über ’Gerät”’.
2. Tippen Sie sieben Mal auf die Build-Nummer, um ”Einstellungen” -> ”Entwickleroptionen” verfügbar zu machen.
3. Dort aktivieren Sie dann die Option ”USB Debugging”

Die Referenzdokumentation zum Aktivieren der Entwickleroptionen finden Sie unter https://developer.android.com/studio/debug/dev-options.

Geräte und PC mittels USB Kabel verbinden

- Wenn Sie Ihr Android-Gerät mit dem Rechner verbunden haben und eine Abfrage erscheint, ob USB Debugging für diesen Rechner erlaubt / dauerhaft erlaubt werden soll, so bestätigen Sie dies bitte.

16.5 Eine QF-Test Startsequenz für Android Tests erzeugen

- Wie immer, öffnen Sie den Schnellstart-Assistenten über das Extras Menü oder über den  Knopf in der Werkzeugleiste.
- Wählen Sie ”Eine Android-Anwendung”.

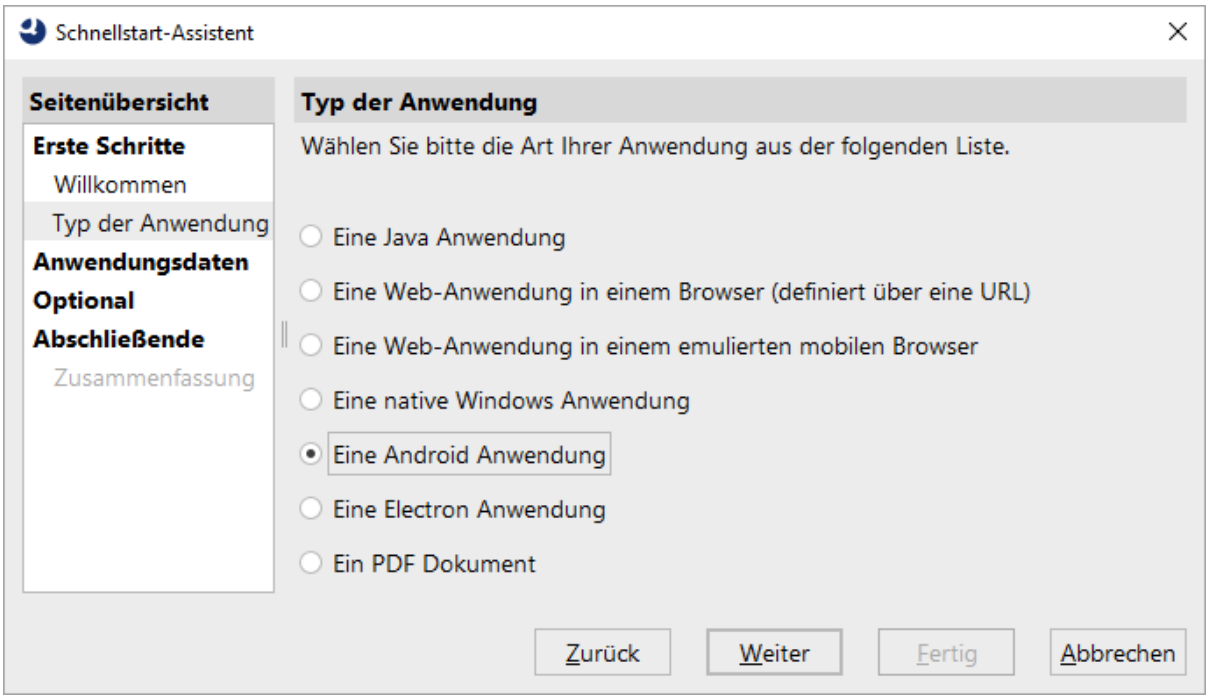


Abbildung 16.7: Auswahl des Anwendungstyps im Schnellstart-Assistenten

16.5.1 Nutzung eines Android-Emulators

- Wählen Sie die erste Auswahl ”Einen Emulator starten und mit virtuellem Gerät verbinden”.

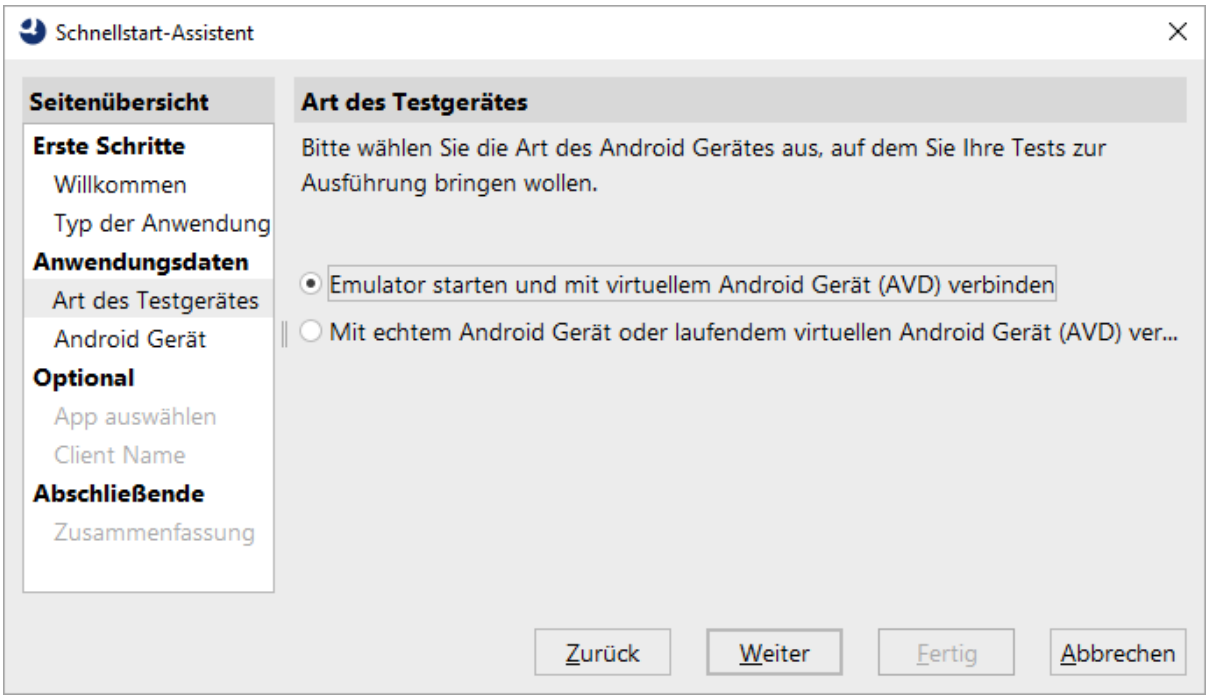


Abbildung 16.8: Auswahl des Emulators als genutztes Testgerät im Schnellstart-Assistenten

- Wählen Sie das virtuelle Gerät aus der Auswahlliste oben. Drücken Sie ”Aktualisieren”, falls kein virtuelles Gerät angezeigt werden sollte. Falls dann immer noch keines sichtbar ist, starten Sie ggf. QF-Test neu, um neu hinzugekommene Geräte einzulesen. Klicken Sie dann ”Weiter”.

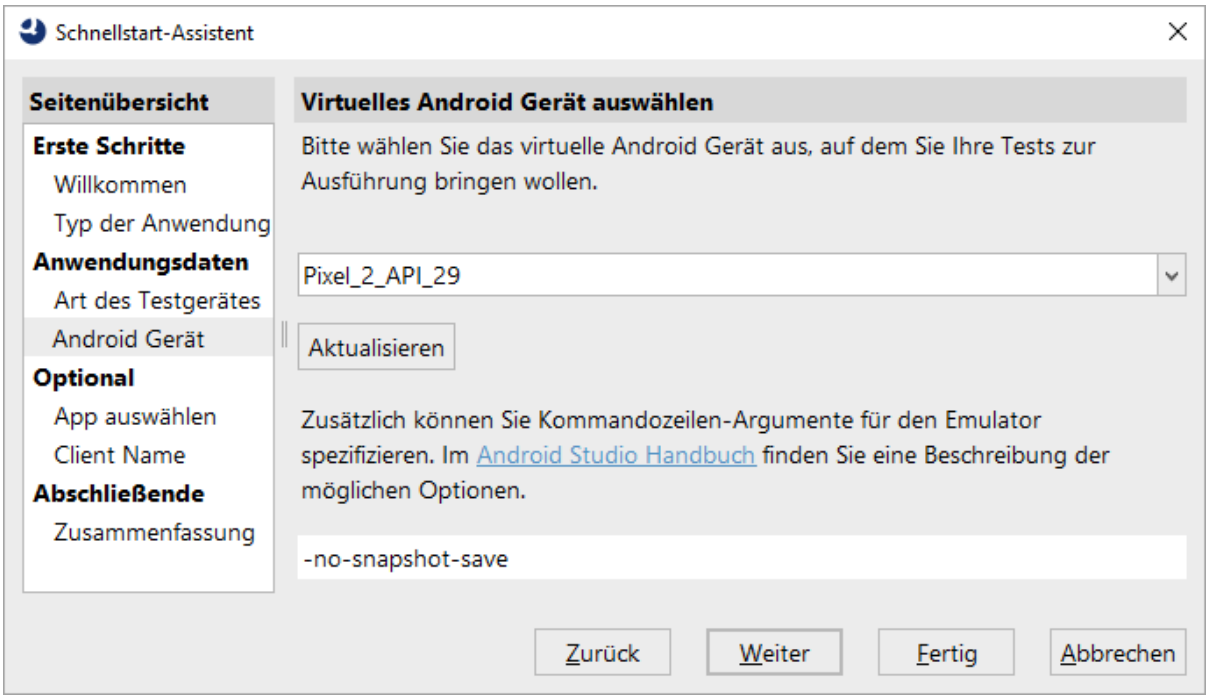


Abbildung 16.9: Auswahl des AVDs im Schnellstart-Assistenten

- Als nächsten Schritt können Sie die Android .apk Datei der zu testenden App angeben. Falls Sie eine App testen wollen, die bereits auf dem Android-Gerät installiert ist, so überspringen Sie diesen Schritt.

Abbildung 16.10: Auswahl der .apk Datei im Schnellstart-Assistenten

- Im nächsten Schritt können Sie einen Client-Namen angeben. Klicken Sie dann ”Weiter” und zum Schluss ”Fertig”.

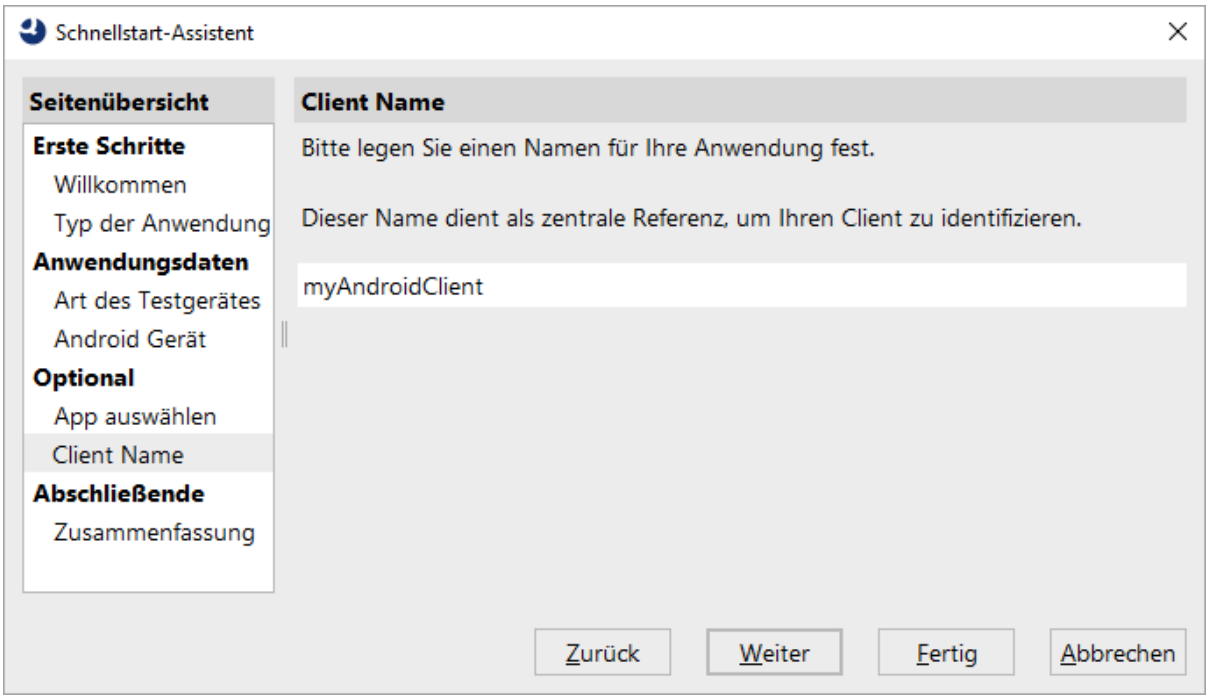


Abbildung 16.11: Auswahl des Client-Namens im Schnellstart-Assistenten

- Nun wird in der aktuellen Testsuite im Knoten ”Extrasequenzen” eine Vorbereitungssequenz erzeugt. Sie sollte selbsterklärend sein.

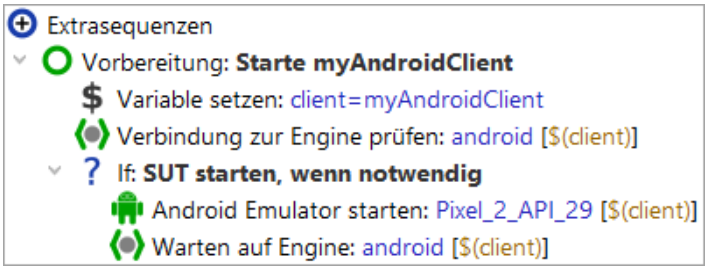


Abbildung 16.12: Vom Schnellstart-Assistenten erzeugte Startsequenz für den Android-Emulator

- Wenn Sie die Vorbereitungssequenz ausführen, sollte das Emulatorfenster erscheinen und nach kurzer Zeit auch die App, falls Sie eine .apk Datei angegeben haben. Der Aufnahmeknopf  sollte aktiviert sein, was anzeigt, dass die Verbindung erfolgreich aufgebaut wurde.

Abbildung 16.13: Fenster des Android-Emulators

16.5.2 Nutzung eines echten Android-Gerätes

- Wählen Sie die zweite Option ”Mit echtem Android-Gerät oder laufendem virtuellen Android-Gerät verbinden”.

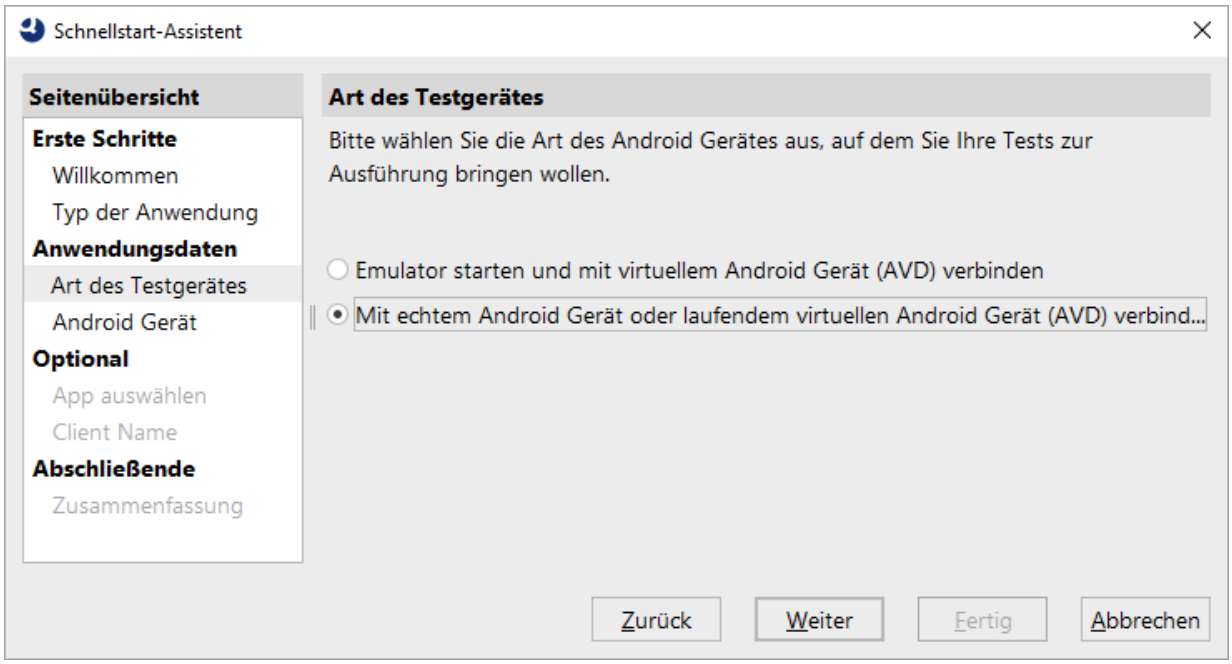


Abbildung 16.14: Auswahl eines echten Gerätes als Typ im Schnellstart-Assistenten

- Wählen Sie aus der Auswahllist den angegebenen Eintrag, die die ID des verbundenen Geräts darstellt. Klicken Sie ”Aktualisieren” falls kein Gerät sichtbar ist. Falls immer noch kein Gerät angezeigt wird, starten Sie QF-Test neu, damit neu angeschlossenen Geräte erkannt werden. Klicken Sie nun ”Weiter”.

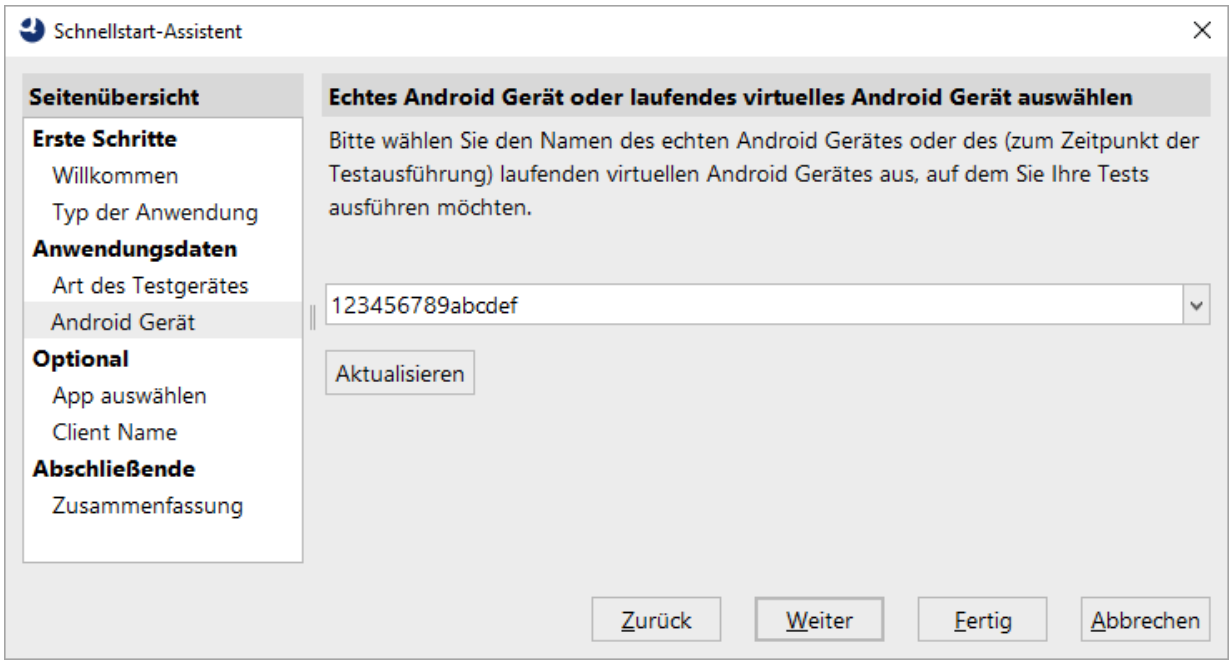


Abbildung 16.15: Auswahl des Gerätes im Schnellstart-Assistenten

- Als nächsten Schritt können Sie die Android .apk Datei der zu testenden App angeben. Falls Sie eine App testen wollen, die bereits auf dem Android-Gerät installiert ist, so überspringen Sie diesen Schritt.

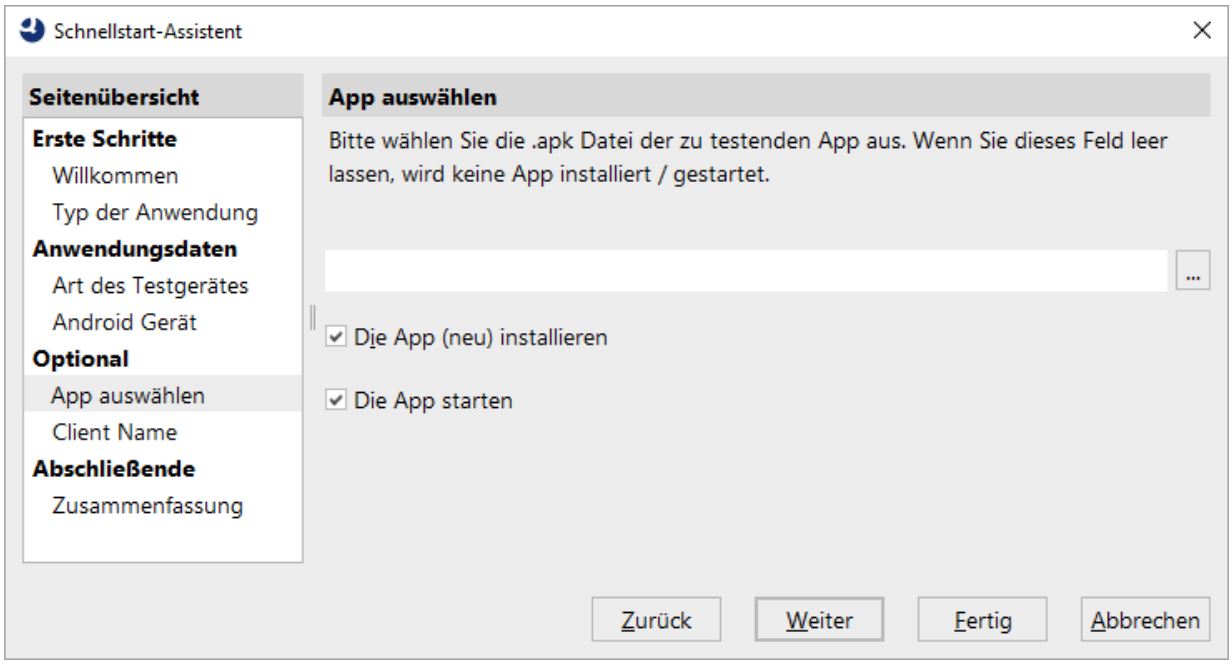


Abbildung 16.16: Auswahl der App im Schnellstart-Assistenten

- Im nächsten Schritt können Sie einen Client-Namen angeben. Klicken Sie dann ”Weiter” und zum Schluss ”Fertig”.

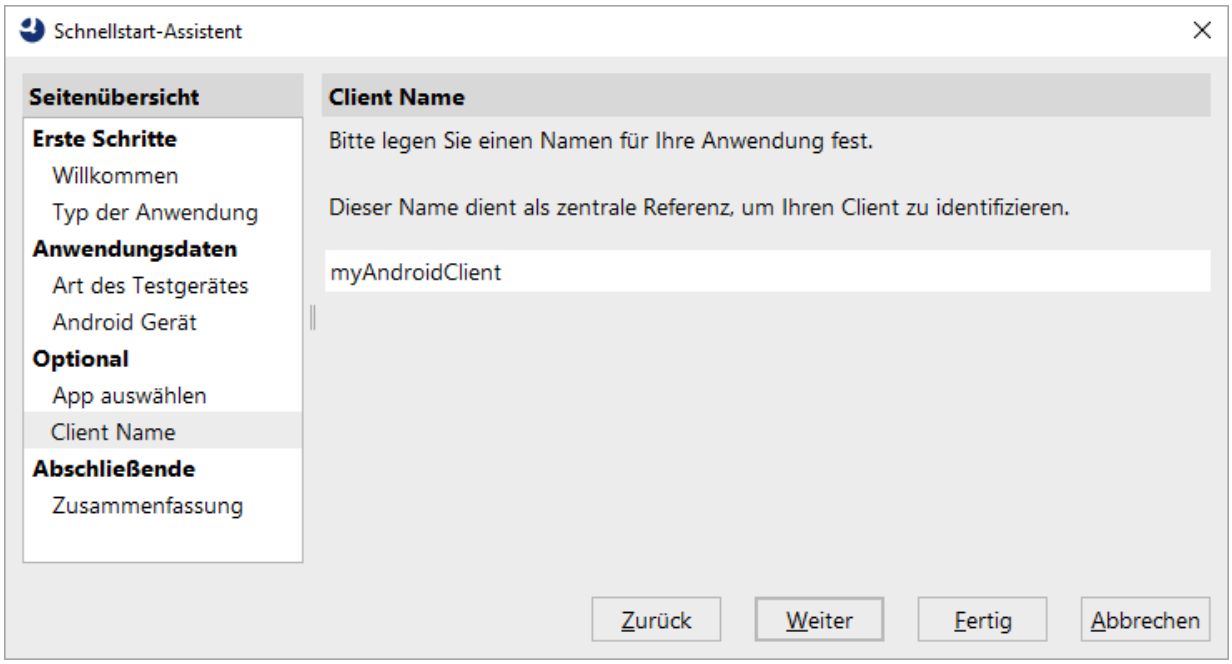


Abbildung 16.17: Festlegung des Client-Names im Schnellstart-Assistenten

- Nun wird in der aktuellen Testsuite im Knoten ”Extrasequenzen” eine Vorbereitungssequenz erzeugt. Sie sollte selbsterklärend sein und enthält den Hinweis, dass in dieser Frühtesterphase die qfsandroid.qft Testsuite für die Ausführung der Vorbereitungssequenz benötigt wird und automatisch inkludiert wurde.

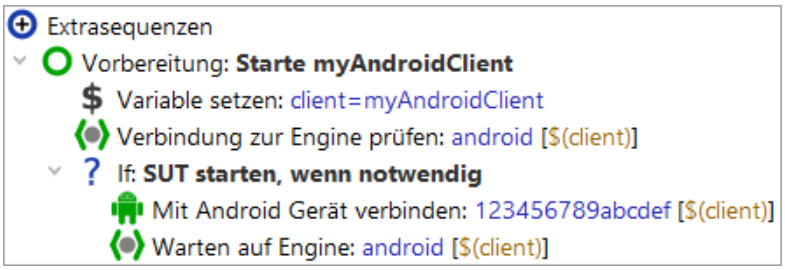


Abbildung 16.18: Vom Schnellstart-Assistenten erzeugte Startsequenz für das echte Android-Gerät

- Wenn Sie die Vorbereitungssequenz ausführen, sollte der Aufnahmeknopf aktiviert werden, was anzeigt, dass die Verbindung erfolgreich zustande gekommen ist. Falls Sie eine .apk Datei angegeben haben, sollte die App nun auf dem Android-Gerät sichtbar werden.

16.6 Aktionen und Checks auf der Android-App aufnehmen

- Bitte drücken Sie den Aufnahmeknopf, um zu sehen, wie sich QF-Test bei Android Aufnahmen verhält.
- Es erscheint ein spezielles Aufnahmefenster, das den Bildschirm des echten Geräts beziehungsweise des Emulators wiedergibt. Dieses Fenster ist notwendig, da es aktuell nicht möglich ist, Aktionen direkt auf dem Gerät oder Emulator aufzuzeichnen. Dies geschieht über das Aufzeichnungsfenster.
- Mit den Zoom Knöpfen können Sie die Anzeige auf die gewünschte Größe bringen. Bitte beachten Sie, dass das Aufnahmefenster nur ein Abbild der Gerätebildschirms zeigt. Dieses muss gegebenenfalls manuell über den Aktualisierungsknopf aufgefrischt werden oder kann auch automatisch durch Aktivieren von erfolgen.

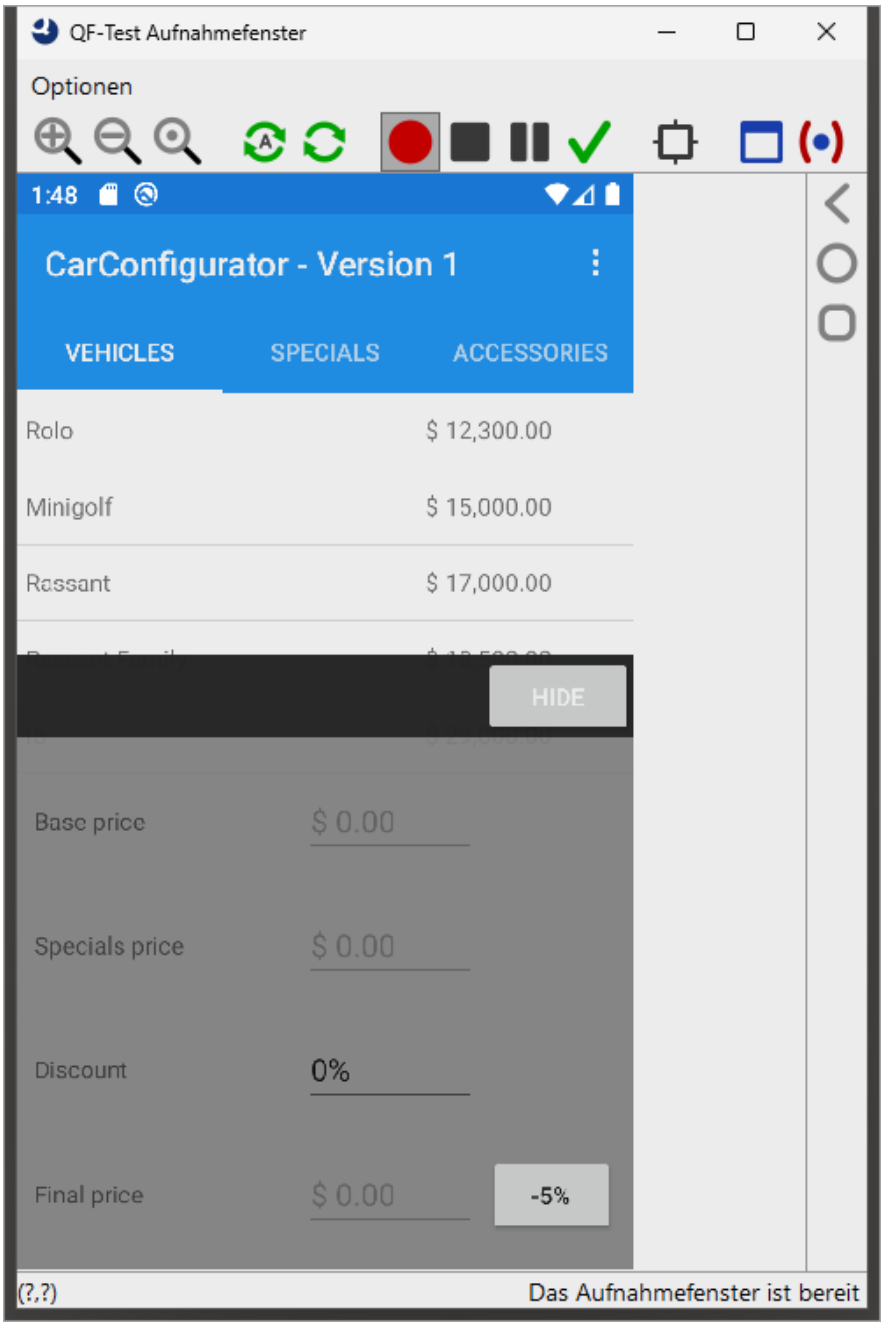


Abbildung 16.19: QF-Test Android-Aufnahmefenster

- Nun können Sie Aktionen und Checks aufnehmen und auch wieder abspielen. Anfangs mag es sich etwas ungewohnt anfühlen, mit dem Aufnahmefenster zu arbeiten, aber Sie werden sich bald daran gewöhnen.
- Abgesehen von diesen Android Spezifika sollte QF-Test genauso funktionieren

wie bei den anderen GUI Technologien, abgesehen von in Abschnitt 16.1.2[246] beschriebenen bekannten Einschränkungen.

- Zusatzfunktionen:

  Das Aufnahmefenster bietet auch ein paar Annehmlichkeiten. In der Fußzeile links werden die Mauskoordinaten angezeigt, was sehr praktisch ist, wenn man mit absoluten Mausklicks arbeiten muss. Rechts wird die Klasse der letzten hervorgehobenen Komponente angezeigt.

  Über den  Fadenkreuz-Knopf in der Werkzeugleiste können Sie ein Inspektor-Fenster öffnen, siehe Abschnitt 5.12.2[108], das die Komponentenhierarchie aller sichtbaren Komponenten inklusive Größe und Koordinaten anzeigt. Dies dient hauptsächlich der Nachverfolgung von Problemen bei der Komponentenaufnahme oder Wiedererkennung, kann aber auch sonst hin und wieder nützlich sein.

16.7 Android Hilfsprozeduren

Es gibt eine ganze Reihe an verfügbaren Hilfsprozeduren für Android in der Standardbibliothek[183]. Sie befinden sich in dem entsprechend benannten Package ”android”.

Einige sind analog zu den anderen GUI Technologien, es gibt aber einige sehr spezifische für mobiles Testen, z.B. um Wischaktionen oder Gesten durchzuführen, zum Blättern auf dem Bildschirm und um bestimmte Einstellungen für Komponenten vorzunehmen.

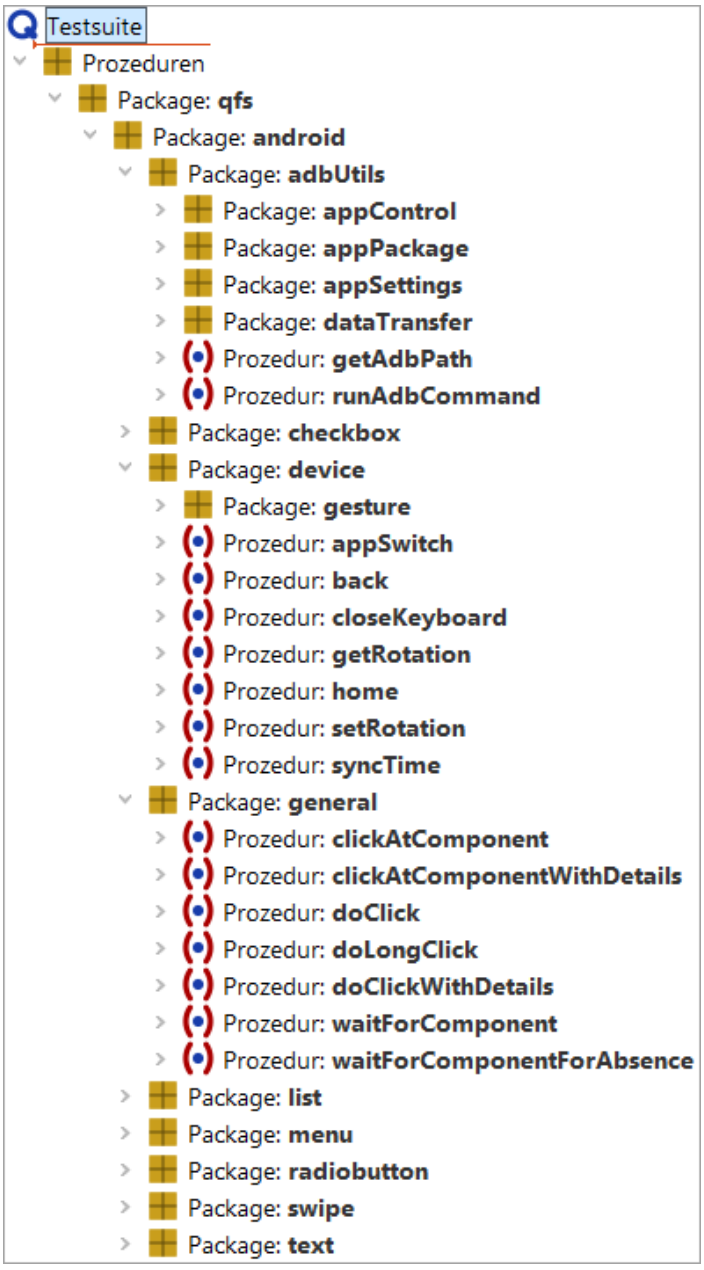


Abbildung 16.20: Android Hilfsprozeduren

Kapitel 17

Testen von iOS-Anwendungen

8.0+

Dieses Kapitel behandelt die Testautomatisierung von nativen iOS-Anwendungen.

Video Zum Einstieg gibt es auch ein kurzes

Überblicksvideo zum iOS-Testing
https://qftest.com/de/yt/ios-ueberblick.html

auf unserem QF-Test YouTube-Kanal.

Im September 2024 fand ein Spezialwebinar zum Thema iOS-Testen mit QF-Test statt, das nach etwas Theorie das detaillierte Arbeiten mit dem Simulator und einem echten Gerät zeigt.

Video Hier geht es zum

Videomitschnitt des Spezialwebinars
https://qftest.com/de/yt/ios-spezialwebinar.html

auf unserem QF-Test YouTube-Kanal.

Hinweis Sollten Sie mobile Web-Anwendungen testen wollen, schauen Sie sich bitte die Möglichkeiten der mobilen Browser-Emulation des Chrome Desktop-Browsers an, welche im Abschnitt 14.6(232) beschrieben werden. Obwohl es möglich ist, einen Webbrowser auf iOS-Geräten (vorausgesetzt er unterstützt die Accessibility Schnittstelle) zu nutzen, bietet die mobile Browser-Emulation bessere Automatisierungsmöglichkeiten und weniger Aufwand für mobiles Web-Testen.

17.1 Voraussetzungen und bekannte Einschränkungen

17.1.1 Voraussetzungen

Für iOS-Tests mit QF-Test müssen folgende Voraussetzungen für den Testrechner erfüllt sein:

- iOS-Anwendungen können nur auf einem macOS-System getestet werden. Auf diesem muss QF-Test installiert sein und der Testlauf dort gestartet werden (interaktiv, im Batch-Modus oder über Daemon-Aufrufe).
- Zum Ausführen der Tests auf dem iOS Simulator oder einem iOS-Gerät muss die Entwicklungsumgebung Xcode mindestens in Version 13 vollständig installiert sein. Um Probleme beim Update zu vermeiden ist es empfohlen, die automatischen Updates in den App Store-Einstellungen zu deaktivieren und Xcode manuell zu aktualisieren, wenn kein Test gestartet ist.
- In Xcode muss die iOS-Entwicklungs-Plattform aktiviert und die passenden iOS-Simulatoren/Runtimes installiert werden. Dies ist im Dialog ”Settings” bzw. ”Preferences” im Tab ”Platforms” bzw. ”Components” möglich. Nach einem Update von Xcode muss dieser Prozess wiederholt werden.
- Der korrekte Entwicklungs-Pfad `/Applications/Xcode.app/Contents/Developer` muss über das Terminal ausgewählt werden: `sudo xcode-select -s /Applications/Xcode.app/Contents/Developer`.
- Zur Steuerung des iOS Gerätes bzw. des iOS Simulators wird die iOS Development Bridge benötigt. Hinweise zu deren Installation finden Sie unter https://fbidb.io/docs/installation.

Im Menü ”Extras” des QF-Test Hauptfensters finden Sie den Eintrag ”iOS-Testumgebung prüfen/einrichten ...”, mit welchem Sie die Voraussetzungen auf Ihrem System überprüfen können sowie Hilfestellung für die Installation der benötigten Hilfsprogramme erhalten. Wenn die Hilfsprogramme zum ersten Mal aufgerufen werden, dann kann es vorkommen, dass diese mehr als 30 Sekunden zur Initialisierung benötigen. In diesem Fall wird aufgrund von Timeouts die Version nicht korrekt ausgelesen. Sie können in diesem Fall die Überprüfung nach einer kurzen Wartezeit erneut starten.

17.1.2 Bekannte Einschränkungen

- Aktionen, die direkt auf einem angeschlossenen Gerät oder im Simulator ausgeführt werden, können nicht aufgezeichnet werden. Diese müssen auf einem dedizierten Aufnahmefenster, siehe Aufnamen und Checks bei iOS(282) ausgeführt werden, analog zu Android.
- Ab iOS-Version 13 wird bei der Verwendung von SecureField-Komponenten (für die Eingabe von Passwörtern und anderen sensiblen Informationen) die Software-Tastatur im Aufnahmefenster nicht mehr angezeigt, und die Textkomponente selbst wird als leer dargestellt, auch wenn sie eine Eingabe enthält. Eingaben können aber grundsätzlich auch über Tastatureingaben aufgenommen werden, ohne dass man dafür die entsprechenden Tasten auf der Software-Tastatur anklicken muss. Die Komponenteninformationen für die Software-Tastatur sind jedoch verfügbar und können in den Tests für das Abspielen von Mausevents auf die Tastatur-Komponenten verwendet werden. Hierfür kann man die Komponentenaufnahme (siehe Komponenten aufnehmen(44)) für das gesamte Fenster verwenden um die Tastatur-Komponenten aufzunehmen oder man kann direkt mit SmartID(81) arbeiten. Einen passenden SmartID-Vorschlag erhält man mithilfe des UI-Inspektor(108).

17.2 Xcode, Simulatoren und IDB installieren

Für iOS-Tests muss die Entwicklungsumgebung Xcode vollständig installiert sein ebenso wie die iOS Development Bridge (idb).

Im Menü ”Extras” des QF-Test Hauptfensters finden Sie den Befehl ”iOS-Testumgebung prüfen/einrichten ...”. Damit können Sie das aktuelle System überprüfen. Außerdem erhalten Sie Empfehlungen für die Installation der benötigten Tools. Wenn ein Tool das erste Mal gestartet wird, kann es vorkommen, dass die Initialisierung mehr als 30 Sekunden dauert. In diesem Fall wird die Wartezeit überschritten und es wird eine falsche Versionsnummer angegeben. Starten Sie in diesem Fall die Prozedur zum Prüfen/Einrichten erneut.

17.2.1 Xcode installieren

- Installieren Sie Xcode Version 13 oder höher vom App Store.

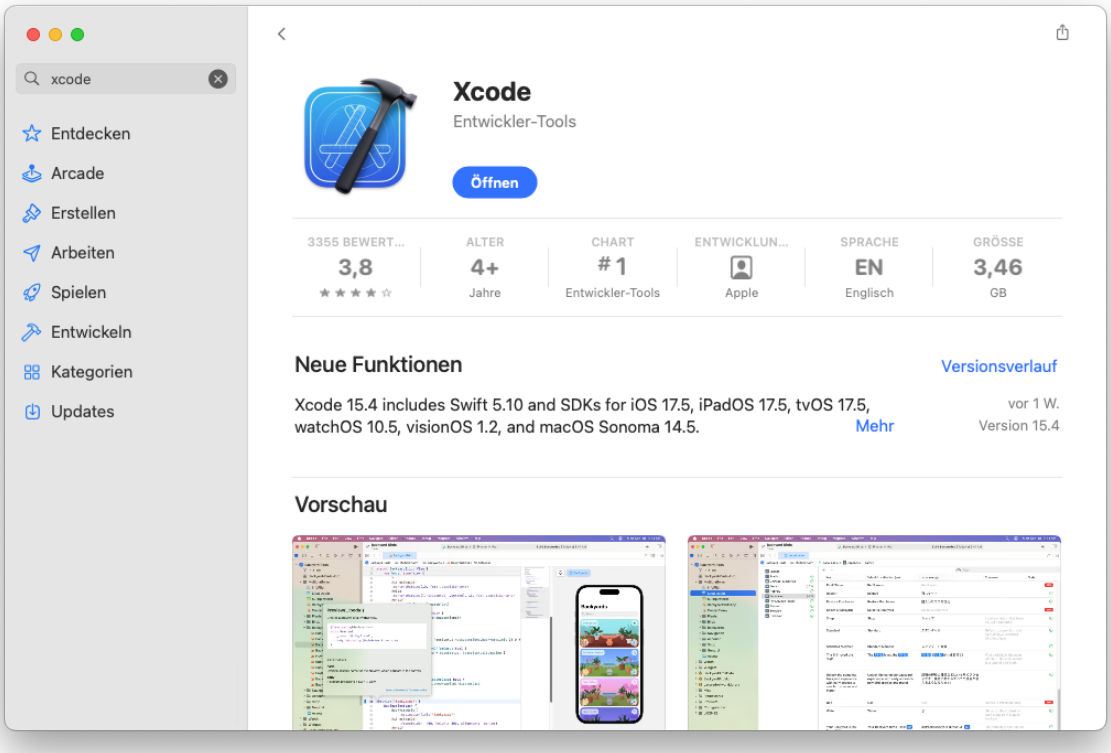


Abbildung 17.1: Xcode im macOS App Store

- Wenn Xcode aktualisiert wird, während ein Test läuft, kann die Installation beschädigt werden. Daher wird empfohlen, die automatische Aktualisierung von Applikationen im App Store oder den Systemeinstellungen zu deaktivieren und Xcode manuell zu aktualisieren, wenn kein Test läuft.

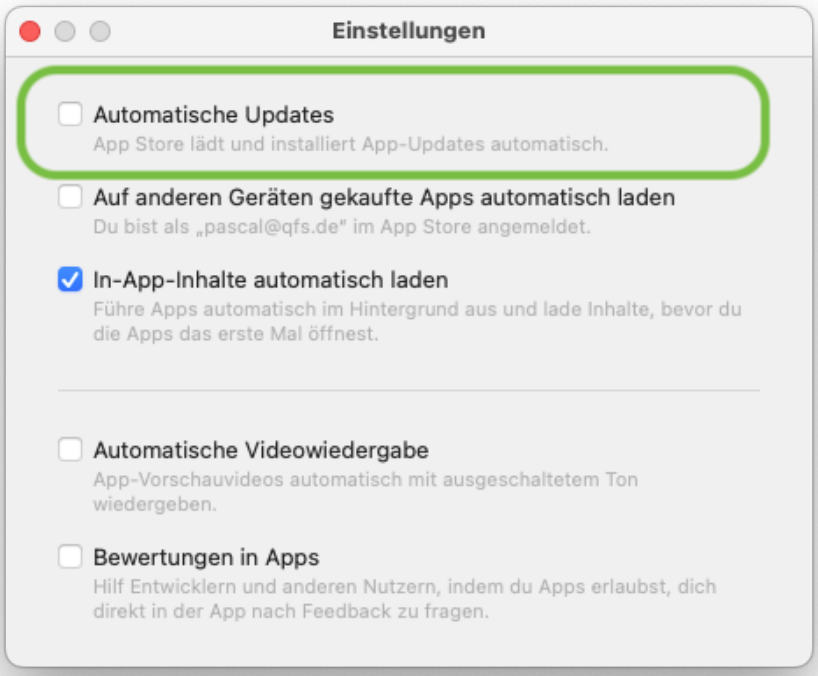


Abbildung 17.2: Empfohlene App Store Einstellungen

- Stellen Sie den korrekten Entwicklungs-Pfad `/Applications/Xcode.app/Contents/Developer` via Terminal ein:

```
sudo xcode-select -s /Applications/Xcode.app/Contents/Developer
```

Beispiel 17.1: Einstellung des Xcode Entwicklungs-Pfad im Terminal

- Öffnen Sie nach der Installation Xcode und stoßen Sie die Installation weiterer benötiger Software gemäß Aufforderung an. Dies muss mindestens eine iOS-Plattform umfassen. Wenn Sie diesen Dialog beim ersten Start übergehen, können Sie Installation über den Dialog XCode→Settings... im Bereich Platforms erneut anstoßen. Über das Menü, das mit Rechtsklick geöffnet wird, ist es auch möglich, installierte Frameworks wieder zu entfernen.

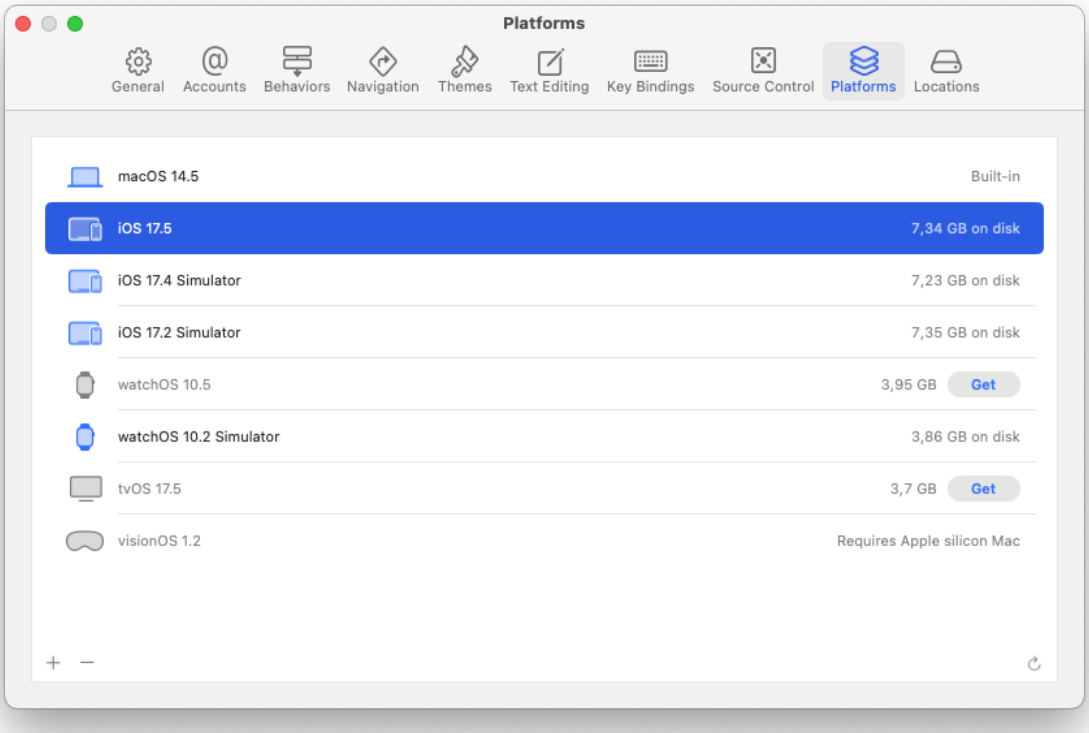


Abbildung 17.3: Plattform Verwaltung in Xcode

- Starten Sie den iOS-Simulator einmal über den Menüeintrag XCode→Open Developer Tool→Simulator um so sicherzugehen, dass die richtigen Simulator-Einstellungen erstellt wurden. Hier ist es auch möglich, weitere Geräte festzulegen.

Abbildung 17.4: Das iOS Simulator Menü

17.2.2 iOS Development Bridge (idb) installieren

Um mit dem iOS-Gerät zu interagieren, verwendet QF-Test die iOS Development Bridge (idb). Sie besteht aus dem idb companion, der direkt mit dem (simulierten) Gerät kommuniziert, und dem Python-basierten idb Client. Beide müssen auf dem System vorhanden sein, um iOS-Tests ausführen zu können. Weitere Informationen über die `idb` finden Sie in der idb Dokumentation.

- Die Installation des idb companion kann über das Kommandozeilenprogramm Homebrew durchgeführt werden (siehe https://brew.sh). Zur Installation des idb companion, führen Sie auf Kommadozeile Folgendes aus:

```
brew tap facebook/fb
brew install idb-companion
```

Beispiel 17.2: idb companion Installation über die Kommandozeile

- Der idb client benötigt eine Installation von Python 3.6 oder höher auf dem System. Dies kann ebenfalls über Homebrew auf Kommadozeile erfolgen. Anschließend wird der idb client mittels `pip` Tool von Python installiert:

```
brew install python3
pip3 install --upgrade pip
pip3 install fb-idb
```

Beispiel 17.3: idb client Installation über die Kommandozeile

17.3 Auf einem echten iOS-Gerät testen

Apps auf dem iOS Simulator zu testen geht schnell und einfach, aber manchmal ist es notwendig, einen Test auf einem echten iOS-Gerät, das an das Testsystem angeschlossen ist, laufen zu lassen. Um iOS-Tests auf einem echten iOS-Gerät ausführen zu können, müssen einige Voraussetzungen erfüllt sein:

1. Das System muss wie in Abschnitt 17.2(271) beschrieben vorbereitet werden, einschließlich einer vollständigen Installation von Xcode.
2. Auf dem Gerät muss unter Xcode→Einstellungen→Datenschutz & Sicherheit der *Entwicklermodus* aktiviert werden.
3. Das Gerät muss an den Rechner angeschlossen sein, wobei auf dem Gerät bestätigt werden muss, dass man dem Rechner ”vertraut”.
4. Das Gerät muss während des Testens entsperrt sein.
5. Ein Entwickler-Account muss in Xcode→Settings. . . →Accounts unter Verwendung seiner Apple ID hinzugefügt werden.
6. Die zum ausgewählten Entwickler-Account gehörige Team ID muss über die Option Code Signing Team ID / Organisationseinheit(565) bereitgestellt werden.
7. Manchmal schlägt der erste Test fehl, weil das Profil auf dem Gerät nicht als vertrauenswürdig eingestellt ist. Um das zu ändern, öffnen Sie die Einstellungen und navigieren Sie auf dem Gerät zu Allgemein→VPN und Geräteverwaltung oder Profile und bestätigen Sie dort die Vertrauenswürdigkeit.

Abbildung 17.5: Navigation zum Abschnitt für Vertrauenswürdigkeit im iOS-Profil

8. Applikationen, die auf dem Gerät während eines Tests installiert werden, müssen für ”Any iOS device” (nicht zu verwechseln mit ”Any iOS Simulator”) gebaut und korrekt signiert werden. Des Weiteren muss ein Provisioning Profile, das die Ausführung erlaubt, auf dem Gerät installiert sein (siehe Apple Dokumentation).

17.4 QF-Test Vorbereitung Sequenz für iOS Tests

- Öffnen Sie den Schnellstart-Assistenten über das Menü Extras oder den Toolbar Button .
- Wählen Sie ”Eine iOS-Application”.

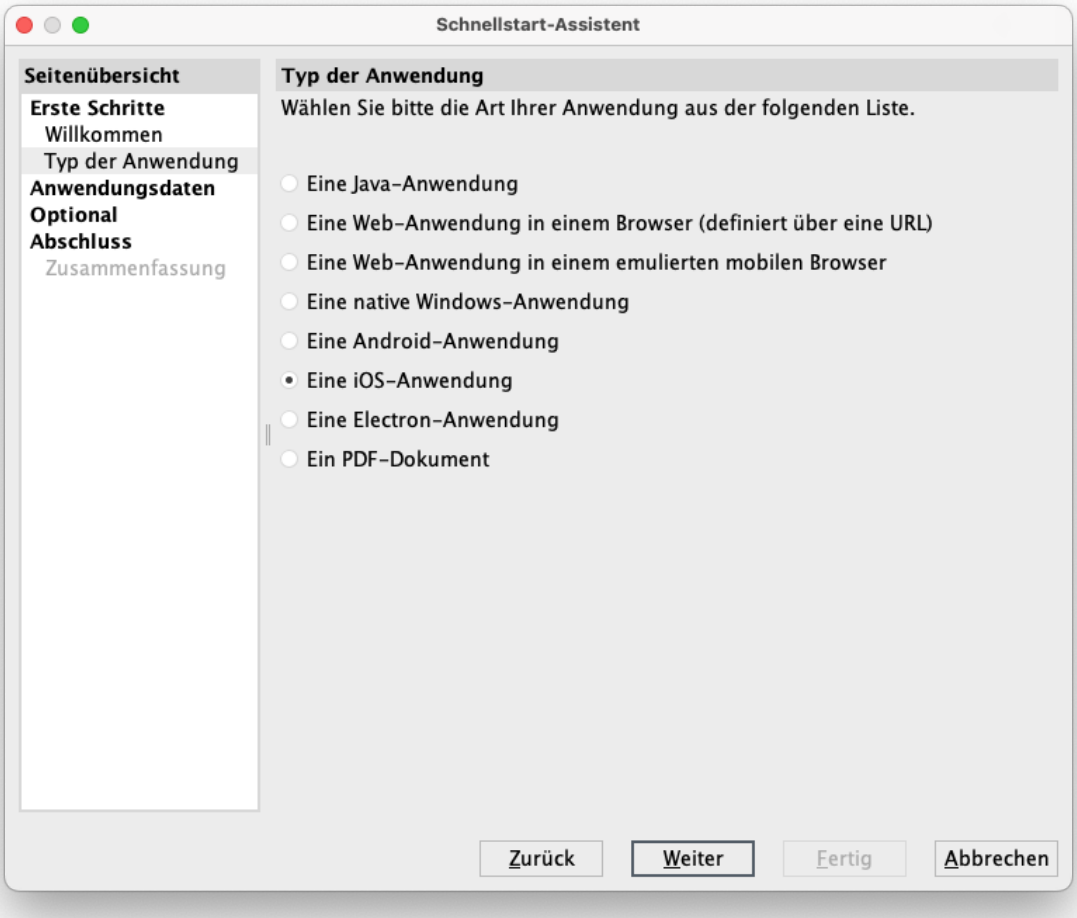


Abbildung 17.6: Dialog des Schnellstart-Assistenten zur Auswahl des Anwendungstyps

- Wählen Sie das echte oder simulierte Gerät aus der Auswahlliste. Falls keine Geräte angezeigt werden, drücken Sie ”Aktualisieren”. Wenn immer noch nichts angezeigt wird, öffnen Sie Xcode und wählen Sie Window→Devices and Simulators um dort sicherzugehen, dass die Einstellungen stimmen.

  Es ist ausreichend, den ersten Teil des Gerätenamens anzugeben, um Tests flexibler zu halten.

  Drücken Sie ”Weiter” um zum nächsten Schritt weiterzugehen.

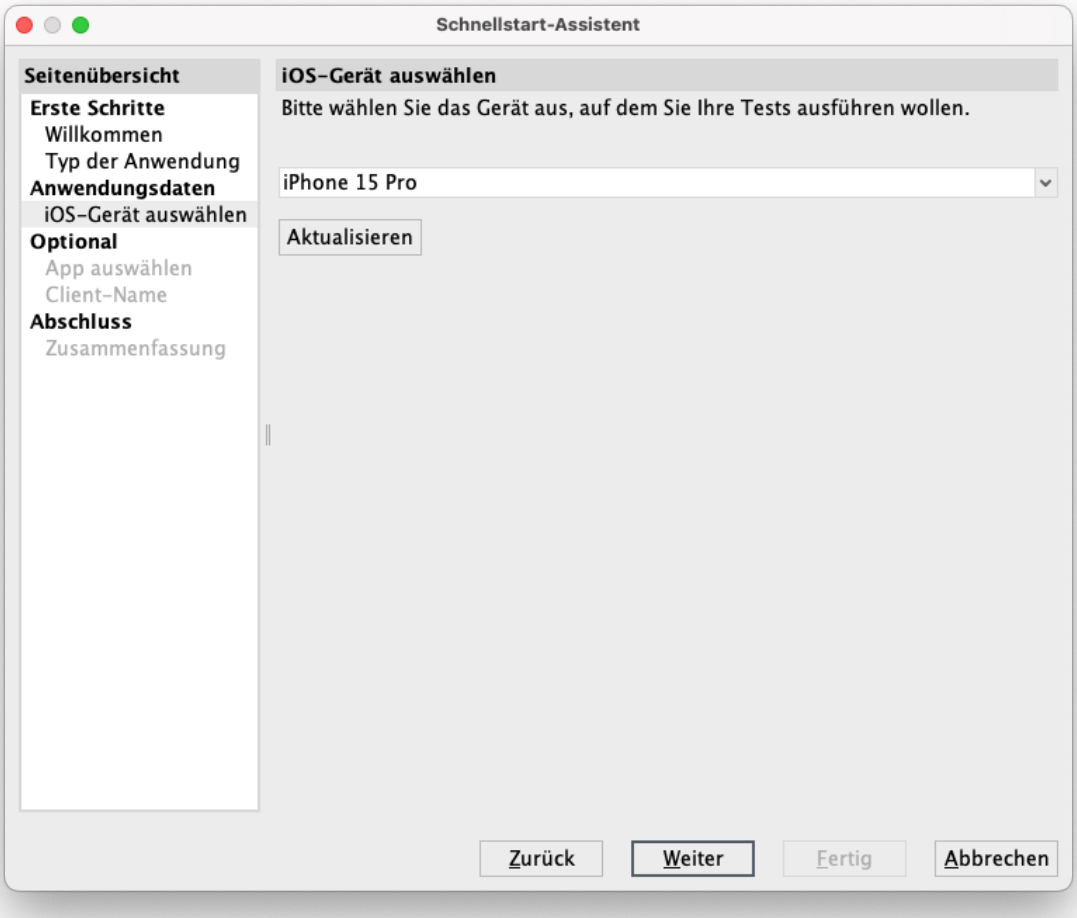


Abbildung 17.7: Dialog des Schnellstart-Assistenten zur Auswahl des Testgeräts

- Als nächsten Schritt können Sie das zu testende iOS-Bundle oder die zu testende .ipa-Datei angeben. Sie können auch direkt eine .zip-Datei, die das App-Bundle enthält, referenzieren. Falls Sie eine App testen wollen, die bereits installiert ist, lassen Sie das Feld leer.

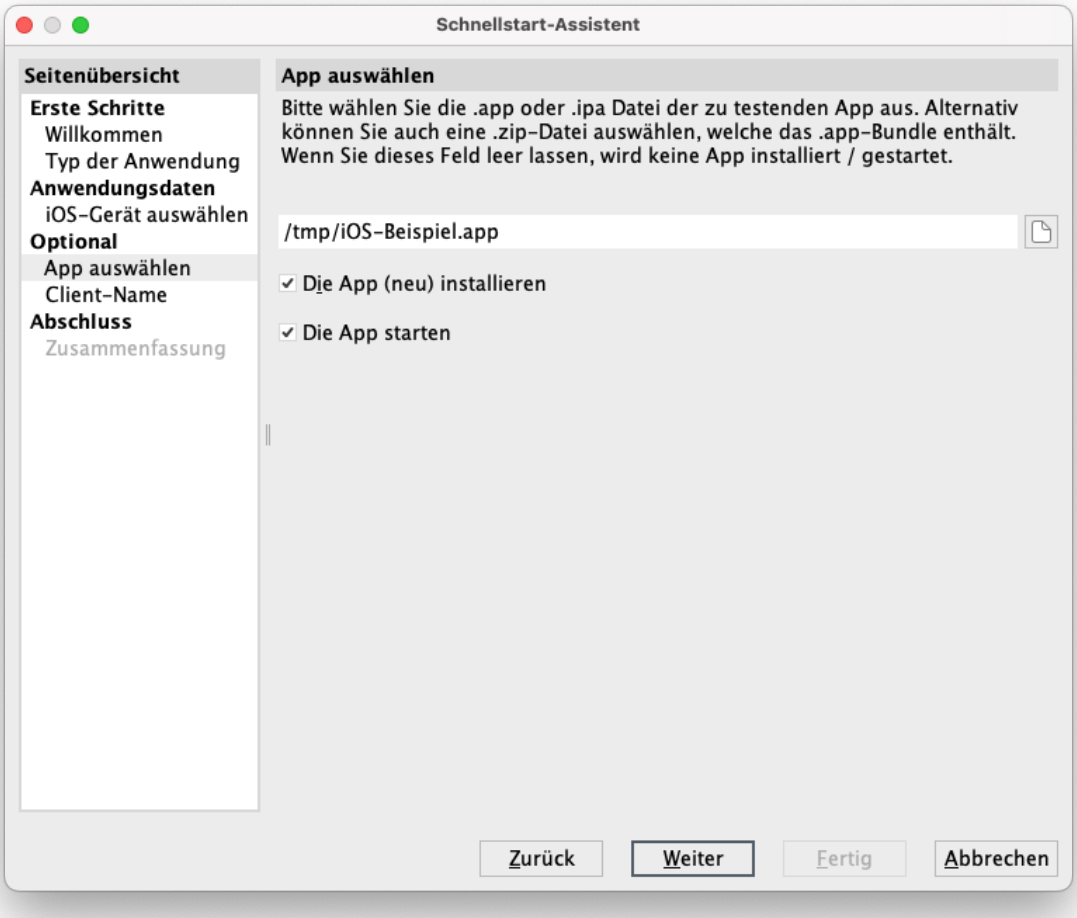


Abbildung 17.8: Dialog des Schnellstart-Assistenten zur Auswahl der App

- Im nächsten Schritt können Sie den vorgeschlagenen Client-Namen übernehmen oder überschreiben. Drücken Sie ”Weiter” beziehungsweise ”Fertig” um den Schnellstart-Assistenten abzuschließen.

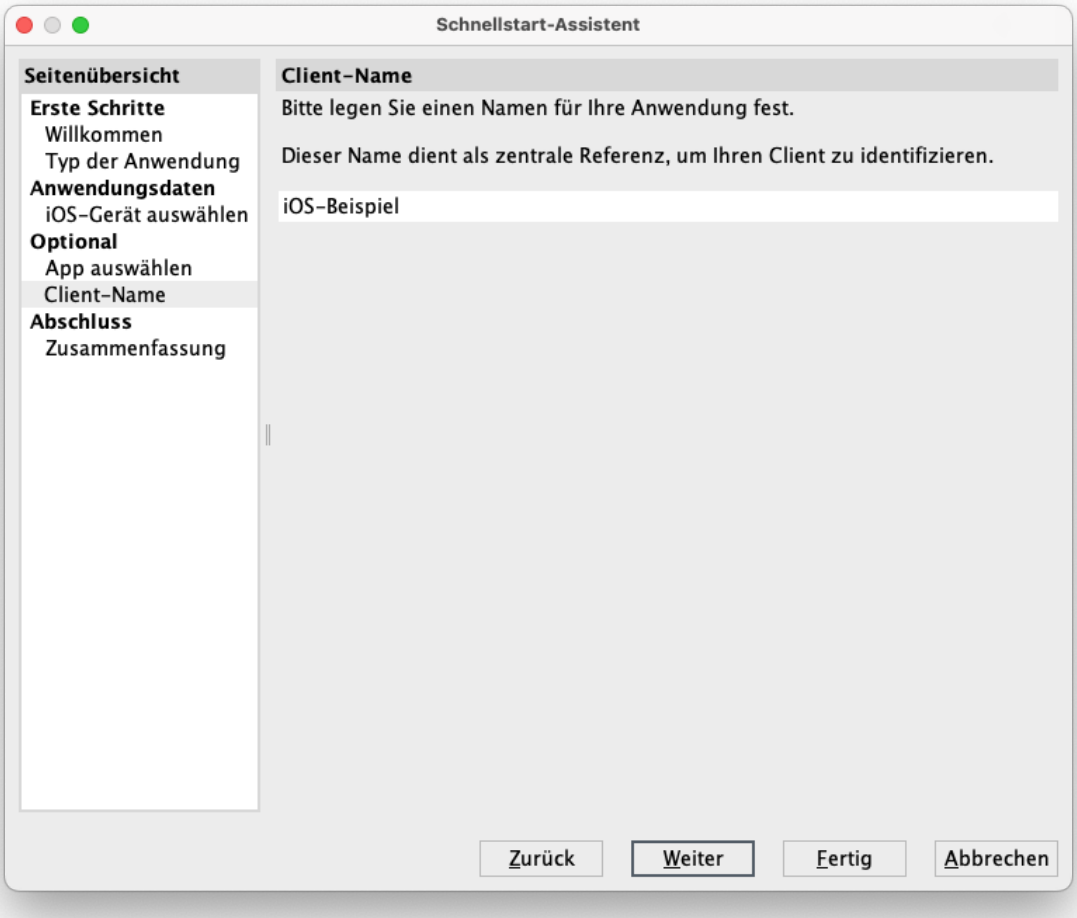


Abbildung 17.9: Dialog des Schnellstart-Assistenten für den Client-Namen

- Als Ergebnis wird die Sequenz Vorbereitung(638) im ”Extras” Knoten der Testsuite erzeugt. Diese Vorbereitung-Sequenz beinhaltet einen Aufruf der Prozedur `qfs.ios.setup.checkEnvironment` aus der Standardbibliothek(183), die prüft, ob das Testsystem ordnungsgemäß eingerichtet ist.

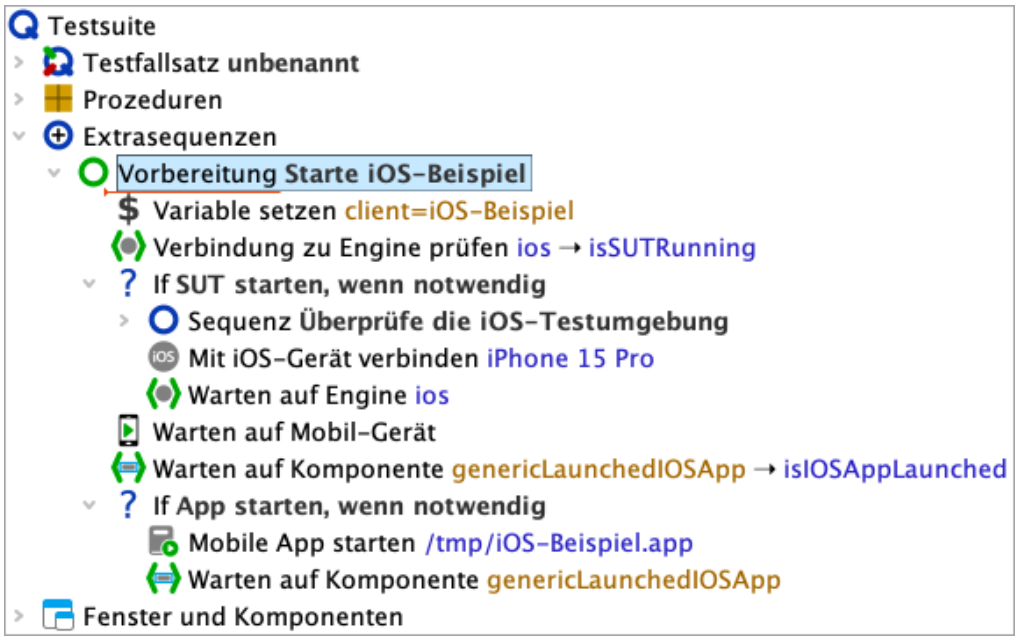


Abbildung 17.10: Vom Schnellstart-Assistenten erstellte iOS-Vorbereitungssequenz

17.5 Aufnamen und Checks bei iOS

- Um eine Aufnahme zu starten, drücken Sie in QF-Test die Schaltfläche  .
- Es wird ein eigenes Aufnahmefenster geöffnet, welches die Oberfläche entweder des Simulators oder des echten Geräts anzeigt. Dieses spezielle Fenster ist notwendig, da es aktuell nicht möglich ist, Aktionen direkt im Simulator oder im echten Gerät aufzuzeichnen.
- Die Oberfläche des (simulierten) Geräts wird laufend in das Aufnahmefenster gespiegelt. Über Buttons können Sie die Größe des angezeigten Bereichs verändern.

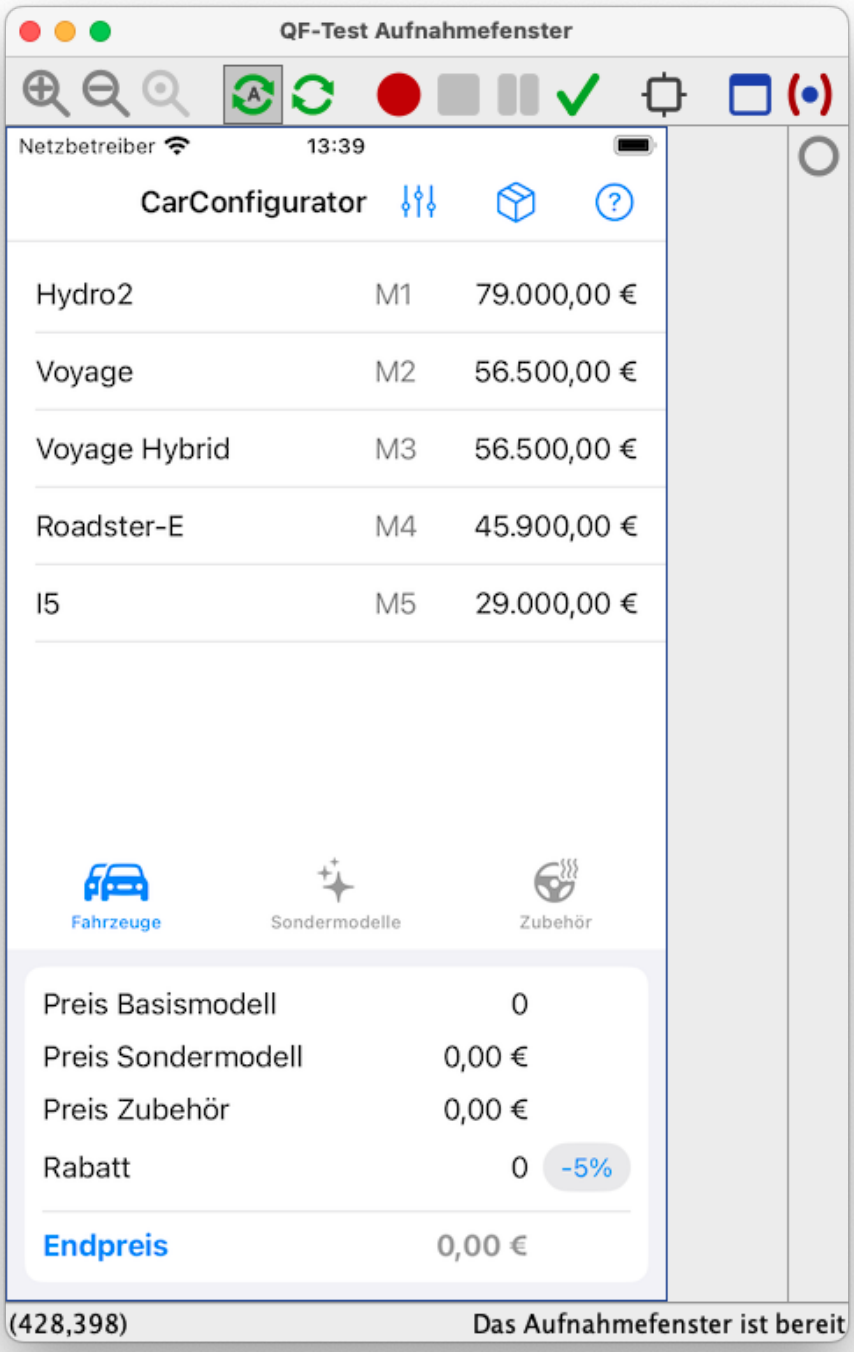


Abbildung 17.11: QF-Test iOS Aufnahmefenster

- Jetzt können Sie Aktionen oder Checks aufnehmen und abspielen.
- In der Toolbar des Aufnahmefensters gibt es die Schaltfläche  um den UI Inspektor zu öffnen, der alle sichtbaren Komponenten mit Größe und Koordinaten anzeigt, siehe UI-Inspektor(108).

17.6 iOS Hilfsprozeduren

Es gibt eine ganze Reihe an verfügbaren Hilfsprozeduren für iOS in der Standardbibliothek(183). Sie befinden sich im dementsprechend benannten Package ”ios”.

Einige sind analog zu den anderen GUI Technologien, es gibt aber einige sehr spezifische für mobiles Testen, z.B. um Wischaktionen oder Gesten durchzuführen, zum Blättern auf dem Bildschirm und um bestimmte Einstellungen für Komponenten vorzunehmen.

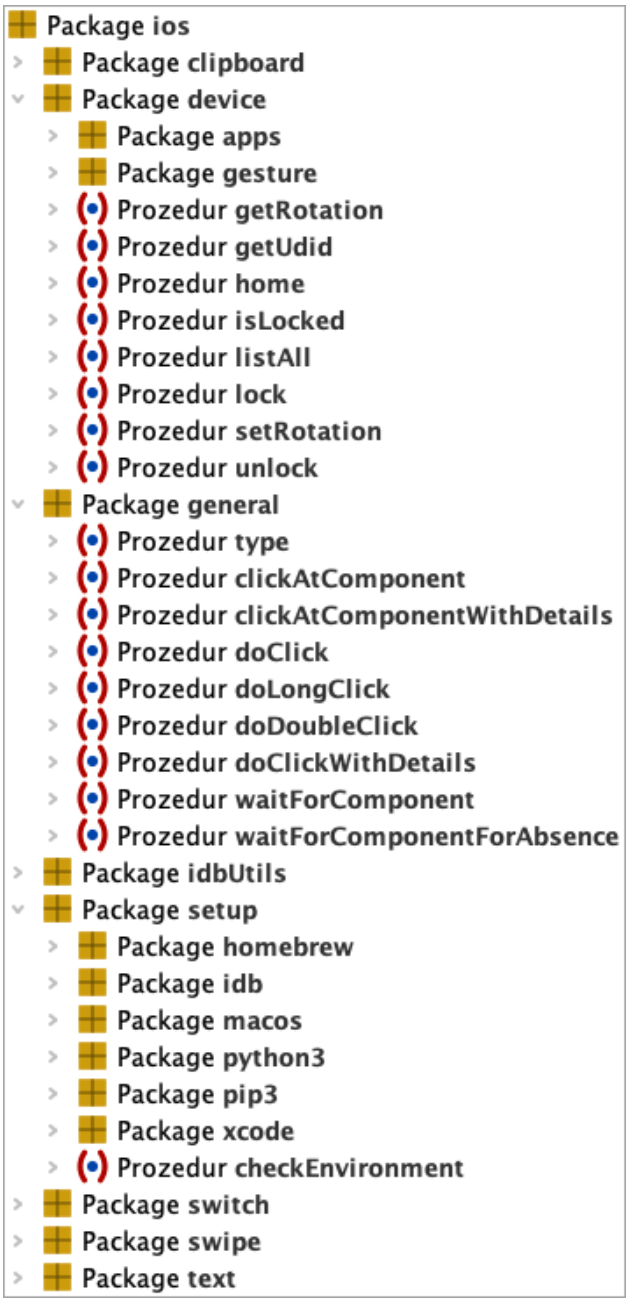


Abbildung 17.12: iOS Hilfsprozeduren

Kapitel 18

Testen von PDF-Dokumenten

4.2+ QF-Test bietet ab Version 4.2 die Möglichkeit, PDF-Dokumente analog zu GUIs zu testen. D.h. QF-Test analysiert die Struktur des PDF-Dokuments und erkennt einzelne Komponenten, deren Eigenschaften getestet werden können.
QF-Test kann mittels Aufnahme und Wiedergabe[39] direkt Events[289] sowie diverse Checks[290] aufnehmen und wiedergeben.

Video Video:

'PDF-Dokumente testen mit QF-Test'
https://www.qftest.com/de/yt/pdf-dokumente-testen-42.html

18.1 PDF-Client

Das zu testende PDF-Dokument wird von QF-Test in einen Viewer geladen, der von QF-Test als Client-Prozess gestartet wird.

18.1.1 PDF-Client starten

Mit Hilfe des Schnellstart-Assistenten lässt sich die passende Setup-Sequenz erstellen, wobei als Typ der Anwendung "PDF-Dokument" zu wählen ist (vgl. Kapitel 3[32]). Dies ermöglicht den einfachen Start des Viewers. Als konkreter Startknoten wird hierbei der PDF-Client starten[741] Knoten verwendet.

18.1.2 Das Fenster des PDF-Client

Im Fenster des PDF-Client befindet sich links eine Navigationsspalte mit einer Übersucht der Seiten des PDF-Dokuments.
Im rechten Fensterbereich wird die aktuell selektierte Seite angezeigt.
Der folgende Screenshot zeigt den PDF-Client mit einem geöffneten Demo PDF-Dokument.

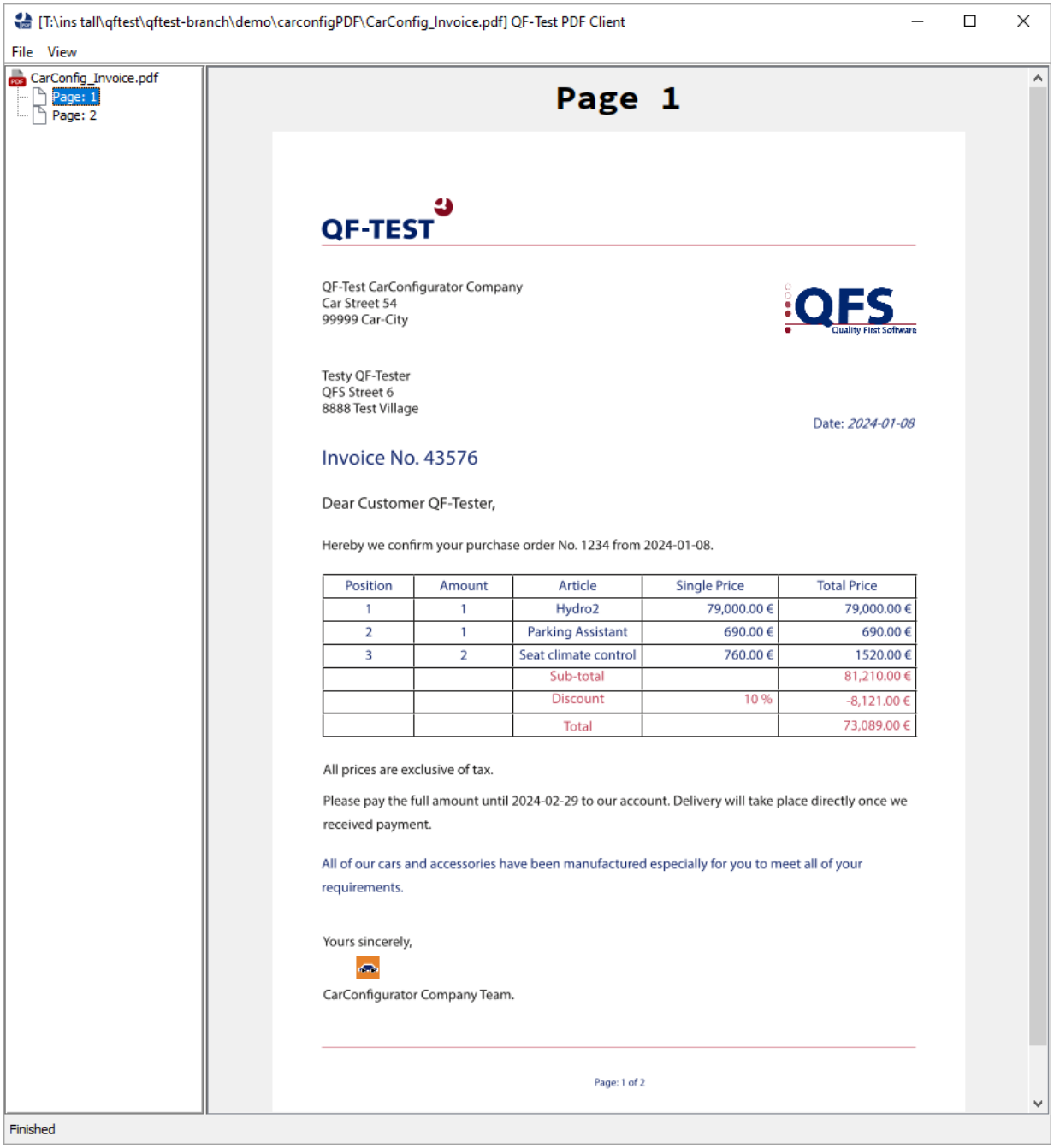


Abbildung 18.1: PDF-Client Hauptfenster mit geöffnetem PDF-Dokument

18.2 PDF Events

Um während der Testausführung z.B. ein anderes Dokument zu öffnen oder die Seite des geöffneten PDFs zu ändern, können Auswahl[793] Knoten verwendet werden. Diese Aktionen lassen sich direkt im Aufnahme-Modus aufzeichnen. Im Auswahl Knoten muss als Komponente das ”Window” des PDFs angegeben werden.

18.2.1 PDF-Dokument öffnen

Es ist möglich, während der Ausführungszeit ein anderes PDF-Dokument zu öffnen. Dazu muss im Detail[795]-Attribut des Auswahl[793] Knotens das `open:` Event angegeben werden.

Hierbei kann nun der Pfad zum PDF-Dokument angegeben werden. Relative Pfade werden relativ zum aktuellen Verzeichnis der Testsuite aufgelöst.

```
open:C:\Users\qfs\meinPDFDokument.pdf
```

Beispiel 18.1: Laden eines PDF-Dokuments

Wenn das Dokument nicht gefunden oder geladen werden kann wird eine `TestException`[958] geworfen.

18.2.2 Seite wechseln

Um auf eine bestimmte Seite zu wechseln, kann im Detail[795]-Attribut des Auswahl[793] Knotens das `goto:` Event angegeben werden.

Hier kann analog zu dem Seitennummer[742]-Attribut, die Seite als Zahl für die Seitennummer bzw. in Anführungszeichen für den Seitentitel angegeben werden.

```
goto:3 bzw. goto:"Einleitung"
```

Beispiel 18.2: Öffnen einer bestimmten Seite

Sollte die gewünschte Seite nicht zur Verfügung stehen, so wird eine `PageNotFoundException` geworfen.

18.3 Checks für PDF-Komponenten

Für PDF-Komponenten (vgl. Abschnitt 18.4(295)) stehen folgende Checks zur Verfügung, die auch direkt mittels dem Check-Aufnahme-Modus aufgezeichnet werden können:

18.3.1 Check Text

Der Check Text Knoten ist in Check Text(806) beschrieben. Für PDF-Text-Komponenten gibt es bei Text-Komponente zwei Check-Typen: ”default” und ”Text positioniert”.

Zeilenumbrüche sind im PDF-Dokument nicht enthalten, Leerzeichen nicht zwingend. Die Abstände ergeben sich aus den Koordinaten der einzelnen Buchstaben. Der Check-Typ ”default” prüft den Text wie er im PDF-Dokument abgespeichert ist - ohne Zeilenumbrüche und eventuell ohne Leerzeichen, wenn keine enthalten sind. QF-Test errechnet aus den Koordinaten der einzelnen Buchstaben wo die Zeilenumbrüche angezeigt werden und wo Abstände, die auf Leerzeichen schließen lassen, vorhanden sind. Der Check-Typ ”Text positioniert” greift auf diesen aufbereiteten Text zu.

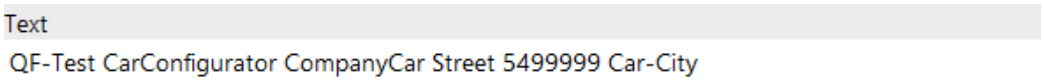


Abbildung 18.2: Check Text ’default’ Aufnahme

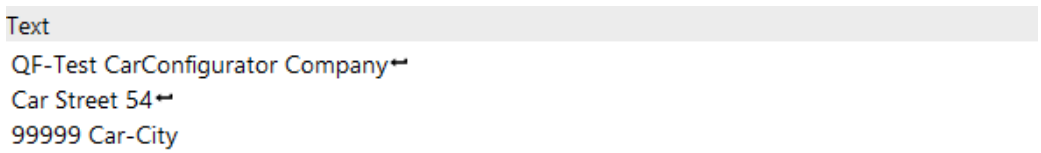


Abbildung 18.3: Check Text ’Text positioniert’ Aufnahme

4.4+ Zusätzlich kann der gesamte Text der Seite auf der Main Stage mit den Check-Typen ”Text (gesamte Seite)”, ”Text positioniert (gesamte Seite)”, ”Text als Elemente (gesamte Seite)” sowie ”Text positioniert als Elemente (gesamte Seite)” geprüft werden.
Dabei werden alle Text-Komponenten der Seite sortiert nach Y/X-Position aufgenommen. Die Check-Typen unterscheiden sich dabei, ob der Text positioniert/aufbereitet wird (siehe oben) oder nicht und ob die einzelnen Text-Komponenten als Check Elemente(818) oder als gemeinsamer Check Text(806) aufgenommen werden.

Elemente

	Text	Regexp
0	QF-Test CarConfigurator CompanyCar Street 5499999 Car-City	☐
1	Testy QF-TesterQFS Street 68888 Test Village	☐
2	Date: 2018-02-01	☐
3	Invoice No. 43576	☐
4	Dear Customer QF-Tester,	☐
5	Hereby we confirm your purchase order No. 1234 from 2018-02	☐
6	PositionAmountArticleSingle PriceTotal Price	☐
7	11Minigolf15,000.00 €15,000.00 €21Alloy rims900.00 €900.00	☐
8	Sub-total15,952.00 €	☐
9	Discount10 %-1,595.20 €	☐
10	Total	☐
11	14,356,80 €	☐
12	All prices are exclusive of tax.Please pay the full amount until 201	☐
13	All of our cars and accessories have been manufactured especially	☐
14	Yours sincerely,CarConfigurator Company Team.	☐
15	Page: 1 of 2	☐

Abbildung 18.4: Check Elemente ’Text als Elemente (gesamte Seite)’ Aufnahme

Elemente

	Text	Regexp
0	QF-Test CarConfigurator Company↩...	☐
1	Testy QF-Tester↩...	☐
2	Date: 2018-02-01	☐
3	Invoice No. 43576	☐
4	Dear Customer QF-Tester,	☐
5	Hereby we confirm your purchase order No. 1234 from 2018-02·	☐
6	Position Amount Article Single Price Total Price	☐
7	1 1 Minigolf 15,000.00 € 15,000.00 €↩...	☐
8	Sub-total 15,952.00 €	☐
9	Discount 10 % -1,595.20 €	☐
10	Total	☐
11	14,356,80 €	☐
12	All prices are exclusive of tax.↩...	☐
13	All of our cars and accessories have been manufactured especially	☐
14	Yours sincerely,↩...	☐
15	Page: 1 of 2	☐

Abbildung 18.5: Check Elemente ’Text positioniert als Elemente (gesamte Seite)’ Aufnahme

Text

QF-Test CarConfigurator CompanyCar Street 5499999 Car-City↩
Testy QF-TesterQFS Street 68888 Test Village↩
Date: 2018-02-01↩
Invoice No. 43576↩
Dear Customer QF-Tester,↩
Hereby we confirm your purchase order No. 1234 from 2018-02-01.↩
PositionAmountArticleSingle PriceTotal Price↩
11Minigolf15,000.00 €15,000.00 €21Alloy rims900.00 €900.00 €32Mats26.00 €52.00 €↩
Sub-total15,952.00 €↩
Discount10 %-1,595.20 €↩
Total↩
14,356,80 €↩
All prices are exclusive of tax.Please pay the full amount until 2018-03-01 to our account. Delive
All of our cars and accessories have been manufactured especially for you to meet all of your rec
Yours sincerely,CarConfigurator Company Team.↩
Page: 1 of 2↩

Abbildung 18.6: Check Text ’Text (gesamte Seite)’ Aufnahme

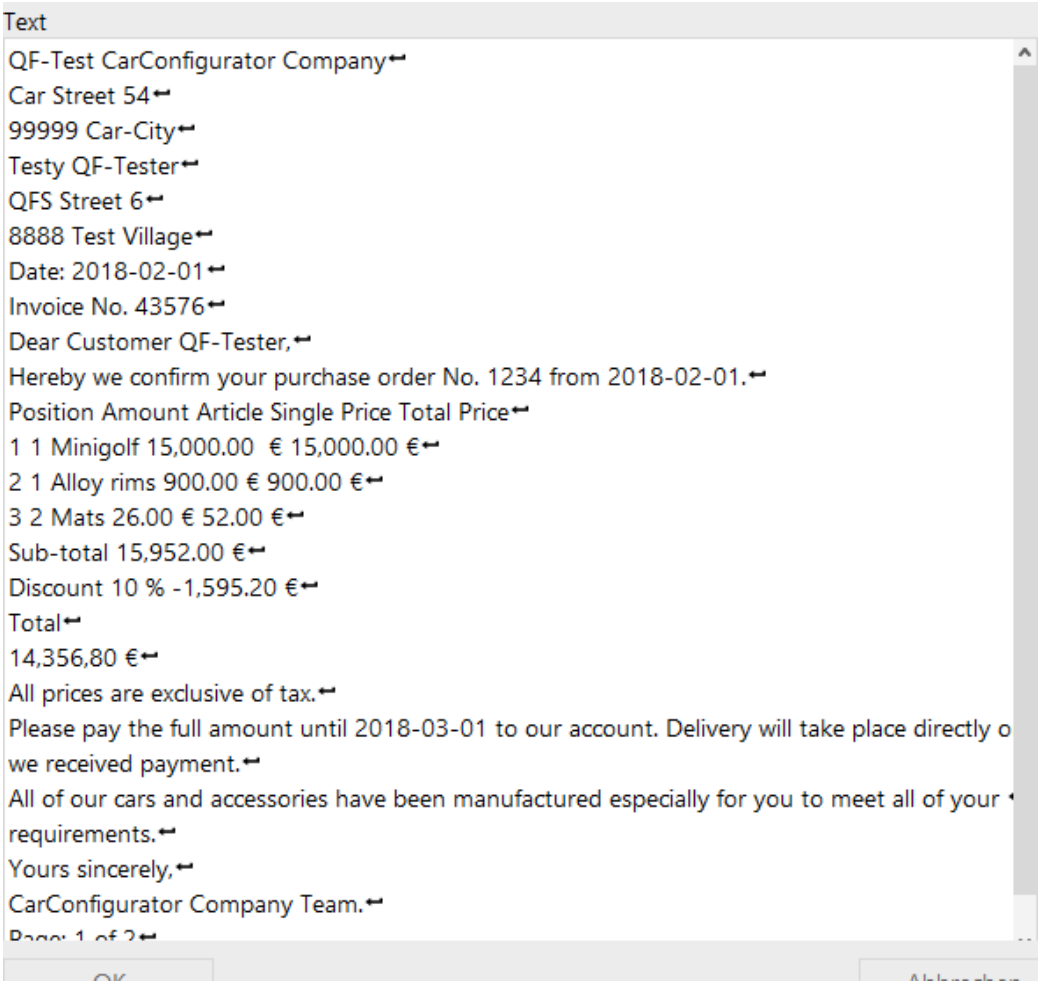


Abbildung 18.7: Check Text ’Text positioniert (gesamte Seite)’ Aufnahme

18.3.2 Check Abbild

Der Check Abbild Knoten ist in Check Abbild(828) beschrieben. Für alle Komponenten Typen gibt es den Check-Typ ”default”.

Der Check-Typ ”default” prüft das Objekt, wie es auf der PDF-Seite angezeigt wird, ggf. skaliert und mit überdeckenden Objekten oder Objektteilen. Das aufgenommene Bild entspricht also der tatsächlichen Darstellung.

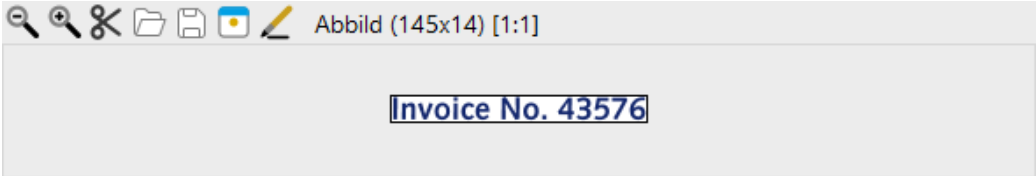


Abbildung 18.8: Check Abbild ’default’ Aufnahme eines Text Objekts

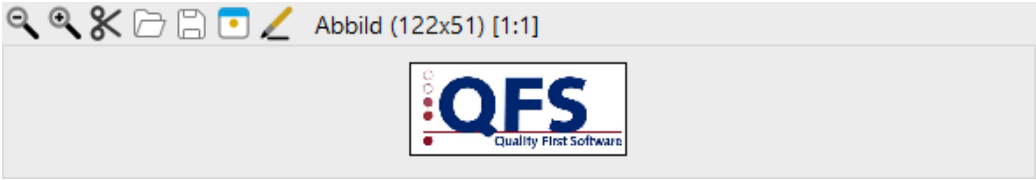


Abbildung 18.9: Check Abbild ’default’ Aufnahme eines Image Objekts

In PDF-Dokumenten können auch echte Bilder eingebettet werden. Dieses Image kann für die Anzeige auf der PDF-Seite skaliert werden. Für echte Bilder werden noch weitere Check-Typen angeboten:

Der Check-Typ ”unskaliert” prüft das in der Datei originale eingebettete unskalierte Bild.

Abbildung 18.10: Check Abbild ’unskaliert’ Aufnahme eines Image Objekts

Der Check-Typ ”skaliert” prüft das auf der PDF-Seite angezeigte, ggf. skalierte Bild ohne überdeckende Objekte. Somit können auch teilverdeckte Abschnitte dieser Bilder geprüft werden.

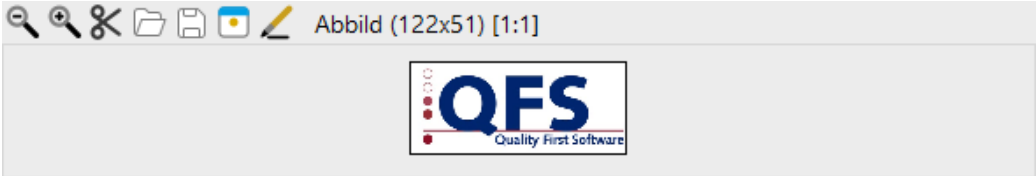


Abbildung 18.11: Check Abbild 'skaliert' Aufnahme eines Image Objekts

18.3.3 'Check Font'

Der Check Text Knoten mit dem Check-Typ "text_font" erlaubt das Überprüfen des Fonts.

Innerhalb einer PDF-Text-Komponente können verschiedene Fonts verwendet werden. 'Check Font' gibt für eine Text-Komponente den Font zurück, der mehrheitlich verwendet wird.

18.3.4 'Check Font-Größe'

Der Check Text Knoten mit dem Check-Typ "text_fontsize" erlaubt das Überprüfen der Fontgröße.

Innerhalb einer PDF-Text-Komponente können verschiedene Fontgrößen verwendet werden. 'Check Font-Größe' gibt für ein Textobjekt die Fontgröße zurück, der mehrheitlich verwendet wird.

18.4 PDF Komponententypen

QF-Test erkennt folgende Objekttypen:

PDF Objekttyp	QF-Test Komponententyp	Bemerkung
Text	Text oder Label	Ansammlung von Buchstaben, denen Font und Fontsize zugeordnet sind.
Image	Graphics	Ansammlung von Pixeln. Kann auch Buchstabenform haben.
Shader und Vektoren	Graphics	Ansammlung von Vektoren, die entweder geometrische Figuren oder auch Buchstabenformen darstellen.
Main stage	MainPanel	Die zugrunde liegende Seite, auf der alle Objekte dargestellt werden.

Tabelle 18.1: Unterstützte PDF-Objekte

QF-Test kann die erkannten PDF-Objekte farbig umranden, wenn diese Funktion im Menü View -> Show components oder das Tastenkürzel STRG-T aktiviert wird. Bei der Aufnahme und Wiedergabe muss diese Funktion deaktiviert sein, da die Rahmen ansonsten bei Abbild-Vergleichen mit aufgenommen werden.
Für die einzelnen Objekttypen gilt folgender Farbcode:

Farbe	PDF Objekttyp
Rot	Text
Blau	Image
Grün	Shader und Vektoren
Türkis	Main stage

Tabelle 18.2: Farbcode der PDF-Objekte

18.5 PDF Komponentenerkennung

QF-Test stellt die PDF-Objekte als Swing-Komponenten dar, auf die über die Swing-API z.B. aus SUT-Skripten heraus zugegriffen werden kann (vgl. Kapitel 11 (186))

Die Basisinformationen für die Identifizierung der PDF-Objekte auf der Seite sind, wie bei allen QF-Test Komponenten, ihre Klasse, Geometrie und Strukturinformationen (Index). Für Textkomponenten stehen außerdem über Weitere Merkmale der mehrheitlich zugeordnete Font und die mehrheitlich verwendete Fontgröße zur Verfügung. Für Image-Objekte wird in Weitere Merkmale der Image hash und für Shader-Objekte der Shader-Typ mit aufgenommen.

Des weiteren versucht QF-Test bei einem Textobjekt anhand seiner Merkmale zu erkennen, ob es sich um eine Überschrift oder Beschriftung handelt. In diesem Fall erhält das Textobjekt die QF-Test Klasse ”Label”. Über den QF-Test Standardalgorithmus für `qfs:label*`-Varianten[74] in Weitere Merkmale kann dieses Label anderen QF-Test Komponenten zugeordnet werden.

Da der Standardalgorithmus zur Erkennung von Überschriften und Beschriftungen mit Annahmen und Wahrscheinlichkeiten arbeiten muss, kann es auch zu Fehltreffern (false positive) oder Nichterkennung kommen. In diesem Fall können Resolver (Abschnitt 54.1[1154]) zur Verbesserung der Erkennung eingesetzt werden. Auch die Zuordnung der Label-Komponente zu anderen Komponenten kann über Resolver gesteuert werden.

Kapitel 19

Testen von Barrierefreiheit

9.0.0+

QF-Test unterstützt für Web-Anwendungen das Testen von Barrierefreiheit auf Basis der WCAG Richtlinien (https://www.wcag.com oder auch https://www.barrierefreiheit-dienstekonsolidierung.bund.de/Webs/PB/DE/gesetze-und-richtlinien/wcag/wcag-artikel.html). Für diese Richtlinien wurde von Deque Systems eine Bibliothek (axe-core) entwickelt, die Methoden bereitstellt, die die Umsetzung eines Teiles dieses Regelwerks kontrollieren (https://dequeuniversity.com/rules/axe/html/).

QF-Test stellt die Prozedur runAxeChecks zur Verfügung, die Zugriff auf die axe-core Bibliothek gewährt. Damit kann die gesamte Funktionalität von axe-core in QF-Test genutzt werden - ohne zusätzlichen Programmieraufwand. Über Parameter wird gesteuert, welche Regeln angewendet und welche Bereiche einer Webseite geprüft werden sollen. Hierbei bietet QF-Test folgende Annehmlichkeiten:

- Übersichtliche HTML-Reports sowie detaillierte QF-Test Protokolle
- Protokollierung fehlerhafter Elemente mit umfangreichen Informationen und Vorschlägen zur Fehlerbehebung
- Erstellung von Bildschirmabbildern mit Hervorhebung fehlerhafter Elemente zur einfachen Identifizierung

Da QF-Test den vollen Zugriff auf HTML-Elemente hat und Benutzeraktionen simulieren kann, hat es das Potential, über axe-core hinausgehende Prüfungen zur Verfügung zu stellen. Im ersten Schritt bieten wir eine Überprüfung von Farbkontrasten für einfache grafische Elemente, wie etwa Icons, über die Prozedur checkColorContrastSimpleGraphics an. Sie befindet sich im gleichen Package in der Standardbibliothek wie die oben genannte Prozedur `runAxeChecks` und folgt den gleichen Prinzipien bei der Protokollierung.

Hinweis

Der Ausdruck *a11y*, der in den Namen der Prozeduren verwendet wird, ist eine gemein-

hin benutzte Abkürzung für den Begriff Barrierefreiheit
(engl. ”Accessibility”, ”A” + 11 Buchstaben + ”y”).

19.1 Allgemeine Parameter der Check-Funktionen

Die bereitgestellten Methoden zum Testen der Barrierefreiheit sind über verschiedene Parameter einstellbar, um beispielsweise den Umfang des Tests oder die Art der Protokollierung festzulegen.
Diese Parameter werden im Folgenden genauer erklärt.

scope
Hier wird die QF-Test ID[931] der Komponente[930] angegeben, innerhalb derer die Checks angewandt werden sollen. Die Tests werden nur auf diese Komponente und ihre Kindkomponenten angewendet.
Default: `genericDocument`, also die gesamte Seite

genericClassesToSkip
Die mit einem Komma getrennten Namen von Generische Klassen[1329], die bei den Tests übergangen werden sollen. Diese Klassen und eventuelle Unterklassen werden bei den Tests nicht überprüft.
Default: `-`

skipInvisibleElements
Legt fest, ob die Checks bei unsichtbaren Elementen übersprungen werden sollen.
Default: `true`

showSuccessfulChecks
Wird dieser Parameter auf `true` gesetzt, werden auch erfolgreiche Checks als Information im Protokoll gelistet.
Default: `false`

showSkippedChecks
Wird dieser Parameter auf `true` gesetzt, werden Checks, die nicht vollständig ausgeführt werden konnten, als Warnung im Protokoll gelistet. Ein Grund hierfür kann zum Beispiel die Überdeckung des zu prüfenden Elementes durch ein anderes sein.
Default: `true`

logOverviewScreenshot
Legt fest, ob im Fehlerfall ein Bildschirmabbild für den Überblick über die fehlerhaften Elemente erzeugt werden soll. Das Bildschirmabbild umfasst den definierten `scope`, kann also auch die gesamten Seite beinhalten. Auf dem Bild

werden Elemente mit Fehlern rot umrandet, Elemente mit Warnungen gelb.
Default: `true`

allowedHeightOfOverviewScreenshot
Legt die maximale Höhe (in Pixeln) für das Bildschirmabbild für den Überblick fest. Wirkt sich nur auf das Bildschirmabbild aus, wenn dieses größer als das Browserfenster ist. Das Limit ist aus Speichergründen erforderlich.
Default: `2000`

logElementScreenshots
Legt fest, ob im Fehlerfall ein Abbild der einzelnen Elemente erzeugt werden soll.
Wenn der Wert `all` ist, wird für jedes einzelne fehlerhafte Element ein Abbild erzeugt.
Wenn der Wert `first` ist, wird nur für das erste Element eines Fehlertyps ein Abbild erzeugt.
Wenn der Wert `none` ist, werden keine Abbilder erzeugt.
Auf dem Bild wird das Elemente bei Fehlern rot umrandet, bei Warnungen gelb.
Die Gesamtanzahl der Abbilder wird durch den folgenden Parameter `allowedNumberOfElementScreenshots` begrenzt.
Default: `all`
Mögliche Werte: `all, first, none`

allowedNumberOfElementScreenshots
Bestimmt die maximale Anzahl an Einzelbildern von Elementen im Fehlerfall. Das Limit ist aus Speichergründen erforderlich.
Default: `10`

logElementSmartIdToMessage
Legt fest, ob die QF-Test spezifische SmartID[81] des überprüften Elementes im Protokoll aufgeführt werden soll.
Die SmartID hilft beim addressieren der fehlerhaften Komponenten innerhalb QF-Tests.
Default: `false`

squashCheckResultsWithSameMessage
Wird dieser Parameter auf `true` gesetzt, werden Elemente mit der gleichen Fehlermeldung im Protokoll unter einem einzigen Fehler (oder Warnung) zusammengefasst.
Default: `false`

19.2 Axe-Checks mit QF-Test

Fehler beim Überprüfen einer Website mit axe finden sich im Protokoll unter folgendem Fehlercode:
QF-Test-Fehlercode: `ERR_AXE-CORE_CHECKS`

Hinweis QF-Test erweitert axe um die Funktion, Elemente in geschlossenen Shadow-Roots auf Barrierefreiheit zu überprüfen. Dies ist allerdings nur bei Verwendung des CDP-Driver Verbindungsmodus[1130] möglich.

19.2.1 Parameter des Axe-Checks

Die Prozedur runAxeChecks hat die folgenden zusätzlichen Parameter:

rules
Die mit einem Komma getrennten rule-IDs der Axe-Regeln oder die von Axe definierten Tags (https://dequeuniversity.com/rules/axe/html). Beispiele:

- button-name
- button-name,color-contrast,aria-required-attr
- wcag2aa
- wcag2aa,best-practice,cat.aria

Wird dieser Parameter leer gelassen werden alle Regeln überprüft.
Default: `wcag2a,wcag2aa,wcag21a,wcag21aa,wcag22aa`, also die Tags der für die Erfüllung der WCAG relevanten Axe-Regeln

rulesToSkip
Ist der `rules`-Parameter leer oder mit Tags befüllt, so ist es hier möglich einzelne Regeln von der Überprüfung auszuschließen. Hierfür muss eine mit Komma getrennte Liste an IDs von Axe-Regeln angegeben werden. Beispiele:

- button-name
- button-name,color-contrast,aria-required-attr

Wird dieser Parameter leer gelassen werden keine gesonderten Regeln von der Überprüfung ausgeschlossen.
Default: –

19.2.2 Die ”impact”-Bewertung von axe-core

Die Entwickler von axe-core vergaben jeder einzelnen Regel einen ”impact”. Dieser Wert wird von QF-Test in den Fehlermeldungen zu den Regeln aufgelistet und quantifiziert die Auswirkung eines Problems auf einen Benutzer mit einer Behinderung. In aufsteigender Reihenfolge gelistet (nach Schwere der Auswirkung) gibt es folgende Kategorien:

Minor: niedrige Priorität
Ein lästiger oder ärgerlicher Fehler.

Moderate: mittlere Priorität
Bereitet Schwierigkeiten für eingeschränkte Nutzer, hindert diese jedoch im Allgemeinen nicht am Zugriff auf grundlegende Funktionalitäten.

Serious: hohe Priorität
Führt zu schwerwiegenden Barrieren für Menschen mit Behinderung und hindert diese ganz oder teilweise am Zugang zu grundlegenden Funktionen oder Inhalten.

Critical: oberste Priorität
Das Problem blockiert Menschen mit Behinderung absolut bei der Nutzung der grundlegenden Funktionalitäten der Seite und dem Zugriff auf die Inhalte.

Hinweis

Die ”impact”-Bewertung erlaubt eine Priorisierung bei der Behebung der Probleme. Für ein Einhalten der WCAG-Richtlinien sind allerdings *alle* Fehler zu beheben - auch die Fehler von niedriger Priorität!

19.3 Farbkontrast-Check für einfache Grafikobjekte

Die WCAG schreibt für Bilder von großem Text, Komponenten der Benutzeroberfläche und informationstragende Grafiken einen Mindestfarbkontrast von 3:1 vor. (§§1.4.3, 1.4.11 WCAG 2.2)
Der Farbkontrast-Check prüft den Farbkontrast einfacher Grafikelemente (wie etwa Icons) gegen die automatisch ermittelte Hintergrundfarbe.

Fehler beim Überprüfen von Grafikelementen einer Website finden sich im Protokoll unter folgendem Fehlercode:
QF-Test-Fehlercode: `ERR_COLOR_CONTRAST_SIMPLE_GRAPHICS`

19.3.1 Parameter des Farbkontrast-Checks

Die Prozedur checkColorContrastOfGraphic hat den folgenden zusätzlichen Parameter:

genericClass

Der Name einer Generischen Klasse (siehe Generische Klassen[1329]). Alle Elemente dieser Klasse innerhalb des `scope` werden auf ihren Farbkontrast überprüft.
Default: `Graphics`

19.4 A11y Protokolle und Reports

Für das Arbeiten mit dem Protokoll und die Generierung eines Reports (Reports und Testdokumentation[330]) gibt es bei Barrierefreiheitstests eigene Tipps, Tricks und Besonderheiten.

19.4.1 Arbeiten mit dem Protokoll

Nach jedem Barrierefreiheitstest wird ein Protokoll angelegt, dass zur Fehleranalyse verwendet werden kann.

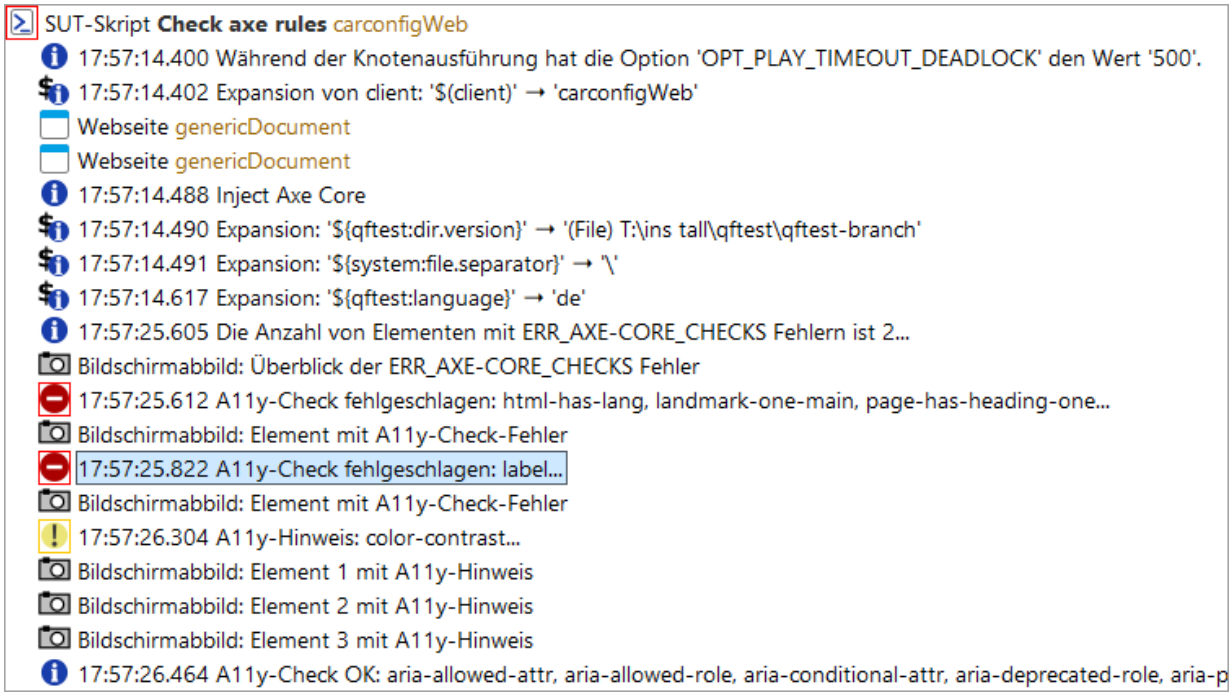


Abbildung 19.1: Ausschnitt aus dem Protokoll eines axe-Barrierefreiheitstests

In der folgenden Abbildung ist die komplette Meldung des ausgewählten Fehlers zu sehen:

A11y-Check fehlgeschlagen: label
Fehlercode: ERR_AXE-CORE_CHECKS
impact critical: label

Element:
URL: file:///T:/ins%20tall/qftest/qftest-branch/demo/carconfigWeb/html/CarConfig.htm?lang=de#
XPath: /html/body/div/div/table/tbody/tr[4]/td[2]/input
SmartID: #INPUT:TEXT:name=DiscountValue_input
Selector: "#DiscountValue_input"

Korrigiere mindestens einen der folgenden Punkte:
Das <form>-Element besitzt kein implizites <label>-Element.
Das <form>-Element besitzt kein explizites <label>.
Es existiert kein aria-label-Attribut oder das Attribut ist leer.
Das aria-labelledby-Attribut existiert nicht oder referenziert ein Element, das nicht existiert, nicht sichtbar oder leer ist.
Element hat kein title-Attribut.
Element hat kein Platzhalterattribut.
Die Standardsemantik des Elements wurden nicht mit der Rolle role="none" oder role="presentation" überschrieben.

Abbildung 19.2: Fehlermeldung zum obig ausgewählten Fehler

In Fehlermeldungen werden Elemente gelistet, die bestimmte Accessibility-Kriterien nicht erfüllen. Der zugehörige Fehler und Zusatzinformationen, wie etwa Lösungsvorschläge, werden in der Fehlermeldung beschrieben.
Für Elemente, die aufgrund unterschiedlicher Probleme, wie etwa die Verdeckung durch ein anderes Element, nicht auf eine bestimmte Regel überprüft werden konnten, werden Warnungen protokolliert.
Je nachdem, auf welchen Wert der Parameter `showSuccessfulChecks` gesetzt wurde, werden auch erfolgreich durchgeführte Checks als Information im Protokoll aufgeführt.

Hinweis

QF-Test logt neben Abbildern auch verschiedene Identifikatoren der Elemente, wie etwa den X-Path oder bei Bedarf (Parameter: `logElementSmartIdToMessage`) die SmartID[81]. Die SmartID kann verwendet werden, um das Element innerhalb QF-Tests anzusprechen. Der X-Path kann verwendet werden, um das Element im Browser mittels Entwicklertools zu finden.

Zudem erzeugt QF-Test einen Screenshot der getesteten Seite, auf dem fehlerhafte und übersprungene Elemente hervorgehoben werden.

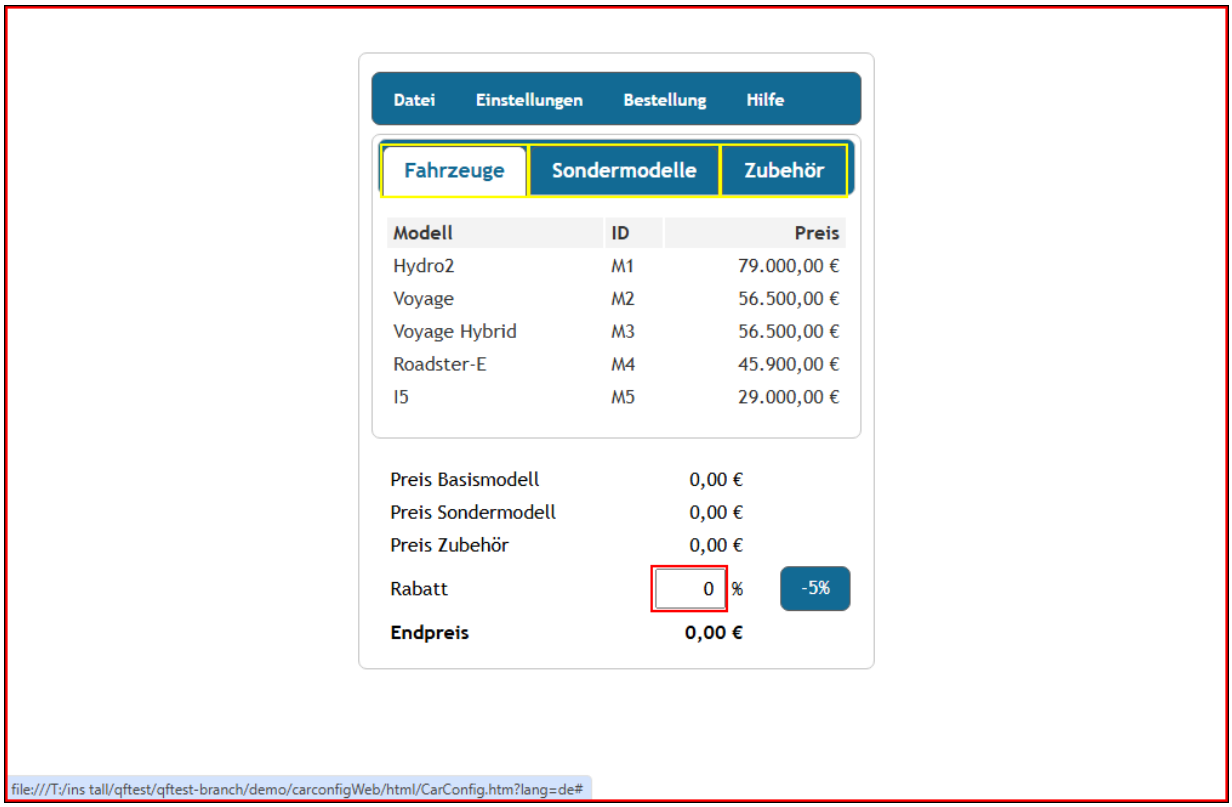


Abbildung 19.3: Bildschirmabbild: Überblick über fehlerhafte Elemente

Fehlerhafte Elemente werden rot, übersprungene Elemente gelb umrandet.

Hinweis Um möglichst akkurate Abbilder und Highlights der Elemente zu erhalten, sollte die Bildschirm- und Browserskalierung auf 100% gesetzt sein.

19.4.2 Hinweise zur Reportgenerierung

Bei der Erstellung des Reports ist es sinnvoll, die zu den Fehlern erzeugten Abbilder der Elemente in den Report aufzunehmen. Hierfür muss im interaktiven Modus ”Miniaturbilder einbetten” ausgewählt werden. Als Skalierung für die Miniaturbilder bietet sich ein fester Wert, wie etwa 300x200 Pixel, an.

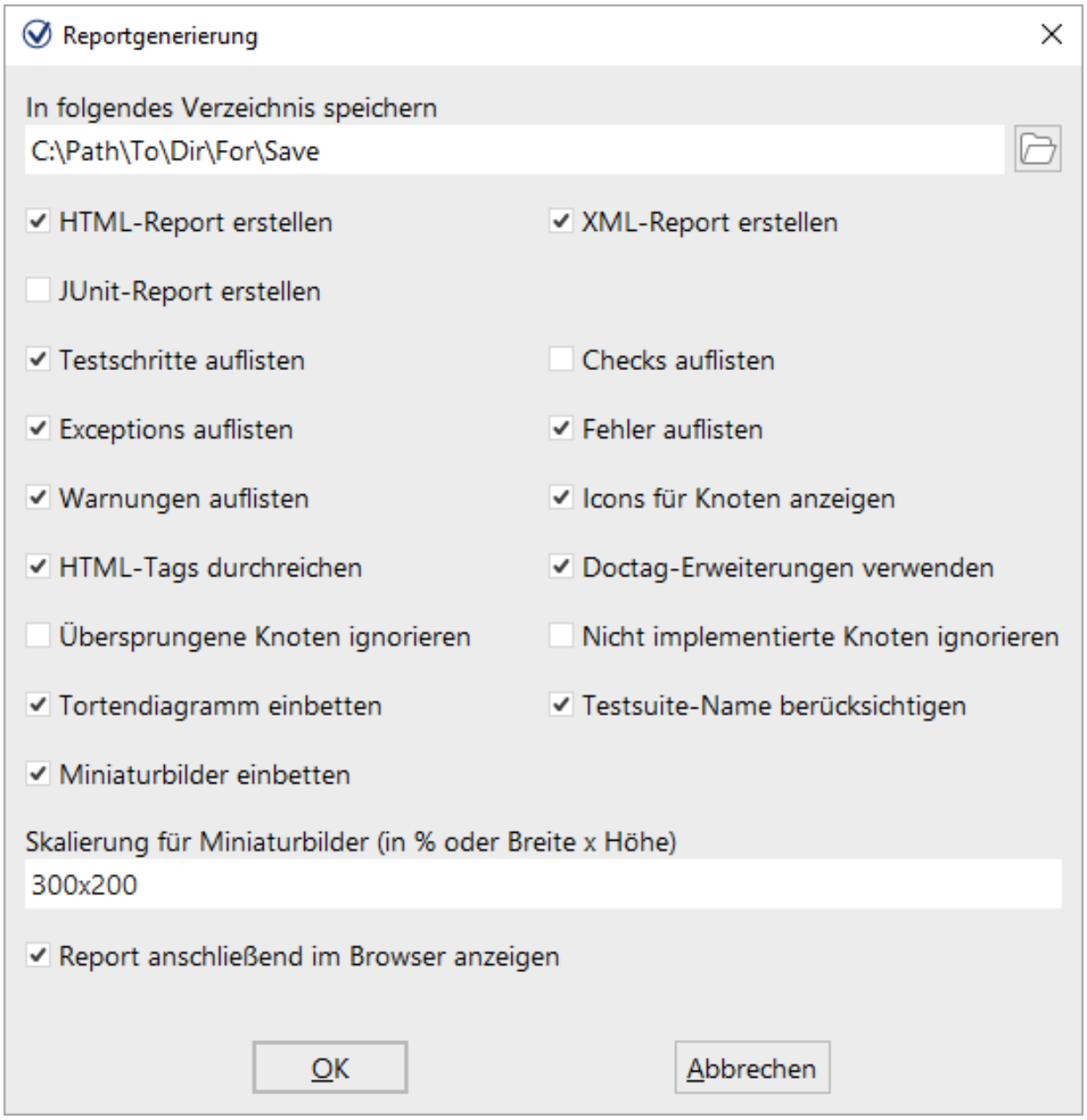


Abbildung 19.4: Beispiel zu Einstellungen bei der Reportgenerierung

Die Kommandozeilenargumente für den batch-Modus lauten `-report-thumbnails`[(988)] und `-report-scale-thumbnails <Prozent>`[(988)].

Kapitel 20

Testen von Java Desktop-Anwendungen im Browser mit Webswing oder JPro

5.2+ Webswing und JPro sind zwei hochinteressante Lösungen, die Swing und JavaFX Desktop-Anwendungen in den Browser bringen. Die zu Grunde liegenden Technologien, Konzepte und Ziele unterscheiden sich deutlich, doch die Herausforderung für QF-Test ist bei beiden die gleiche: Es gilt zwei SUT-Clients gemeinsam in einer koordinierten Weise anzusteuern.

Die Migration von bestehenden Anwendungen ist hier eines der häufigsten Szenarios, so dass die Wiederverwendbarkeit von bestehenden QF-Test Testsuiten für die Java Desktop-Anwendungen äußerst wichtig ist. Dies ist einer der Gründe dafür, warum Testen alleine über den Browser nicht ausreichend ist. Der andere Grund ist, dass QF-Test im Browser nur einen `CANVAS` mit bunten Pixeln sieht (Webswing), bzw. eine Hierarchie von zueinander sehr ähnlichen `DIV` Knoten. Auch wenn letzteres für Tests zumindest halbwegs brauchbar ist und für spezielle Themen wie Lasttests noch interessant werden kann, ist es im Vergleich zum tief reichenden Durchgriff, den QF-Test bei Java-Anwendungen hat, doch sehr eingeschränkt.

Hier betritt ”JiB” die Bühne - QF-Test’s Lösung für ”Java im Browser”.

Hinweis Für JiB werden neben Swing und/oder JavaFX Engine-Lizenzen auch QF-Test Lizenzen für die Web-Engine benötigt.

Video Zum Einstieg gibt es auch ein kurzes

Überblicksvideo zum Webswing-Testing
https://qftest.com/de/yt/webswing-ueberblick.html

auf unserem QF-Test YouTube-Kanal.

Video Im November 2020 fand ein Spezialwebinar zum Thema Webswing-Testen mit QF-Test

statt. Hier geht es zum

Videomitschnitt des Spezialwebinars
https://qftest.com/de/yt/webswing-spezialwebinar.html

auf unserem QF-Test YouTube-Kanal.

QF-Test kommt mit einer Demo-Testsuite für Webswing, die zum besseren Verständnis des folgenden Abschnitts beiträgt. Sie finden diese über den Menüeintrag Hilfe→Beispiel-Testsuiten erkunden..., Eintrag "Webswing SwingSet Suite".

20.1 Technische Konzepte von JiB für Webswing und JPro

Beim JiB Ansatz betrachtet QF-Test die Swing oder JavaFX-Anwendung als das primäre SUT. Fast die gesamte Interaktion wird über die jeweilige Swing oder JavaFX SUT Engine gesteuert. QF-Test öffnet zusätzlich ein Browserfenster und nutzt seine Web-Engine, um dieses Frontend anzusteuern, durch das die Anwendung dargestellt wird und über das der Anwender mit ihr interagiert.

Es gibt zwei Formen der Interaktion von QF-Test mit der Anwendung:

Java-Modus
: QF-Test kann die Event-Steuerung komplett innerhalb der Swing oder JavaFX-Anwendung halten. In diesem Modus dient der Browser nur zum Start der Anwendung, als Referenz für den Anwender und für Sonderfälle, bei denen der Workflow der Anwendung in eine Web-Schnittstelle verlagert wurde, vor allem der Up- und Download von Dateien.

  Dieser Modus ist sehr ähnlich zum normalen Test einer Swing oder JavaFX-Anwendung. Die Wiedergabe von Events passiert identisch dazu. Die Bilder für Abbild-Checks werden von Swing bzw. JavaFX per Off-Screen-Rendering in einen Speicherbereich gezeichnet, ebenfalls identisch zur Desktop-Version.

Web-Modus
: Was der obige Fall nicht abdeckt, ist die Verifikation, dass die Webswing bzw. JPro Integration tatsächlich wie erwartet Ende-zu-Ende funktioniert, also dass der Anwender die Benutzerschnittstelle wirklich wie gewünscht zu sehen bekommt und mit der Anwendung mittels Maus und Tastatur über den Browser interagieren kann. Auch wenn man diskutieren kann, bis zu welchem Grad man zu Grunde liegenden Technologien einfach vertrauen oder diese selbst mit testen sollte, ist die Möglichkeit, echte Ende-zu-Ende Tests über den Browser durchzuführen für dieses Szenario ein sehr wichtiger Aspekt.

QF-Test kann die Wiedergabe der Events über einige Optionen zum Browser umleiten. Tests werden dabei immer noch über die Java-Anwendung ausgeführt, die Komponentenerkennung funktioniert unverändert und QF-Test kümmert sich um die Synchronisation und führt alle vorbereitenden Schritte aus, wie das sichtbar Scrollen von Elementen oder das implizite öffnen von Baumknoten. Im letzten Schritt führt die Swing bzw. JavaFX Engine den Event nicht selbst aus, sondern nutzt eine spezielle Verbindung, um die Event-Information an die QF-Test Web-Engine im Browser weiterzuleiten und dann darauf zu warten, dass der Event dort ausgeführt und über Webswing bzw. JPro zurück an die Java-Anwendung gelangt.

Der letzte Baustein für Ende-zu-Ende Tests ist die Verifikation der Darstellung im Browser, wie sie der Anwender zu sehen bekommt. Anstelle des Off-Screen-Renderings kann QF-Test die Bilderstellung an die Web-Engine delegieren, die ein Bildschirmabbild der entsprechenden Region im Browserfenster aufnimmt. Diese Bilder können in Details von den Java Off-Screen-Varianten abweichen, speziell bei der Textdarstellung und beim Antialiasing. Dies kann mit Hilfe der Algorithmen für Abbild-Checks in QF-Test kompensiert werden. Näheres hierzu finden Sie in Kapitel 59(1309).

Tests im Java-Modus sind sehr robust und effizienter. Wir empfehlen, diesen für die Migration bestehender Tests und für den Großteil der funktionalen Tests zu nutzen. Diese sollten durch verschiedene Tests im Web-Modus ergänzt werden, um Ende-zu-Ende-Zuverlässigkeit zu gewährleisten. Als Daumenregel sollten wiederholte Tests für dieselbe Schnittstelle mit verschiedenen Werten und dem Fokus auf Funktionalität primär im Java-Modus durchgeführt werden. Tests für unterschiedliche Komponenten mit dem Fokus auf Interaktion sollten den Web-Modus nutzen.

Prozeduren zum Umschalten der verschiedenen Optionen werden im Package `qfs.jib` in der Standardbibliothek `qfs.qft` bereitgestellt.

Kapitel 21

Testen von Electron-Anwendungen

4.5+ Electron[1] ist ein Framework zur Ausführung von Cross-Plattform Desktop-Anwendungen mit Hilfe des Webbrowsers Chromium und des Node.js-Frameworks. Diese Anwendungen können mittels HTML, CSS und JavaScript entwickelt werden und auf native Funktionen des Betriebssystems wie Menüs, Dateien oder die Taskleiste zugreifen.

Seit QF-Test Version 4.5 können Anwendungen, die mit dem Electron Framework entwickelt wurden, getestet werden. Sämtliche Features, die QF-Test für das Web bietet, können auch hier verwendet werden.

21.1 Electron Client starten

Die Verbindung zur Electron-Anwendung kann wie bei einer Web-Anwendung über den empfohlenen CDP-Driver-Verbindungsmodus (siehe Abschnitt 51.3.2(1130)) oder den WebDriver-Verbindungsmodus (siehe Abschnitt 51.3.3(1130)) erfolgen.

Mit Hilfe des Schnellstart-Assistenten (vgl. Kapitel 3(32)) lässt sich die passende Vorbereitung(638) Sequenz erstellen. Dies ermöglicht den einfachen Start der Anwendung.

Die Electron spezifischen Angaben im Schnellstart-Assistenten werden weiter unten in diesem Kapitel erläutert. Die verbleibenden optionalen Einstellungen sind im Schnellstart-Assistenten selbst beschrieben.

Video Das Video zeigt die

’ Anbindung der Electron-Anwendung via Schnellstart-Assistenten’.
https://www.qftest.com/de/yt/electron-45.html

[1] https://electronjs.org

21.1.1 Electron Einstellungen im Schnellstart-Assistenten

Wählen Sie in der Rubrik ”Typ der Anwendung” des Schnellstart-Assistenten den Punkt `Eine Electron-Anwendung`.

Geben Sie in der Rubrik ”Electron-Anwendung” Ihre Anwendung inklusive Pfad an, wobei Sie das Dateiauswahlmenü nutzen können, das über den Button rechts des Eingabefelds aktiviert werden kann. Wenn Sie beim Start der Anwendung eigene Kommandozeilenargumente benötigen, so können Sie diese hier ebenfalls angeben.

Electron basiert auf Node.js, welches in der JavaScript-Laufzeitumgebung ”V8” ausgeführt wird. Ab Electron 6 und QF-Test 5.4.0 wird zum Steuern der Anwendung bevorzugt der CDP-Driver-Verbindungsmodus verwendet. Für ältere Anwendungen muss auf den WebDriver-Verbindungsmodus in Kombination mit dem ChromeDriver zurückgegriffen werden. Dabei wird in den meisten Fällen der passende ChromeDriver automatisch ermittelt und heruntergeladen. Dieser wird dann im Unterverzeichnis `chromedriver` des QF-Test Installationsverzeichnisses gespeichert.

21.2 Electron spezifische Funktionalität in QF-Test

Neben den Features, die QF-Test für das Web bietet, steht für Electron die nachfolgend beschriebene Funktionalität zur Verfügung.

21.2.1 Native Menüs

Zur Ansteuerung der nativen Menüs der Electron-Anwendung verwenden Sie bitte einen Auswahl(793) Knoten, bei dem Sie im Attribut QF-Test ID der Komponente die QF-Test ID des Knotens `Webseite` des SUT angeben.

Im Attribut Detail(795) tragen Sie den Menüpunkt mit der folgenden Syntax ein: `clickmenu:@/<Menüpfad>`, wobei statt `<Menüpfad>` das Menü mit dem oder den Untermenüpunkten, getrennt durch `/`, einzutragen ist. Wenn Sie zum Beispiel im Menü `Datei` den Unterpunkt `Speichern unter` aktivieren wollen, lautet der Eintrag `clickmenu:@/Datei/Speichern unter`.

21.2.2 Native Dialoge

5.1.0+ QF-Test unterstützt die Aufnahme, Prüfung und Ansteuerung von Dialogen, welche mit dem `dialog`-Modul von Electron aufgerufen werden. Aus technischen Gründen können sich dabei die angezeigten Dialoge optisch von den üblichen Dialogen unterscheiden.

Bei der Aufnahme wird ein Komponentenknoten mit der Klasse `Dialog` erstellt, auf dem mit Hilfe eines Check Text(806)-Knotens der Text des Dialogfensters geprüft werden kann. Zum Interagieren mit dem Dialogfenster wird ein Auswahl(793) Knoten verwendet, wobei die Details vom Typ des Dialogs abhängen:

- Message Box: Der Wert im Attribut Detail(795) entspricht der Nummer des ausgewählten Buttons, z.B. `2`. Enthält die Message Box zusätzlich eine CheckBox, so kann deren Wert mit `:` getrennt angefügt werden, also z.B. `2:true`.
- Error Box: Bei einer Error Box ist nur ein Knopf vorhanden, der Rückgabewert muss daher `0` lauten.
- Open File Dialog: Im Attribut Detail(795) muss der Dateiname der auszuwählenden Datei angegeben werden. Ist die Auswahl mehrerer Dateien erlaubt, so kann alternativ als Attribut-Wert ein Json-Array mit Dateinamen angegeben werden, also z.B. `["datei.txt","C:\\TEMP\\andere.txt"]`. Möchte man den Dialog abbrechen, gibt man als Attribut-Wert `<CANCEL>` an.
- Save File Dialog: Die Attribut-Werte für Save File Dialog entsprechen denen des Open File Dialogs. Eine Mehrfachauswahl wird von Electron beim Save File Dialog nicht unterstützt.

21.2.3 Erweiterte Javascript-API

5.4.0+ Bei Electron sind separate Render-Prozesse für die Darstellung des Inhalts der Anwendungs-Fenster verantwortlich. Zusätzlich existiert ein Haupt-Prozess, der auf der Node.js-Engine aufsetzt und die Haupt-Anwendungslogik enthält. Für die Ausführung von eigenem Skript-Code in diesem Prozess bietet QF-Test mit den Methoden `mainCallJS` und `mainEvalJS` eine mächtige Erweiterung der `DocumentNode`-API (siehe Abschnitt 54.10.2(1262)) an.

`Object mainCallJS(String code)`
Führt den JavaScript-Code im Hauptprozess der Electron-Anwendung in einer Funktion aus.

Parameter

`code`	Der auszuführende Code.
Rückgabewert	Was immer der Code mit `return` explizit zurückliefert, konvertiert in einen passenden Objekttyp. Allgemeine Javascript-Objekte werden in Json-Objekte konvertiert. Die spezielle Variable `_qf_window` wird dabei durch das `BrowserWindow`-Objekt ersetzt, welches zur aktuellen `DocumentNode` gehört.

`Object mainEvalJS(String script)`

Evaluiert JavaScript Code im Hauptprozess der Electron-Anwendung.

Parameter

`script`	Das auszuführende Skript.
Rückgabewert	Was immer das Skript zurückliefert, konvertiert in einen passenden Objekttyp. Allgemeine Javascript-Objekte werden in Json-Objekte konvertiert. Die spezielle Variable `_qf_window` wird dabei durch das `BrowserWindow`-Objekt ersetzt, welches zur aktuellen `DocumentNode` gehört.

Im Beispiel werden für das aktuell angezeigte Electron-Fenster die ”Chrome Developer Tools” eingeblendet.

```
rc.getComponent("genericDocument").mainCallJS("_qf_window.webContents.openDevTools()")
```

Beispiel 21.1: SUT-Skript zum Anzeigen der Dev Tools in einem Electron-Fenster

21.3 Technische Anmerkungen zum Testen von Electron-Anwendungen im WebDriver-Verbindungsmodus

Damit QF-Test auf die Electron-APIs zugreifen kann, zum Beispiel um Interaktionen mit nativen Menüs aufzunehmen oder abzuspielen, muss QF-Test bei der Nutzung des WebDriver-Verbindungsmodus die Electron-API über den Render-Prozess der getesteten Anwendung aufgerufen werden. Dazu sollte die `nodeIntegration` Property des BrowserWindow nicht auf `false` gesetzt werden. Zusätzlich muss `contextIsolation` deaktiviert bleiben und `enableRemoteModule` den Wert `true` behalten.

```
mainWindow = new BrowserWindow({
    webPreferences: {
        nodeIntegration: true,
        enableRemoteModule: true,
        contextIsolation: false,
        ...
    },
    ...
})
```

Beispiel 21.2: Einfaches Beispiel für eine Electron-Anwendung, die gut getestet werden kann

Wenn Sie vermeiden wollen, dass die vollständige Node-Integration im Render-Prozess zugänglich ist, können Sie die API-Integration mit Hilfe eines `preload`-Skripts verfügbar machen:

```
mainWindow = new BrowserWindow({
    webPreferences: {
        nodeIntegration: false,
        ...
        preload: `${__dirname}/preload.js` // absolute pathname required
    },
    ...
})
```

Beispiel 21.3: Die Einstellungen für den eingeschränkten Node Zugriff

```
// Expose require API in test mode:
if (process.env.NODE_ENV === 'test') {
    window.electronRequire = require;
}
```

Beispiel 21.4: Die zugehörige Datei `preload.js`

Da QF-Test die `NODE_ENV`-Umgebungsvariable immer mit dem Wert `test` belegt, können Sie hierdurch dynamisch die Zugriffssicherheit während des Tests auf die benötigten Werte herabsetzen:

```
const inTestMode = (process.env.NODE_ENV === 'test');
mainWindow = new BrowserWindow({
    webPreferences: {
        nodeIntegration: inTestMode,
        enableRemoteModule: inTestMode,
        contextIsolation: ! inTestMode,
        ...
    },
    ...
})
```

Beispiel 21.5: Dynamisches Beispiel für eine Electron-Anwendung, die gut getestet werden kann

Ab Electron 14 ist das `remote` Modul nicht mehr Teil des Electron-Frameworks, sondern muss explizit in die Anwendung eingebunden werden. Dazu muss beim Erstellen der Anwendung das Modul `@electron/remote` in der `package.json` referenziert und in der `main.js` initialisiert werden:

```
// im "Main"-Prozess:
require('@electron/remote/main').initialize()
```

Beispiel 21.6: Die Initialisierung des `@electron/remote` Moduls

QF-Test verwendet dann für die Zugriffe auf Haupt-Prozess automatisch dieses neue Modul. Weitere Infos finden sich in der Modul-Dokumentation unter https://github.com/electron/remote/.

Bei der Nutzung des CDP-Driver-Verbindungsmodus ist keine Anpassung der Electron-Anwendung für den Test notwendig.

Kapitel 22

Testen von Webdiensten

4.2+ Mit QF-Test Version 4.2 wurde die Möglichkeit eingeführt Webdienste zu testen. Bei HTTP-Requests handelt es sich um eine Technologie, die eher für Entwickler und/oder Programme gedacht sind. Das Aufnehmen einer Browser-HTTP Anfrage und das anschließende Konvertieren in eine Server-HTTP Anfrage ist zwar technisch möglich, macht aber oft nur bedingt Sinn. Daher wird empfohlen, dass Sie die Dokumentation des zu testenden Webdienstes kennen und nutzen.

22.1 REST Webservices

Der Knoten Server-HTTP-Request(910) kann genutzt werden um beliebige HTTP Anfragen zu verschicken.

22.1.1 Der HTTP Standard und Webdienste

Webdienste und Webseiten benutzen das Hypertext Transfer Protocol (kurz HTTP). Dieses Protokoll stellt einen textbasierten Standard dar, der aus Anfragen und Antworten besteht. Dieser sehr nützliche und überraschend kurze Internetstandard kann hier eingesehen werden: Hypertext Transfer Protocol – HTTP/1.1

HTTP Authentication, 2 Basic Authentication Scheme

Eine Liste der aktuell unterstützen Anfrage Methoden

Unterstützte HTTP Methoden
GET
POST
PUT
DELETE
HEAD
OPTIONS
TRACE

Tabelle 22.1: Unterstützte HTTP Methoden

22.1.2 HTTP Anfragen

Lassen Sie uns eine einfache GET-Anfrage eines Browsers analysieren. Wenn Sie eine URL in die Adresszeile des Browsers eingeben wird eine solche HTTP Anfrage automatisch vom Browser erzeugt. Mit Hilfe der Web-Entwicklertools (zum Beispiel in Chrome) kann die abgesendete Anfrage angezeigt werden. Die Anfrage besteht hierbei aus den Teilen `Header`, der `URL` und der `Payload` (body). Die Payload ist optional und kann z.B. bei POST/PUT Anfragen verwendet werden.

Abbildung 22.1: Die vom Browser abgesetzte HTTP GET-Anfrage

Die Antwort des Servers besteht aus einem `Response code`, einem `Header` und dem optionalen `Payload`.

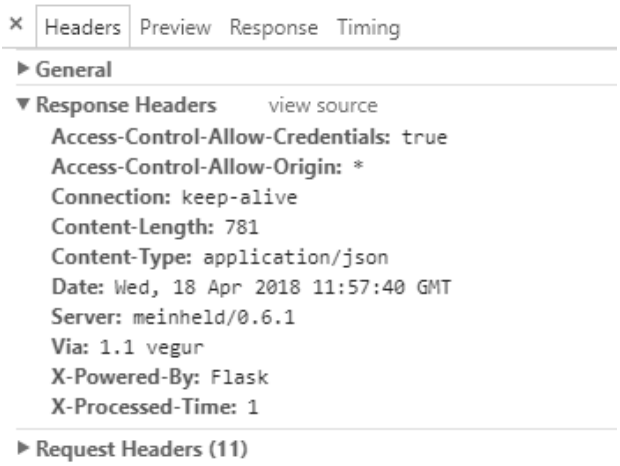


Abbildung 22.2: Die GET-Antwort des Webservers

22.1.3 Beispiele

Bei der Verwendung des HTTP Server Request Knotens muss darauf geachtet werden, dass alle benötigten Daten in den dafür vorgesehenen Attributen spezifiziert werden, also `URL`, `Header` und `Payload`. Wenn Sie Teile der Antwort des Webservers benötigen, sollten Sie einen entsprechenden Variablenname in das jeweilige Attribut unter ”Variablen für die Antwort des Servers” eintragen. Beispiele für Verwendung des HTTP Server Request Knoten befinden sich unter `demo/webservices` in der Beispieltestsuite `webservice_testing.qft`.

Die Beispiele wurden mit Hilfe eines HTTP-Proxies erstellt. Beispiele für solche Proxies sind Charles (https://www.charlesproxy.com/) oder seine freie Alternative James (https://github.com/james-proxy/james).

Kapitel 23

Datengetriebenes Testen

Datengetriebenes Testen ist ein sehr wichtiger Aspekt der Testautomatisierung. Das Ziel besteht, kurz gesagt, darin, einen Testfall mehrfach mit dem selben Ablauf, aber unterschiedlichen Eingabe- und Vergleichswerten durchzuführen. QF-Test bietet verschiedene Möglichkeiten, um Daten für datengetriebene Tests abzulegen oder aus externen Quellen zu laden. Die bequemste Variante basiert auf einem Datentreiber Knoten, der die Umgebung zur Iteration über die Datensätze bereitstellt, sowie ein oder mehrere Daten Knoten, welche die Variablen Werte für die Testdurchläufe liefern. Dabei gibt es in QF-Test keinen Daten Knoten als solches. Er dient als Oberbegriff für spezielle Ausprägungen wie eine Datentabelle oder eine CSV-Datei. Das Ganze lässt sich am besten anhand von Beispielen erläutern. Eine Demo-Testsuite mit einfachen und komplexeren Beispielen finden Sie unter dem Namen `datadriver.qft` im Verzeichnis `doc/tutorial` unterhalb des Wurzelverzeichnisses von QF-Test. Bitte beachten Sie, dass Sie veränderte Testsuiten am besten in einem projektspezifischen Ordner speichern.

23.1 Beispiele für Datentreiber

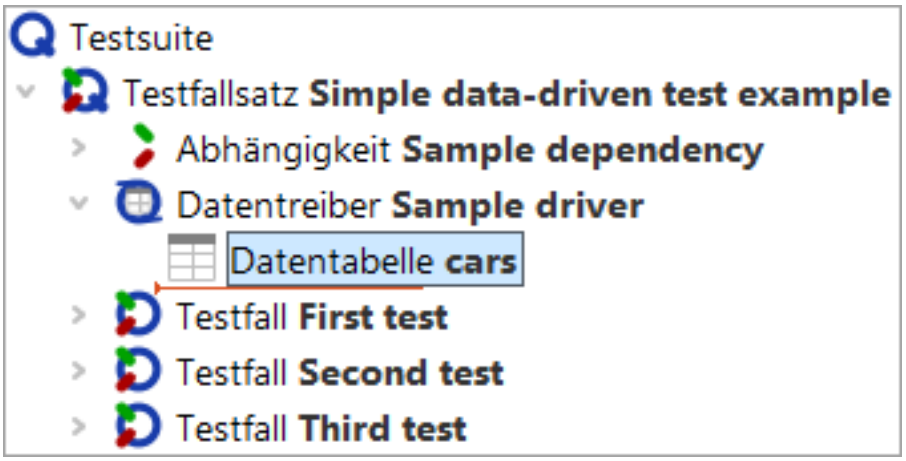


Abbildung 23.1: Ein einfacher datengetriebener Test

Obige Abbildung zeigt einen Testfallsatz mit einem Datentreiber Knoten, der einen einzelnen Daten Knoten in Form einer Datentabelle enthält. Der Inhalt des Datentabelle Knotens ist wie folgt:

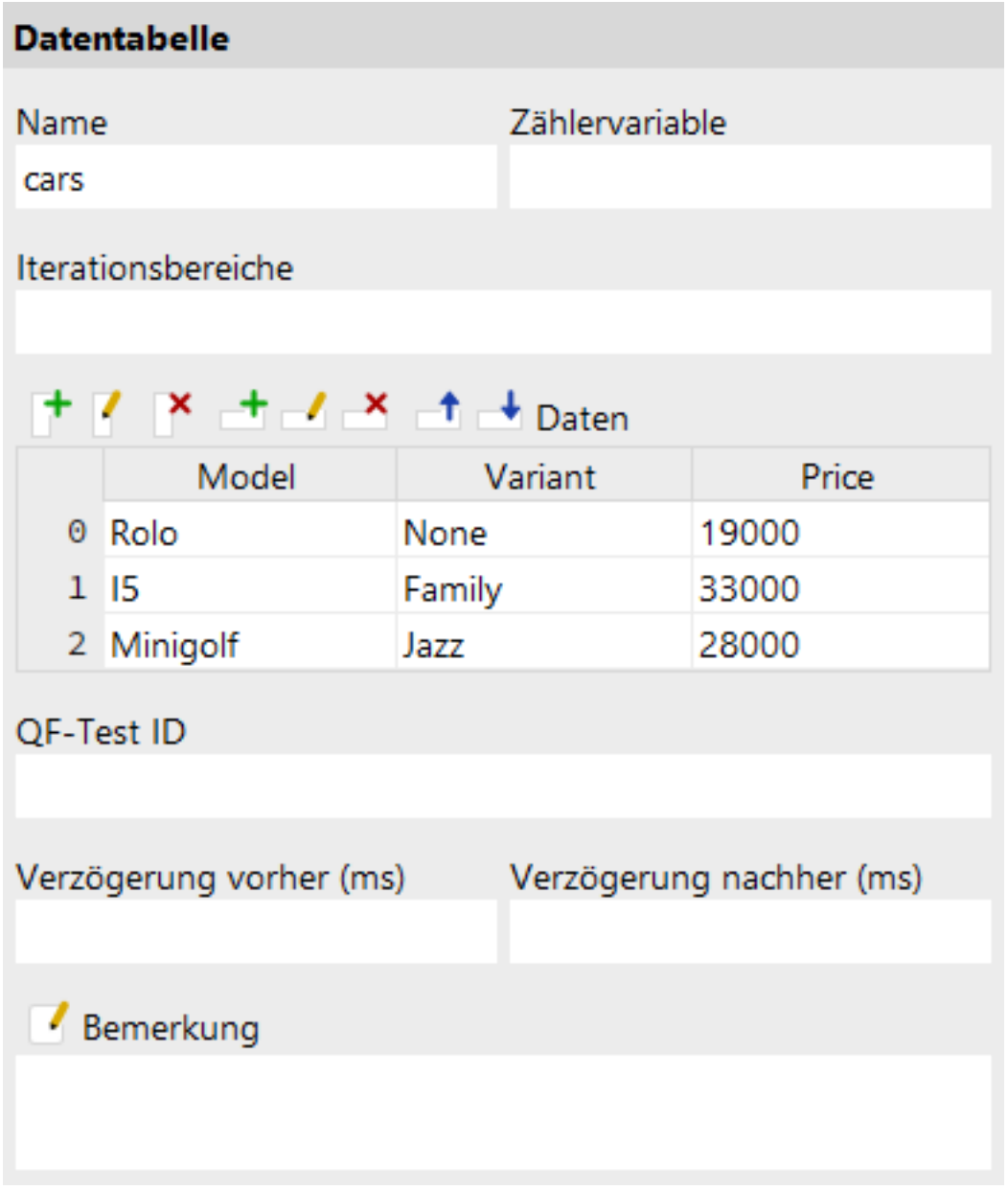


Abbildung 23.2: Beispiel einer Datentabelle

Wird der Testfallsatz ausgeführt, iteriert er über die Zeilen der Datentabelle. Für jeden der drei Iterationsschritte werden die Werte der entsprechenden Zeile der Tabelle an die Variablen mit dem Namen der jeweiligen Spalte gebunden. Im Beispiel ergibt das für den ersten Durchlauf die Bindungen ”Model=Rolo”, ”Variant=None” und ”Price=19000”. Im zweiten Durchlauf ist dann ”Model=I5”, im dritten ”Model=Minigolf”. Bei jedem Durchgang werden alle Testfall Childknoten des Testfallsatz Knotens ausgeführt.

Die folgende Abbildung zeigt ein Protokoll für obigen Testfallsatz:

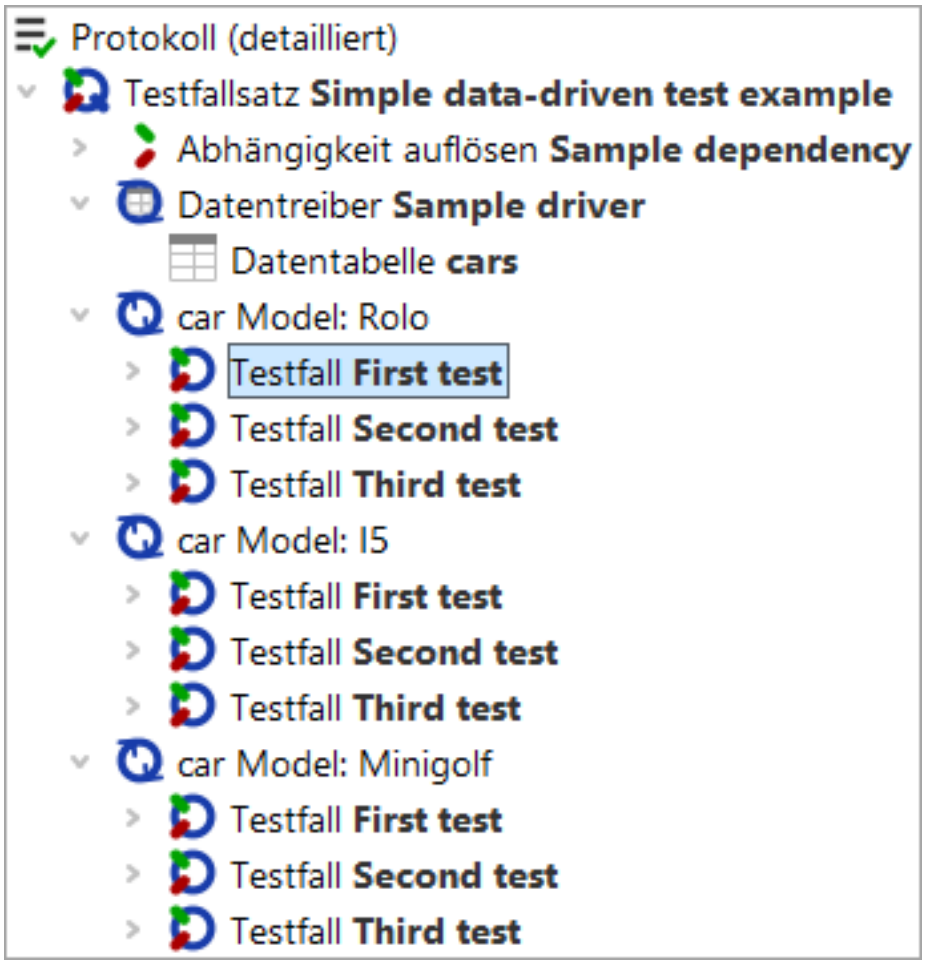


Abbildung 23.3: Protokoll eines datengetriebenen Tests

Im nächsten Beispiel sehen wir, dass datengetriebene Tests nicht auf eine einfache Schleife beschränkt sind:

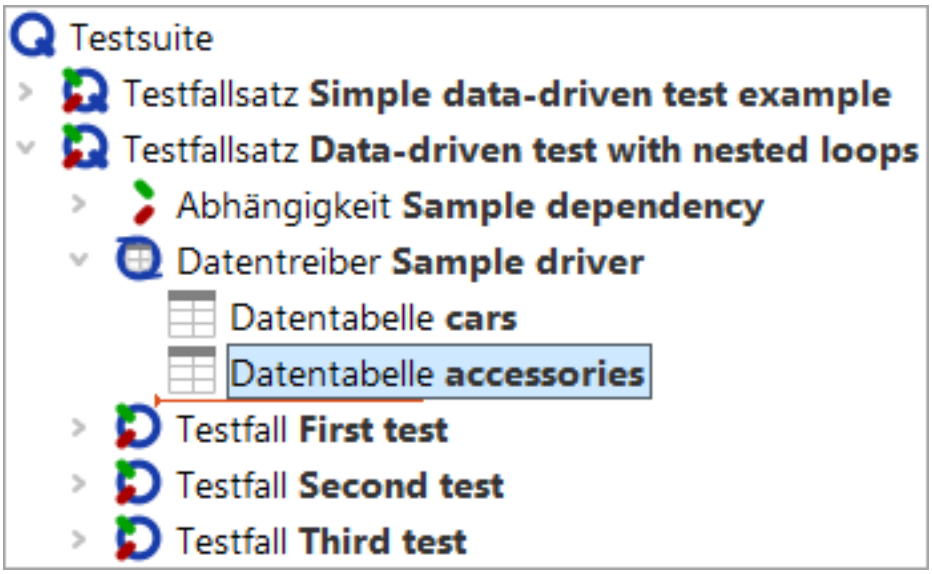


Abbildung 23.4: Datengetriebene Tests mit verschachtelten Schleifen

Der Datentreiber enthält nun eine zweite Datentabelle mit folgendem Inhalt:

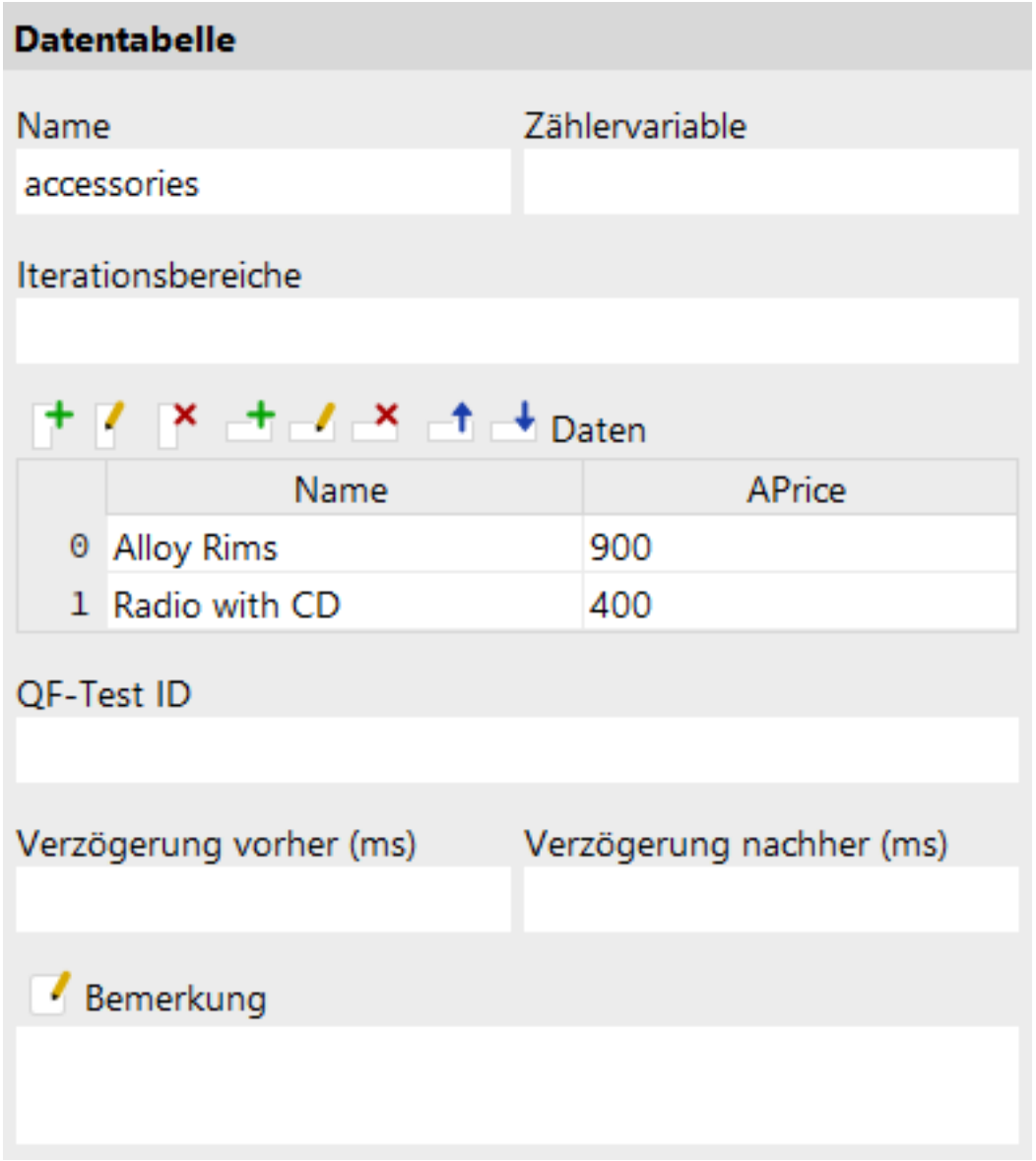


Abbildung 23.5: Zweites Beispiel einer Datentabelle

Der Testfallsatz wird nun insgesamt sechs Iterationen durchlaufen, da für jede der drei Iterationen der äußeren Schleife namens ”cars” beide Iterationen der inneren Schleife namens ”accessories” durchlaufen werden. Im folgenden Protokoll ist dies gut zu erkennen:

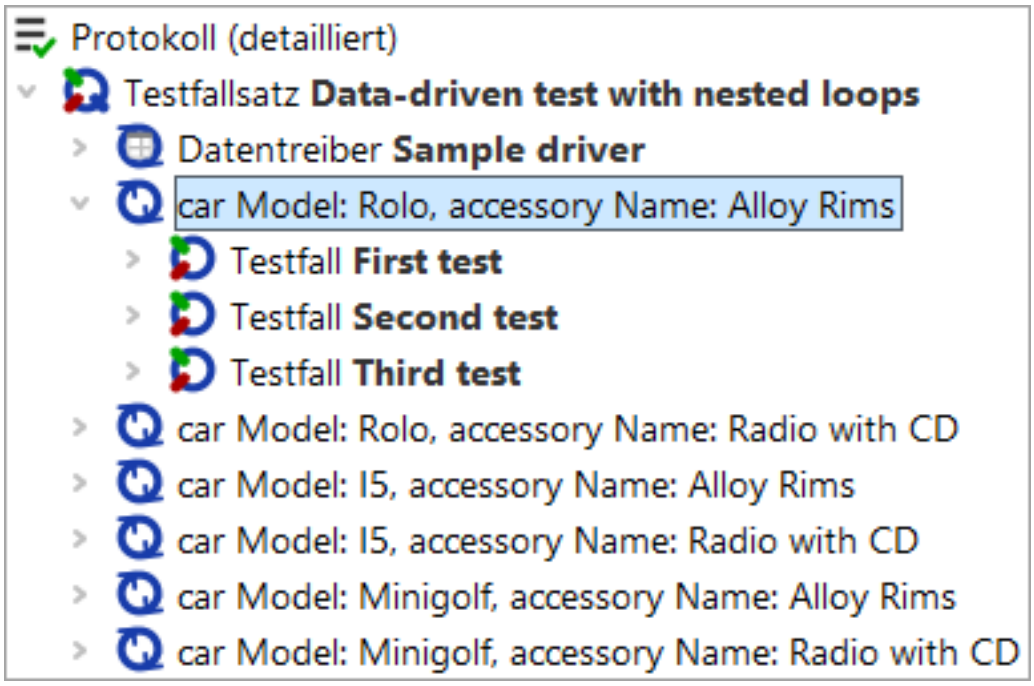


Abbildung 23.6: Protokoll eines datengetriebenen Tests mit verschachtelten Schleifen

Hinweis

Die äußerst hilfreichen dynamisch generierten Namen der Schleifendurchgänge erhalten Sie, indem Sie das Attribut Name für Schleifendurchgang im Protokoll des Datentreiber Knotens auf den Wert ”car Model: $(Model)” im ersten bzw. ”car Model: $(Model), accessory Name: $(Accessory)” im zweiten Beispiel setzen. Wie Sie sehen, wird dieser Wert für jede Iteration individuell expandiert, so dass Sie auf die Variablen zugreifen können, die für die Iteration gebunden werden.

23.2 Anwendung von Datentreibern

Wie im vorangehenden Beispiel erklärt, muss ein Datentreiber Knoten in einen Testfallsatz Knoten eingefügt werden, und zwar zwischen die optionalen Abhängigkeit und Vorbereitung Knoten. Wird der Testfallsatz ausgeführt, führt er zunächst einen eventuell vorhandenen Datentreiber Knoten aus. Der Inhalt des Datentreiber Knotens ist nicht auf Datentabelle und CSV-Datei beschränkt. Wie ein normaler Sequenz Knoten kann der Datentreiber jede Art von ausführbaren Knoten enthalten, um damit eventuelle Vorbereitungen durchzuführen, die zur Bereitstellung der Daten notwendig sind. Hierdurch ist es auch möglich, die Daten Knoten mehrfach zu nutzen, indem diese einfach in eine Prozedur gestellt werden, welche aus den Datentreiber Knoten heraus aufgerufen wird.

Im Prinzip entspricht ein Daten Knoten einer Schleife, bei der in jedem Durchgang ver-

schiedene Werte an Variablen gebunden werden. Ein Daten Knoten muss mit einem Namen im Datentreiber Kontext eines Testfallsatzes registriert werden. Dadurch kann die Schleife durch einen Break(691) Knoten mit dem gleichen Namen abgebrochen werden. Nachdem der Datentreiber ausgeführt wurde, iteriert der Testfallsatz über die dabei registrierten Daten Schleifen.

Bei verschachtelten Schleifen wird der zuerst registrierte Daten Knoten als äußerste Schleife ausgeführt. Seine Variablen werden zuerst gebunden und haben damit geringere Bindungskraft als die Variablen der inneren Schleife(n).

23.3 Beispiele für Datentreiber

In der mitgelieferten Testsuite `doc/tutorial/datadriver.qft` finden Sie Beispiele für die Verwendung von CSV- und Excel-Dateien.

23.4 Fortgeschrittene Anwendung

Neben dem Datentabelle(650) Knoten gibt es verschiedene weitere Möglichkeiten, Daten in einem Datentreiber zu binden. Die Knoten Excel-Datei(659), CSV-Datei(664), Datenbank(653) und Datenschleife(668) werden alle ausführlich in Abschnitt 42.4(646) erläutert.

Außerdem können Daten durch Aufruf der Prozeduren `qfs.databinder.bindList` oder `qfs.databinder.bindSets` in der Standardbibliothek `qfs.qft` gebunden werden. Als Parameter werden Listen oder Sätze von Werten in Form von Texten übergeben, die aufgeteilt werden, um über die Werte zu iterieren. Informationen zur Standardbibliothek finden Sie in Kapitel 8 des Tutorials.

Und schließlich können Daten aus Jython (und analog aus Groovy bzw. JavaScript) auch direkt mit Hilfe des `databinder` Moduls gebunden werden, welches folgende Methoden bietet:

void bindDict(Object rc, String loopname, Map dict, String counter=None, String intervals=None)
Erzeugt und registriert ein databinder Objekt, das Daten aus einem Dictionary bindet. Die Schlüssel definieren die Namen der Variablen und die Werte sind Sequenzen, deren Inhalt an die jeweilige Variable gebunden wird.

Parameter

rc	Der aktuelle Runcontext.
loopname	Der Name unter dem die Daten gebunden werden, entsprechend dem Attribut Name eines Daten Knotens.
dict	Das zu bindende Dictionary.
counter	Ein optionaler Variablenname für den Iterationszähler.
intervals	Optionale Bereiche von Indizes, getrennt durch Komma, z.B. ”0,2-3”.

void bindList(Object rc, String loopname, String varname, Object values, String separator=None, String counter=None, String intervals=None)
Erzeugt und registriert ein databinder Objekt, das eine Liste von Werten an eine Variable bindet.

Parameter

rc	Der aktuelle Runcontext.
loopname	Der Name unter dem die Daten gebunden werden, entsprechend dem Attribut Name eines Daten Knotens.
varname	Der Name der zu bindenden Variable.
values	Die zu bindenden Werte. Entweder eine Sequenz oder ein Text, der aufgeteilt wird.
separator	Optionales Trennzeichen für die Aufteilung der Werte, falls sie als Text übergeben werden. Standard ist Leerraum.
counter	Ein optionaler Variablenname für den Iterationszähler.
intervals	Optionale Bereiche von Indizes, getrennt durch Komma, z.B. ”0,2-3”.

`void bindSets(Object rc, String loopname, Object varnames, Object values, String separator=None, String counter=None, String intervals=None)`
Erzeugt und registriert ein databinder Objekt, das Sätze von Werten an einen Satz von Variablen bindet.

Parameter

`rc`	Der aktuelle Runcontext.
`loopname`	Der Name unter dem die Daten gebunden werden, entsprechend dem Attribut Name eines Daten Knotens.
`varnames`	Die Namen der zu bindenden Variablen. Entweder eine Sequenz, oder ein Text, der aufgeteilt wird.
`values`	Die Sätze von zu bindenden Werten. Entweder eine Sequenz von Sequenzen, wobei jede innere Sequenz einem Satz von Daten entspricht, oder ein Text der aufgeteilt wird.
`separator`	Optionales Trennzeichen für die Aufteilung der Variablen und der Sätze von Werten, falls diese als Text übergeben werden. Standard ist Leerraum. Sätze von Werten werden jeweils durch Zeilenumbrüche getrennt.
`counter`	Ein optionaler Variablenname für den Iterationszähler.
`intervals`	Optionale Bereiche von Indizes, getrennt durch Komma, z.B. ”0,2-3”.

Einige Beispiele:

```
import databinder
# Three iterations with the values "spam", "bacon" and "eggs"
# bound to the variable named "ingredient"
databinder.bindList(rc, "meal", "ingredient", ["spam", "bacon", "eggs"])
# Same with string values
databinder.bindList(rc, "meal", "ingredient", "spam bacon eggs")
# Same with string values and special separator
databinder.bindList(rc, "meal", "ingredient", "spam|bacon|eggs", "|")
# Two iterations, the first with item="apple" and number="5",
# the second with item="orange" and number="3"
databinder.bindSets(rc, "fruit", ["item", "number"],
                    [["apple",5], ["orange",3]])
# Same with string values, note the linebreak
databinder.bindSets(rc, "fruit", "item number", """apple 5
orange 3""")
# Same as before with the data stored in a dict
databinder.bindDict(rc, "fruit",
                    {"item": ["apple", "orange"],
                     "number": [5,3]})
```

Beispiel 23.1: Beispiele für die Verwendung des `databinder` Moduls

Kapitel 24

Reports und Testdokumentation

Neben Testsuiten und Protokollen erstellt QF-Test noch eine Reihe weiterer Dokumente. Das wichtigste davon ist der Report, der eine Gesamtübersicht über die Ergebnisse eines oder mehrerer Testläufe liefert, sowie eine Aufstellung der daran beteiligten Testsuiten und deren Einzelergebnissen. Der Report ist leicht zu lesen und ohne tiefere Kenntnis von QF-Test zu verstehen. Er ergänzt damit das Protokoll, welches vor allem der Fehleranalyse dient und ein gutes Verständnis von QF-Test erfordert.

Die folgende Abbildung zeigt ein Beispiel einer Zusammenfassung eines Reports:

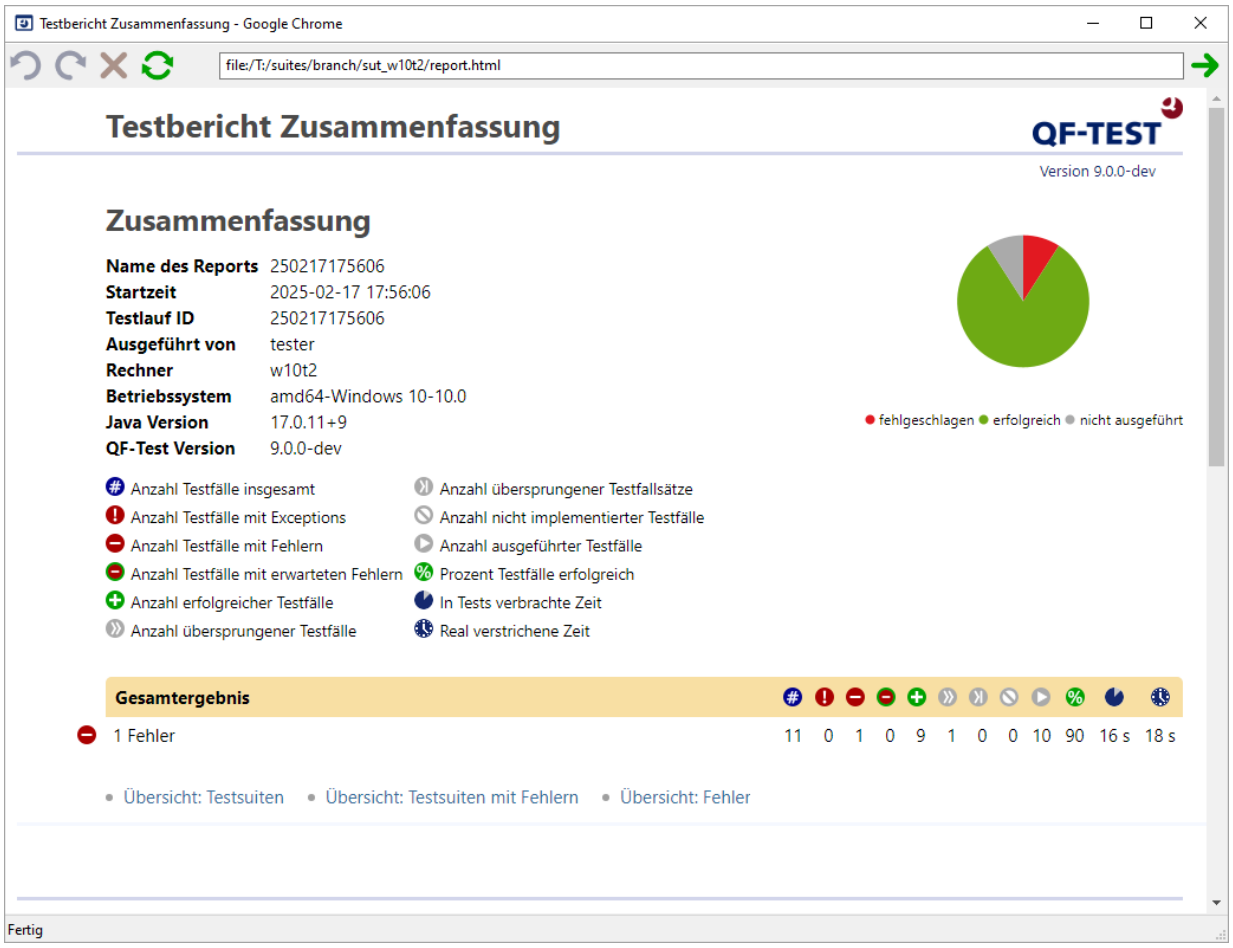


Abbildung 24.1: Beispiel Report

Die anderen Dokumente sind eher statischer Natur. Sie beschreiben keine Testergebnisse, sondern den Inhalt von Testsuiten. Ein *testdoc* Dokument gibt einen Überblick über die Struktur der Testfallsatz(606) und Testfall(599) Knoten einer oder mehrerer Testsuiten. Es dient Test-Architekten und QS Projektleitern zur Dokumentation des aktuellen Standes der Testentwicklung. Ein *pkgdoc* Dokument ist ähnlich aufgebaut, beschreibt stattdessen aber die Struktur der Package(680), Prozedur(672) und Abhängigkeit(630) Knoten, vergleichbar zu JavaDoc, welches zur Dokumentation von Java-Bibliotheken genutzt wird. Zielgruppe hierfür sind die Mitarbeiter, welche die Tests implementieren und hierzu Informationen zu bereits vorhandenen Prozeduren und deren Parameter benötigen.

24.1 Reports

Bevor wir näher auf das Erstellen von Reports eingehen können, müssen zunächst einige grundlegende Konzepte erläutert und Begriffe definiert werden.

24.1.1 Reportkonzepte

Ein Report ist das Ergebnis einer oder mehrerer *Testläufe*. Ein Testlauf umfasst die Ausführung einer oder mehrerer Testsuiten, normalerweise gemeinsam in einem Batch-Lauf. Ein Testlauf wird durch seine *Run-ID* identifiziert. Es ist möglich einen Testlauf in mehreren Schritten durchzuführen indem den Protokollen die selbe Run-ID zugewiesen wird.

Ein Report kann einen *Reportnamen* haben. Für einen Report, der einen einzelnen Testlauf repräsentiert, stimmt dieser normalerweise mit der Run-ID des Testlaufs überein. Für Reports, die mehrere Testläufe zusammenfassen, kann ein eigener Name festgelegt werden.

Reports können in verschiedenen Varianten erstellt werden: XML, HTML und JUnit. Die meisten Anwender werden wohl die HTML-Variante nutzen, welche in einem Browser dargestellt, ausgedruckt und archiviert werden kann. Die XML-Variante kann als Basis zur weiteren Verarbeitung der Testergebnisse dienen, z.B. um diese in einer Datenbank zu sammeln oder um speziell angepasste HTML-Reports zu generieren. Wir empfehlen immer HTML und XML-Reports gemeinsam zu erstellen, sofern Sie keinen guten Grund für ein anderes Vorgehen haben. JUnit-Reports basieren auf dem JUnit-XML-Format, wie es Apache Ant mit Hilfe der JUnitReport-Task erzeugt. Dieses Format ist nicht so schön und detailliert wie bei den beiden anderen von QF-Test angebotenen Varianten, jedoch wird es direkt von vielen Continuous Integration Tools verstanden und kann hilfreich für eine schnelle Integration mit solchen sein.

Ein Report besteht aus einem Dokument mit der Zusammenfassung sowie je einem Dokument pro Protokoll. Diese werden zusammen mit ergänzenden Dateien wie Icons, Stylesheets und Bildschirmabbildern in einem gemeinsamen Verzeichnis abgelegt. Auf Dateiebene repräsentiert dieses Verzeichnis den Report.

Die Anordnung der Dateien innerhalb des Report Verzeichnisses kann wie weiter unten beschrieben über die Kommandozeile beeinflusst werden. Es gibt im Wesentlichen zwei Varianten die Dateien anzuordnen: Analog zur Dateistruktur der ursprünglichen Testsuiten oder entsprechend der Struktur der Protokolldateien.

24.1.2 Inhalt von Reports

Vor dem Gesamtergebnis enthält ein Testbericht, wie oben gezeigt, eine Zusammenfassung von wichtigen Systeminformationen und eine Legende mit der Bedeutung der verwendeten Zählersymbole (siehe auch Abspielen von Tests[41]).

Hinweis

Der Unterschied zwischen ”In Tests verbrachte Zeit” und ”Real verstrichene Zeit” sind explizite Verzögerungen in Knoten mittels ’Verzögerung vorher/nachher’ oder Unterbrechungen durch den Benutzer.

Die Inhalte von Reports basieren auf den ausgeführten Testsuiten. Die Grobstruktur entspricht deren Testfallsatz[606] und Testfall[599] Knoten. Die Bemerkung[613] Attribute des Wurzelknotens, sowie der Testfallsatz und Testfall Knoten unterstützen die selben Doctags wie die in Abschnitt 24.2[336] beschriebenen testdoc Dokumente. Zusätzlich kann über das Doctag ’@title’ in der Bemerkung des Wurzelknotens ein Titel für das Report-Dokument festgelegt werden, das für die jeweilige Testsuite erstellt wird.

Falls `-report-teststeps`[988] im Batchmodus oder die entsprechende Option im interaktiven Dialog verwendet wird (Standard ist true), können Testfälle mit Hilfe von Testschritt[621] Knoten weiter heruntergebrochen werden. Neben dem expliziten Verpacken in Testschritt Knoten können beliebige Knoten auch durch Angabe des Doctags ’@teststep’, optional gefolgt von einem Namen, in der Bemerkung des jeweiligen Knotens als Testschritt ausgezeichnet werden. Für Testschritt Knoten werden außerdem die Doctags ’@author’, ’@version’ und ’@since’ unterstützt. Namen, Kommentare und Werte von Doctags können Variablen enthalten, die zum Zeitpunkt der Ausführung expandiert werden, so dass der expandierte Wert im Report angezeigt wird. Dies ist speziell für Testschritte in Prozeduren sehr hilfreich.

Ist das Aufführen von Testschritten im Report aktiv, werden auch Vorbereitung, Aufräumen und Abhängigkeit Knoten aufgeführt und Checks, Bildschirmabbilder und Meldungen, darunter Warnungen, Fehler und Exceptions, passend in die verschachtelten Schritte einsortiert. Wenn die Testsuiten ordentlich implementiert sind kann der resultierende Report als einfach lesbare Zusammenfassung dessen dienen, was während der Ausführung eines Tests vor sich ging.

Ob Warnungen und Checks im Report aufgeführt werden, hängt von den Kommandozeilenargumenten `-report-warnings`[988] und `-report-checks`[986] bzw. deren interaktiven Pendants ab. Warnungen zur Wiedererkennung von Komponenten werden nie aufgeführt, da sie zu technisch sind und leicht den Report überfluten könnten. Für Checks muss zwischen solchen unterschieden werden, die eine echte Verifikation repräsentieren und solchen, die nur zur Steuerung des Tests dienen. Ein klassisches Beispiel für letztere ist die Überprüfung, ob eine Checkbox bereits selektiert ist, bevor auf diese geklickt wird. Normalerweise führt QF-Test nur solche Check Knoten im Report auf, deren Ergebnisbehandlung dem Standard entspricht, d.h. Fehlerstufe der Meldung ist ’Fehler’, es wird keine Exception geworfen und keine Ergebnisvariable gebunden. Alle anderen werden als Helfer für die Teststeuerung interpretiert und erscheinen nicht im

Report. Für Fälle, in denen dieses Vorgehen nicht passt, kann ein Check Knoten explizit durch Angabe des Doctags '@report' in den Report aufgenommen, oder durch Angabe von '@noreport' davon ausgeschlossen werden. Natürlich werden fehlgeschlagene Checks als Warnung, Fehler oder Exception interpretiert (abhängig von ihrem Attribut Fehlerstufe der Meldung) und können nicht vom Report ausgeschlossen werden, wenn Meldungen mit der entsprechenden Fehlerstufe dort angezeigt werden.

Zusätzliche Meldungen, Checks und Bildschirmabbilder können mit Hilfe von Skripten in den Report eingefügt werden. Hierzu dienen die Methoden `rc.logMessage`, `rc.check` mit Varianten und `rc.logImage`, die einen optionalen `report` Parameter haben. Details finden Sie in der API Dokumentation zum Runcontext in Abschnitt 50.5(1030).

24.1.3 Reports erstellen

Es gibt drei verschiedene Wege Reports zu erstellen:

- Interaktiv aus einem Protokoll über den Menüeintrag Datei→Report erstellen....
- Im Batchmodus als Ergebnis eines Testlaufs.
- Im Batchmodus durch Transformation zuvor erstellter Protokolle.

Die interaktive Variante ist einfach und bedarf keiner weiteren besonderen Erklärungen. Wählen Sie das Zielverzeichnis für den Report und entscheiden Sie sich für die XML und/oder HTML-Variante.

Für die Reporterstellung im Batchmodus gibt es einige Kommandozeilen Optionen, welche in Abschnitt 44.2(976) erklärt werden. Sehen wir uns nun zunächst die Variante zur Erstellung eines Reports als Ergebnis eines Testlaufs an:

Die Kommandozeilen Syntax für einen normalen Testlauf im Batchmodus ist `qftest -batch <Testsuite> [<Testsuite>...]`

Um einen kombinierten XML- und HTML-Report zu erstellen, verwenden Sie `-report <Verzeichnis>`(986). Für das Erzeugen der XML- oder HTML-Varianten alleine oder um diese zu trennen, dienen die Argumente `-report-xml <Verzeichnis>`(988) und/oder `-report-html <Verzeichnis>`(987). Für JUnit-Reports funktioniert `-report-junit <Verzeichnis>`(987) entsprechend.

Die Run-ID eines Testlaufs wird mittels `-runid [<ID>]`(989) angegeben, der Reportname mittels `-report-name <Name>`(987). Ist `-report-name <Name>` nicht angegeben entspricht der Reportname der Run-ID.

Um die Dateien im Report Verzeichnis entsprechend den Testsuiten anzuordnen, verwenden Sie `-sourcedir <Verzeichnis>`(990). Eine Struktur auf Basis der Protokolle können Sie mittels `-runlogdir <Verzeichnis>`(989) erreichen.

Ein typisches Beispiel einer Kommandozeile für einen Batch-Lauf, die Gebrauch von den in Abschnitt 44.2.4(995) beschriebenen Platzhaltern macht, sieht etwa wie folgt aus:

```
qftest -batch -runid +M+d -runlog logs/+i -report report_+i
       -sourcedir . suite1.qft subdir/suite2.qft
```

Beispiel 24.1: Erstellen eines Reports als Ergebnis eines Testlaufs

Das Erstellen eines Reports als separaten Schritt aus bereits vorhandenen Protokollen ist in vieler Hinsicht ähnlich. Anstelle der Testsuiten müssen die zu transformierenden Protokolle angegeben werden. Die Optionen `-runid [<ID>]` und `-sourcedir <Verzeichnis>` haben keinen Effekt. Das folgende Beispiel zeigt, wie Sie einen Report zur Zusammenfassung der Testergebnisse einer Woche erstellen können. Es wird davon ausgegangen, dass die Protokolle unterhalb des `logdir` Verzeichnisses liegen, gegebenenfalls in Unterverzeichnissen:

```
qftest -batch -genreport -report report_+M+d
       -report.name week_of_+y+M+d logdir
```

Beispiel 24.2: Erstellen eines Reports für eine Wochen-Zusammenfassung

24.1.4 Individualisierung von Reports

Die XML und HTML-Reports werden über eine XSL-Transformation des QF-Test Protokolls erzeugt. Durch die Anpassung der XSLT Stylesheets können Inhalt und Struktur der erzeugten Dokument angepasst werden.

Weitere Informationen hierzu finden sie in unserem Blog-Artikel ”Wie erstellt man kundenspezifische HTML-/XML-/JUnit-Reports?”: https://www.qftest.com/blog/article/2019/02/28/wie-erstellt-man-kundenspezifische-htmlxmljunit-reports.html

Alternativ kann die Darstellung des HTML-Reports durch gängige Web-Techniken via JavaScript beeinflusst werden. Eine Datei namens `user.js` wird in das Report-Verzeichnis kopiert und in alle Seiten des HTML-Reports eingebunden. Um die Darstellung anzupassen überschreiben Sie diese Datei mir Ihrer eigenen Version nach Generierung des Reports. Ein Beispiel hierzu finden Sie im Kommentar der `user.js` Datei.

24.2 Testdoc-Dokumentation für Testfallsätze und Testfälle

Die Dokumente vom Typ *testdoc* liefern eine Übersicht über und Detailinformationen zu den Testfallsatz[606] und Testfall[599] Knoten einer oder mehrerer Testsuiten. Wenn Testschritte[621] in Testfälle enthalten sind, werden diese Schritte auch aufgeführt. Testaufruf[614] Knoten werden normalerweise bei der Testdoc-Generierung ignoriert. Mit `-testdoc-followcalls`[993]=true werden die referenzierten Ziele, also Testfall, Testfallsatz oder ganze Testsuiten, so eingebunden, als wären diese Teil der Ausgangssuite.

Diese Dokumentation ist ein wichtiges Werkzeug für QS Projektleiter um den aktuellen Stand der Testentwicklung im Blick zu behalten. Wie die Reports sind testdoc Dokumente als Verzeichnis aufgebaut, mit einer Zusammenfassung und je einem detaillierten Dokument pro Testsuite.

Ein testdoc Dokument für eine einzelne Testsuite kann interaktiv über den Eintrag Testdoc-Dokumentation erstellen... im Datei Menü generiert werden. Das ist hilfreich um während der Testentwicklung schnell überprüfen zu können, ob alle Tests ordentlich dokumentiert sind.

Für den Gebrauch als Referenz sind dagegen komplette Sätze von Dokumenten über mehrere Testsuiten aus einem gesamten Projekt vorzuziehen. Diese können durch Aufruf von QF-Test im Batchmodus mit Hilfe des Kommandozeilenarguments `-gendoc`[981] erstellt werden. In seiner einfachsten Form sieht ein Aufruf zum Erstellen einer testdoc Dokumentation für eine gesamte Verzeichnishierarchie so aus:

```
qftest -batch -gendoc -testdoc test_documentation
        directory/with/test suites
```

Beispiel 24.3: Erstellen von testdoc Dokumentation

Detaillierte Informationen zu den verfügbaren Kommandozeilenargumenten finden Sie in Kapitel 44[971].

Für optimale Ergebnisse können Sie in den Bemerkung[613] Attributen von Testfallsatz und Testfälle Knoten HTML-Auszeichnungen verwenden und Doctags einsetzen. Ein *Doctag* ist ein mit '@' beginnendes Schlüsselwort, manchmal gefolgt von einem Namen und immer von einer Beschreibung. Dies ist ein bewährtes Konzept von JavaDoc, dem Format zur Dokumentation von Java-Programmen (siehe http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html#tag).

Hinweis Die Doctags müssen nach der eigentlichen Beschreibung stehen, da Beschreibung, welche nach den Doctags steht, ignoriert wird. Weiterhin dürfen keine Doctags innerhalb

der Beschreibung vorkommen.

Die folgenden Doctags werden für Testfallsatz und Testfall Knoten unterstützt:

@deprecated
Gibt an, dass ein Testfallsatz oder Testfälle nicht länger verwendet werden soll. Die Beschreibung sollte erklären, seit wann und warum das der Fall ist und vor allem angeben, was stattdessen verwendet werden soll.

@condition
Informelle Beschreibung der Bedingung unter welcher der Knoten ausgeführt oder übersprungen wird.

@param
Ein Prozedurparameter. Es folgen der Name des Parameters und seine Beschreibung.

@charvar
Eine charakteristische Variable. Es folgen der Name der Variable und ihre Beschreibung.

@author
Autor des Testfallsatzes oder Testfalls.

@version
Version des Testfallsatzes oder Testfalls.

@since
Seit welcher Version dieser Testfallsatz oder Testfall verfügbar ist.

Neben den oben beschriebenen Doctags kann über das Doctag '@title' ein Titel für das testdoc Dokument festgelegt werden, das für die jeweilige Testsuite erstellt wird.

24.3 Pkgdoc-Dokumentation für Packages, Prozeduren und Abhängigkeiten

Die Konzepte von *pkgdoc* Dokumenten und die Methoden für deren Erstellung sind praktisch identisch zur testdoc-Dokumentation. Im Unterschied zu dieser beschreibt pkgdoc die Package(680), Prozedur(672) und Abhängigkeit(630) Knoten einer Testsuite. Es dient Test-Entwicklern als Übersicht über die Prozeduren, die zur Implementierung von Tests zur Verfügung stehen.

Die Standardbibliothek `qfs.qft` ist ein schönes Beispiel, wie ein pkgdoc Dokument aussehen kann.

Ein pkgdoc Dokument kann ebenfalls interaktiv über Datei→HMTL/XML Pkgdoc erstellen... oder im Batchmodus generiert werden. Auch hier finden Sie nähere Informationen zu den verfügbaren Kommandozeilenargumenten in Kapitel 44(971).

Wie das folgende Beispiel zeigt, können testdoc und pkgdoc auch gemeinsam in einem einzigen Batch-Lauf erstellt werden:

```
qftest -batch -gendoc -testdoc tests -pkgdoc procedures
       directory/with/test suites
```

Beispiel 24.4: Erstellen von testdoc und pkgdoc Dokumentation in einem Durchgang

Natürlich unterstützt pkgdoc ebenso HTML-Auszeichnungen und Doctags. Die folgenden Doctags stehen für Packages, Prozeduren und Abhängigkeiten zur Verfügung:

@deprecated
Gibt an, dass ein Package, eine Prozedur oder eine Abhängigkeit nicht länger verwendet werden soll. Die Beschreibung sollte erklären, seit wann und warum das der Fall ist und vor allem angeben, was stattdessen verwendet werden soll.

@param (nur Prozedur und Abhängigkeit)
Ein Parameter für eine Prozedur oder Abhängigkeit. Es folgen der Name des Parameters und seine Beschreibung.

@charvar (nur Abhängigkeit)
Eine charakteristische Variable. Es folgen der Name der Variable und ihre Beschreibung.

@return (nur Prozedur)
Der Rückgabewert der Prozedur.

@result (nur Prozedur und Abhängigkeit)
Hier könnten Seiteneffekte wie das Setzen von globalen Variablen dokumentiert werden.

@throws (nur Prozedur)
Erwartete Exception. Es folgen der Name der Exception und die Beschreibung, in welchem Fall diese geworfen wird.

@catches (nur Abhängigkeit)
Eine Exception, die von der Abhängigkeit gefangen wird. Es folgen der Name der Exception und die Beschreibung Ihrer Behandlung.

@author
Autor des Packages der Prozedur oder der Abhängigkeit.

@version
Version des Packages der Prozedur oder der Abhängigkeit.

@since
Seit welcher Version dieses Package diese Prozedur oder diese Abhängigkeit verfügbar ist.

Neben den oben beschriebenen Doctags kann über das Doctag '@title' ein Titel für das pkgdoc Dokument festgelegt werden, das für die jeweilige Testsuite erstellt wird.

Kapitel 25

Testausführung

Die Ausführung von Tests ist unter verschiedenen Blickwinkeln zu betrachten: Zum einen werden Tests während der Entwicklung aus der laufenden QF-Test Anwendung heraus ausgeführt, um die Funktionstüchtigkeit der Tests zu überprüfen. Diese Situation wurde bereits in Abschnitt 4.2(41) behandelt; sie reduziert sich im Wesentlichen auf den Aufruf von Wiedergabe→Start im QF-Test Hauptmenü.

Auf der anderen Seite sollen die entwickelten Tests regelmäßig ausgeführt werden, um die Stabilität der getesteten Software sicherzustellen - etwa bei nächtlichen Regressionstests. Anstatt die auszuführende Testsuite mit QF-Test zu öffnen und über die Benutzeroberfläche auszuführen, ist es hier viel praktischer, den Testlauf von der Kommandozeile im sogenannten Batchmodus zu starten. Diese Art der Testausführung bildet den ersten Abschnitt dieses Kapitels (Testausführung im Batchmodus(340)).

In manchen Situationen, etwa wenn der Testlauf auf einem entfernten Rechner ausgeführt werden soll, kommt eine weitere Variante ins Spiel: der Daemonmodus. Hier dient eine laufende QF-Test Instanz sozusagen als Ausführungsorgan für Tests. Mit diesem Thema wird sich der zweite Abschnitt befassen (Testausführung im Daemonmodus(346)).

Hinweise zur Integration von QF-Test mit Build Tools wie `ant`, `maven` oder `Jenkins` finden Sie in Kapitel 29(397).

25.1 Testausführung im Batchmodus

Grundlage für die Ausführung von Tests über die Kommandozeile bildet eine Vielzahl von Argumenten, mit denen QF-Test im Batchmodus gestartet werden kann. Anschließend werden exemplarisch einige davon behandelt; eine Übersicht über alle Optionen findet man in Kapitel 44(971).

Die nachfolgenden Beispiele sind für Windows geschrieben, lassen sich aber leicht auf

Linux übertragen. Neben den Pfadangaben unterscheidet sich die Syntax nur noch in Bezug auf die Platzhalter (Abschnitt 44.2.4(995)): Unter Linux kann neben der Form +X auch %X verwendet werden. Unter Windows gibt es außerdem neben der GUI-Anwendung `qftest.exe` auch eine Konsolen-Variante `qftestc.exe`. Diese wartet, bis die Ausführung von QF-Test beendet ist und gibt u. a. auch print-Ausgaben von einem Server-Skript(717) aus.

25.1.1 Verwenden der Kommandozeile

Der einfachste Aufruf von QF-Test, um einen Test auszuführen, sieht so aus:

```
qftest -batch -run c:\mysuites\suiteA.qft
```

Beispiel 25.1: Testausführung von der Kommandozeile

Das Argument `-batch` sorgt dafür, dass QF-Test ohne graphische Benutzeroberfläche gestartet wird. Das zweite Argument, `-run`, sagt QF-Test, dass ein Test ausgeführt werden soll. Den Abschluss der Kommandozeile bildet die auszuführende Testsuite.

Hinweis

Das Argument `-run` ist optional, d. h. die Testausführung ist als Standard für den Batchmodus definiert.

Führt man die obige Anweisung aus, werden alle Testfall und Testfallsatz Knoten nacheinander ausgeführt, die in der Testsuite `suiteA.qft` auf der obersten Ebene (d. h. direkt unterhalb des Testsuite Knotens) definiert sind. Nach Beendigung des Testlaufs findet man im aktuellen Verzeichnis, unter dem gleichen Namen wie die Testsuite, eine Protokolldatei, der man das Testergebnis entnehmen kann.

Durch Angabe von `-nolog` kann die Erstellung der Protokolldatei auch unterdrückt werden. Das macht aber wahrscheinlich nur dann Sinn, wenn man seinen Test um eigene Protokollausgaben bereichert hat, die etwas in eine Datei geschrieben werden. Ansonsten lieferte nur noch der Rückgabewert von QF-Test einen Hinweis auf den Ausgang des Testlaufs: 0 bedeutet, dass der Test erfolgreich absolviert wurde, ein positiver Wert hingegen ist die Folge von Warnungen, Fehlern oder gar Exceptions (siehe Abschnitt 44.3(996)).

In der Regel möchte man also wohl ein Protokoll des Testlaufs erstellen und dieses an einer bestimmten Stelle im Dateisystem ablegen. Das geht über den Parameter `-runlog`:

```
qftest -batch -compact -runlog c:\mylogs\+b c:\mysuites\suiteA.qft
```

Beispiel 25.2: Testausführung mit Protokolldatei

Nun wird eine Protokolldatei `suiteA.qrz` im angegebenen Verzeichnis `c:\mylogs` erstellt. Der Platzhalter `+b` sorgt dafür, dass sie den Namen von der Testsuite übernimmt. Durch die Angabe von `-compact` wird erreicht, dass nur die wichtigsten Knoten in die Protokolldatei übernommen werden, also nur solche, die für den Report und gegebenenfalls die Fehleranalyse benötigt werden. Bei sehr umfangreichen Tests kann dieser Parameter helfen, den Anspruch an Arbeitsspeicher in Grenzen zu halten, wobei die neuere Möglichkeit, Protokolldateien zu splitten, sogar noch mächtiger ist. Nähere Informationen dazu finden Sie in Abschnitt 7.1[138].

Hinweis

Ob die Datei hier tatsächlich als komprimiertes Protokoll (das ist vom obigen "compact" zu unterscheiden) mit der Endung `.qrz` erstellt wird, hängt von den Einstellungen ab. Um ein bestimmtes Format zu erzwingen, kann die Endung explizit angegeben werden, also z. B. `-runlog c:\mylogs\+b.qrl`, wenn unkomprimiertes XML gewünscht wird.

Manchmal möchte man vielleicht nicht die komplette Testsuite, sondern nur bestimmte Testfälle ausführen. Über den Parameter `-test` lässt sich ein ganz bestimmter Knoten zur Ausführung festlegen:

```
qftest -batch -runlog c:\mylogs\+b -test "Mein Test" c:\mysuites\suiteA.qft
```

Beispiel 25.3: Ausführung eines bestimmten Knotens

Die Angabe `"Mein Test"` bezieht sich dabei auf das Attribut QF-Test ID des auszuführenden Knotens; bei Testfall oder Testfallsatz Knoten kann auch deren qualifizierter Name angegeben werden. Sollen mehrere Knoten ausgeführt werden, kann das Argument `-test <ID>` auch mehrfach angegeben werden. Neben der QF-Test ID eines Knotens versteht `-test` auch die Angabe eines numerischen Index. So würde etwa `-test 0` den ersten Knoten unterhalb von Testsuite ausführen.

Das Protokoll liefert eine eher technische Sicht auf den Ablauf eines Tests; es ist vor allem nützlich, um während eines Testlaufs aufgetretene Fehler zu analysieren (vgl. Abschnitt 7.1[138]). Eine übersichtliche Darstellung über die ausgeführten Testfälle und eventuell aufgetretene Fehler bietet hingegen der Report. Er wird aus der Protokolldatei erstellt und liegt anschließend im XML- und/oder HTML-Format vor. Der Report kann also auch nachträglich aus einer (oder mehreren) Protokolldateien erstellt werden (vgl. Kapitel 24[330]). Um gleich bei der Testausführung dafür zu sorgen, verwendet man `-report:`

```
qftest -batch -runlog c:\mylogs\+b
       -report c:\mylogs\rep_+b_+y+M+d+h+m
       c:\mysuites\suiteA.qft
```

Beispiel 25.4: Testausführung mit Reportgenerierung

Hier werden die XML- und HTML-Reportdateien in einem Verzeichnis erstellt, dessen Name sowohl die Testsuite wie auch Datum und Uhrzeit enthält, etwa: `c:\mylogs\rep_suiteA_0806042152`. Ersetzt man das Argument `-report` durch `-report.xml` bzw. `-report.html`, so wird nur der XML- bzw. nur der HTML-Report erstellt.

Testfälle sind oft parametrisiert, das heißt, sie verwenden Variablen, deren Werte den Testablauf bestimmen. Ist beispielsweise im Testsuite[595] Knoten eine Variable `myvar` definiert, kann deren Vorgabewert zur Testausführung überschrieben werden:

```
qftest -batch -variable myvar="Value from command line"
       -runlog c:\mylogs\+b c:\mysuites\suiteA.qft
```

Beispiel 25.5: Testausführung mit Variablen

Sollen mehrere Variablen gesetzt werden, kann `-variable <name>=<wert>` auch mehrfach angegeben werden.

25.1.2 Windows Befehlsskript

Die Möglichkeit, Tests über die Kommandozeile ausführen zu können, bildet die Grundlage zur einfachen Integration von QF-Test in bestehende Testmanagement-Systeme (siehe Anbindung an Testmanagementtools[373]). Wer hingegen kein Testmanagement-System betreibt, mag es vielleicht praktisch finden, das Kommando zur Testausführung in ein Skript einzubetten. Ein einfaches Windows Befehlsskript (`qfbatch.bat`) könnte zum Beispiel wie folgt aussehen:

```
@echo off
setlocal
if "%1" == "" (
    echo Usage: qfbatch Testsuite
    goto end
) else (
    set suite=%~f1
)
set logdir=c:\mylogs
pushd c:\programs\qftest\qftest-9.0.0\bin
@echo on
.\qftest -batch -compact -runlog %logdir%\+b %suite%
@echo off
if %errorlevel% equ 0 (
  echo Test terminated successfully
  goto end
)
if %errorlevel% equ 1 (
  echo Test terminated with warnings
  goto end
)
if %errorlevel% equ 2 (
  echo Test terminated with errors
  goto end
)
if %errorlevel% equ 3 (
  echo Test terminated with exceptions
  goto end
)
if %errorlevel% leq -1 (
  echo Error %errorlevel%
  goto end
)
:end
popd
```

Beispiel 25.6: Befehlsskript `qfbatch.bat` zur Ausführung einer Testsuite

Nun braucht man nur noch das Skript auszuführen, mit dem Dateinamen der auszuführenden Testsuite als Parameter. Der Rest geht von alleine: Die Testsuite wird ausgeführt, die Protokolldatei im Verzeichnis `logdir` abgelegt und abschließend gibt das Skript in Abhängigkeit vom QF-Test Rückgabewert eine Statusmeldung aus.

25.1.3 Groovy

3.0+

Seit Version 3 ist die Sprache Groovy Bestandteil von QF-Test (siehe Kapitel 11(186)). Hauptsächlich gedacht ist sie zum Schreiben von Server- und SUT-Skripten, aber Groo-

vy kann ebenso wie Jython auch außerhalb von QF-Test verwendet werden. Diese Sprache ist sicher gut geeignet, um sich selbst ein kleines Management-System zur Testautomatisierung zu erstellen. Mit Groovy lässt sich übrigens auch der Einsatz von Ant vereinfachen, denn statt mit klobigen XML-Dateien, über die sich zudem etwa Bedingungen nur schwer umsetzen lassen, kann man dank `AntBuilder` mit übersichtlichem Groovy-Code arbeiten. Das folgende Beispiel kommt allerdings ohne Ant aus:

```
def suite = ''
if (args.size() == 0) {
    println 'Usage: groovy QfExec Testsuite'
    return
}
else {
    suite = args[0]
}
def qftestdir = 'c:\\programs\\qfs\\qftest\\qftest-9.0.0'
def qftest = qftestdir + '\\bin\\qftest.exe'
def command = [qftest,
               "-batch",
               "-compact",
               "-runlog", "c:\\mylogs\\+b",
               suite]
def printStream = { stream ->
    while (true) {
        try {
            stream.eachLine { println it }
        } catch (IOException) {
            break
        }
    }
}
println "Running command: $command"
def proc = command.execute()
new Thread().start() { printStream(proc.in) }
new Thread().start() { printStream(proc.err) }
proc.waitFor()
switch (proc.exitValue()) {
    case '0': println 'Test terminated successfully'; break
    case '1': println 'Test terminated with warnings'; break
    case '2': println 'Test terminated with errors'; break
    case '3': println 'Test terminated with exceptions'; break
    default: println "Error ${proc.exitValue()}"
}
```

Beispiel 25.7: Groovy-Skript `QfExec.groovy` zur Ausführung einer Testsuite

Sollte Groovy unabhängig von QF-Test auf Ihrem Rechner installiert sein, kann die Beispiel-Testsuite einfach mit `groovy QfExec c:\mysuites\suiteA.qft` ausgeführt werden. Andernfalls kann auch die Groovy jar-Datei aus der QF-Test Installation

verwendet werden. Am besten nutzt man in diesem Fall zur Ausführung von Groovy wieder ein Befehlsskript:

```
@echo off
setlocal
if "%1" == "" (
    echo Usage: qfexec Testsuite
    goto end
)
set qftestdir=c:\programs\qftest\qftest-9.0.0
set scriptfile=QfExec.groovy
java -cp %qftestdir%/lib/groovy-all.jar groovy.ui.GroovyMain %scriptfile% %*
:end
```

Beispiel 25.8: Befehlsskript `qfexec.bat` zur Ausführung von Groovy (hier `QfExec.groovy`)

Die Testsuite kann nun mit `qfexec c:\mysuites\suiteA.qft` ausgeführt werden.

25.2 Testausführung im Daemonmodus

Im Daemon-Modus gestartet lauscht QF-Test auf RMI Verbindungen und stellt darüber ein Interface für die verteilte Ausführung von Tests zur Verfügung. Dies kann für die Testdurchführung in einem verteilten Lasttest Szenario (vgl. Kapitel 33(437)) ebenso hilfreich sein wie für die Integration mit vorhandenen Testmanagement- oder Testdurchführungs-Werkzeugen (vgl. Kapitel 28(373)).

Hinweis

GUI Tests benötigen eine aktive Benutzersession um korrekt ausgeführt werden zu können. Sie finden im Kapitel Aufsetzen von Testsystemen(474) nützliche Tipps und Tricks für die Einrichtung Ihrer Testsysteme. Der technische Hintergrund ist in FAQ 14 beschrieben.

25.2.1 Starten des Daemons

!!! Warnung !!!

Jeder, der Zugriff auf den QF-Test Daemon hat, kann auf dessen Rechner Programme mit den Rechten des Benutzerkontos starten, unter dem der Daemon läuft. Daher sollte Zugriff nur berechtigten Nutzern gewährt werden.

Wenn Sie den Daemon nicht in einer sicheren Umgebung betreiben, in der jeder Nutzer als berechtigt gilt, oder wenn Sie eine eigene Bibliothek zum Zugriff auf den Daemon

entwickeln, sollten Sie unbedingt **Abschnitt 55.3**(1294) **lesen**. Darin ist beschrieben, wie Sie die Kommunikation mit dem Daemon mittels SSL absichern können.

Um mit dem Daemon arbeiten zu können, muss er zunächst auf irgendeinem Rechner im Netzwerk gestartet werden (das kann natürlich auch `localhost` sein):

```
qftest -batch -daemon -daemonport 12345
```

Beispiel 25.9: Starten des QF-Test Daemon

Hinweis Wichtiger Hinweise zur Kompatibilität:

3.5+ Beginnend mit QF-Test Version 3.5 wird für die Kommunikation mit dem Daemon standardmäßig SSL verwendet. Um mit einer QF-Test Version älter als 3.5 interagieren zu können, muss der Daemon mit leerem Kommandozeilenargument `-keystore <Keystore-Datei>`(983) in folgender Form gestartet werden:

```
qftest -batch -keystore= -daemon -daemonport 12345
```

Beispiel 25.10: Starten des QF-Test Daemon ohne SSL

Lässt man das Argument `-daemonport` weg, lauscht der Daemon auf Port 3543. Ob der Daemon erfolgreich gestartet wurde, kann man z. B. mit dem Programm `netstat` prüfen:

Windows
```
netstat -a -p tcp -n | findstr "12345"
```

Linux
```
netstat -a --tcp --numeric-ports | grep 12345
```

Will man den Daemon auf einem entfernten Rechner starten, bieten sich zum Beispiel ssh oder VNC an. Ob und wie das ggf. in Ihrem Netzwerk funktioniert, weiß der Netzwerkadministrator. Um die folgenden Beispiele nachzuvollziehen, reicht aber auch ein lokaler Daemon.

25.2.2 Steuern des Daemons über die QF-Test Kommandozeile

3.0+

Die einfachste Möglichkeit, den lauschenden Daemon anzusprechen bietet die Kommandozeile, indem man QF-Test im sogenannten *calldaemon* Modus startet. Das folgende Beispiel prüft, ob der Daemon an der angegebenen Host-/Portkombination erreichbar ist:

```
qftestc -batch -calldaemon -daemonhost localhost -daemonport 12345 -ping
```

Beispiel 25.11: Pingen eines QF-Test Daemon

Anders als das obige `netstat`-Kommando funktioniert `-ping` auch über Rechnergrenzen hinweg (auf dem lokalen Rechner kann man das Argument `-daemonhost` einfach weglassen).

Auf ähnliche Weise wie man eine Testsuite im Batchmodus ausführt, kann man nun einen Daemon dazu bringen, einen bestimmten Testfall auszuführen und ein Protokoll des Testlaufs zu schreiben:

```
qftest -batch -calldaemon -daemonhost somehost -daemonport 12345
       -runlog c:\mylogs\+b
       -suitedir c:\mysuites
       suiteA.qft#"Mein Testfall"
```

Beispiel 25.12: Ausführung eines Testfalls mit dem QF-Test Daemon

Hinweis

Anders als im Batchmodus wird beim Verwendung eines Daemons ein Testfall oder ein Testfallsatz (andere Knotentypen sind nicht erlaubt) stets über seinen qualifizierten Namen angesprochen, z.B. "Mein Testfallsatz.Mein Testfall" (zur Erinnerung: Beim Batchmodus wird `-test <ID>` verwendet). Will man die komplette Testsuite `suiteA.qft` ausführen, so lässt man die Angabe des Testfalls einfach weg oder schreibt `suiteA.qft#.`.

Wird der Daemon auf einem entfernen Rechner gestartet, gibt man diesen bei der Ausführung von `calldaemon` über den Parameter `-daemonhost` explizit an (Vorgabe ist `-daemonhost localhost`). Man beachte, dass sich dabei der Parameter `-suitedir` auf den entfernten Rechner bezieht (auf dem der Daemon läuft), während `-runlog` eine lokale Datei bezeichnet.

3.4+

Gerade dann, wenn man die Testausführung nicht so leicht beobachten kann, bietet es sich an, zusätzlich das Argument `-verbose` anzugeben, um so Statusinformationen auf der Konsole angezeigt zu bekommen (auf Windows muss dazu `qftestc` verwendet werden).

Ein Daemon, lokal oder entfernt, lässt sich über das *calldaemon* Kommando `-terminate` wieder beenden:

```
qftest -batch -calldaemon -daemonport 12345 -daemonhost localhost -terminate
```

Beispiel 25.13: Beenden eines QF-Test Daemon

Eine vollständige Übersicht über die *calldaemon*-Parameter finden Sie im Kapitel Kommandozeilenargumente und Rückgabewerte(971).

25.2.3 Steuern des Daemons über die Daemon API

Das Ansprechen des Daemons über die QF-Test Kommandozeile ist auf der einen Seite ganz praktisch, auf der anderen jedoch bietet sie nur eingeschränkte Möglichkeiten. Um die daemonischen Fähigkeiten voll auszureizen, muss man sich der Daemon-API bedienen. Wir werden diese hier beispielhaft vorstellen, die vollständige Schnittstelle ist in Kapitel 55(1276) beschrieben.

Für erste Experimente mit der Daemon-API bietet sich ein Server-Skript Knoten an:

```
from de.qfs.apps.qftest.daemon import DaemonRunContext
from de.qfs.apps.qftest.daemon import DaemonLocator
host = "localhost"
port = 12345
# Leading r means raw string to allow normal backslashes in the path string.
testcase = r"c:\mysuites\suiteA.qft#Mein Testfall"
timeout = 60 * 1000
def calldaemon(host, port, testcase, timeout=0):
    daemon = DaemonLocator.instance().locateDaemon(host, port)
    trd = daemon.createTestRunDaemon()
    context = trd.createContext()
    context.runTest(testcase)
    if not context.waitForRunState(DaemonRunContext.STATE_FINISHED, timeout):
        # Run did not finish, terminate it
        context.stopRun()
        if not context.waitForRunState(DaemonRunContext.STATE_FINISHED, 5000):
            # Context is deadlocked
            raise UserException("No reply from daemon RunContext.")
        rc.logError("Daemon call did not terminate and had to be stopped.")
    result = context.getResult()
    log = context.getRunLog()
    rc.addDaemonLog(log)
    context.release()
    return result
result = calldaemon(host, port, testcase, timeout)
rc.logMessage("Result from daemon: %d" %result)
```

Beispiel 25.14: Daemon-API im Server-Skript

Das Skript zeigt den grundlegenden Mechanismus der Daemon-Ansteuerung:

- Zunächst muss mit `locateDaemon` ein laufender Daemon gefunden werden.
- Über den Aufruf von `createTestRunDaemon` wird eine Umgebung für Testläufe bereitgestellt.
- Zur eigentlichen Testausführung benötigt man ein Context-Objekt (`createContext`). Hierzu wird eine (Runtime-)Lizenz benötigt.

- Über den Context lässt sich der Testlauf starten (`runTest`) und dessen Zustand abfragen. `waitForRunState` wartet während der (in Millisekunden) angegebenen Zeitspanne, bis ein bestimmter Zustand eingetreten ist; hier wird eine Minute lang darauf gewartet, dass der Testlauf abschließt.
- Schließlich, nach Ende des Testlaufs, liefert der Context über die Methode `getResult` einen Rückgabewert, der Auskunft über das Ergebnis des Testlaufs gibt (vgl. Rückgabewerte von QF-Test(996)).
- Darüber hinaus kann man über den Context auch das Protokoll des Testlaufs abholen und mittels der `rc`-Methode `addDaemonLog` in das lokale Protokoll einfügen.

Hinweis Das Beispielskript verzichtet aus Gründen der Übersichtlichkeit auf jegliche Fehlerbehandlung. Gerade beim Arbeiten mit einem Daemon sollte man aber jeden Aufruf auf eventuelle Fehler überprüfen.

Hinweis Ein Nachteil ist mit der Daemon-Steuerung aus einem Server-Skript verbunden: Es wird eine zusätzliche QF-Test Lizenz benötigt, um den Skript-Knoten interaktiv oder im Batchmodus auszuführen. Das gilt allerdings nicht, wenn man den oben beschriebenen *calldaemon* Modus verwendet oder sich außerhalb von QF-Test mit dem Daemon verbindet (siehe unten).

Die Verwendung der Daemon-API ist nicht auf Server-Skripte beschränkt. Außerhalb von QF-Test kann der Daemon über ein Java-Programm oder, einfacher noch, ein Groovy-Skript angesprochen werden. Das folgende Groovy-Beispiel arbeitet mit mehreren Daemon-Instanzen und kann daher auch als Ausgangspunkt für Lasttests dienen. Nehmen wir an, dass auf verschiedenen Rechnern jeweils ein QF-Test Daemon gestartet wurde. Jeder der Daemonen soll einen bestimmten Testfall ausführen und für jeden der Testläufe soll ein Protokoll abgelegt werden (`daemon1.qrl, ..., daemonN.qrl`). Die Testsuite mit dem auszuführenden Testfall sei allen Daemon-Instanzen über ein Netzlaufwerk (hier `z:`) zugänglich.

```
import de.qfs.apps.qftest.daemon.DaemonLocator
import de.qfs.apps.qftest.daemon.DaemonRunContext
def testcase = "z:\\mysuites\\suiteA.qft#Mein Testfall"
def logfile = "c:\\mylogs\\daemon"
def timeout = 120 * 1000
def keystore = "z:\\mysuites\\mydaemon.keystore"
def password = "strengGeheim"
def locator = DaemonLocator.instance()
locator.setKeystore(keystore)
locator.setKeystorePassword(password)
def daemons = locator.locateDaemons(10000)
def contexts = []
// Start tests
for (daemon in daemons) {
    def trd = daemon.createTestRunDaemon()
    trd.setGlobal('machines', daemons.size().toString())
    def context = trd.createContext()
    contexts << context
    context.runTest(testcase)
}
// Wait for tests to terminate
for (i in 0..<contexts.size()) {
    def context = contexts[i]
    context.waitForRunState(DaemonRunContext.STATE_FINISHED, timeout)
    byte[] runlog = context.getRunLog()
    def fos = new FileOutputStream("$logfile${i + 1}.qrl")
    fos.write(runlog)
    fos.close()
    context.release()
}
```

Beispiel 25.15: Groovy-Daemon-Skript `CallDaemon.groovy`

Zur Ausführung des Groovy-Skripts werden die QF-Test Bibliotheken `qftest.jar`, `qfshared.jar` und `qflib.jar` benötigt und außerdem die Groovy-Bibliothek, die auch Bestandteil der QF-Test Installation ist. Das folgende Befehlsskript zeigt, wie das geht:

```
@echo off
setlocal
set qftestdir=c:\programs\qftest\qftest-9.0.0
set qflibdir=%qftestdir%\qflib
set classpath=%qftestdir%\lib\groovy-all.jar
set classpath=%classpath%;%qflibdir%\qftest.jar;%qflibdir%\qfshared.jar;
              %qflibdir%\qflib.jar
java -cp %classpath% groovy.ui.GroovyMain CallDaemon
```

Beispiel 25.16: Befehlsskript `calldaemon.bat` zur Ausführung von Calldaemon.groovy

Der `DaemonLocator` kann beim externen Zugriff den Keystore zur Sicherung der Kommunikation nur automatisch ermitteln, wenn die Datei `qftest.jar` (wie in diesem Befehlsskript) direkt aus dem QF-Test Verzeichnis geladen wird. Alternativ kann der Keystore wie im Groovy-Skript gezeigt mit `setKeystore` und `setKeystorePassword` direkt gesetzt werden, oder indirekt über die System-Properties `javax.net.ssl.keyStore` und `javax.net.ssl.keyStorePassword`.

Damit aus dem Daemon-Beispiel ein Lasttest wird (vgl. Kapitel 33(437)), müssen die Testläufe an mindestens einer Stelle innerhalb von "Mein Testfall" synchronisiert werden (z. B. nach dem Starten des SUT). Dazu dient die `rc`-Methode `syncThreads`:

```
def machines = rc.getNum('machines')
rc.syncThreads('startup', 60000, -1, machines)
```

Beispiel 25.17: Groovy Server-Skript Knoten zur Synchronisation der Testläufe

Die Variable `machines` bezeichnet die Anzahl der Rechner. Sie wird zum Beispiel im Testsuite Knoten mit einem Vorgabewert von `1` definiert. Bei der Ausführung der Testläufe wird sie mit dem aktuellen Wert überschrieben.

25.3 Erneute Ausführung von Knoten (Rerun)

4.1+

25.3.1 Erneute Ausführung aus dem Protokoll

Nach einem automatischen Testlauf ist das Protokoll bzw. der erstellte Report dieser Ausführung ein guter Einstiegspunkt um sich ein Bild von den Testergebnissen zu machen. Sind Fehler aufgetreten, stellt sich für Sie als Anwender oft die Herausforderung die fehlgeschlagenen Testfälle nochmals auszuführen, um das Fehlverhalten zu analysieren oder nach einer erfolgten Fehlerbehebung diesen Testfall nochmals offiziell nachzutesten. Eventuelle Nachtestergebnisse könnten bei Bedarf auch im Report angezeigt werden und zwar wahlweise zusätzlich oder an Stelle des fehlerhaften Ergebnisses des vorherigen Laufs. In manchen Fällen kann es auch einfach vom Interesse sein, einige Testfälle nochmals gezielt mit dem Variablenstand vom vorigen Lauf anzustoßen und die vorherigen Protokolle unangetastet zu lassen.

Genau hierfür bietet QF-Test nun die Möglichkeit, direkt aus dem Protokoll heraus Testfälle nochmals auszuführen. Dies erreichen Sie, indem Sie den Protokoll-Knoten oder die gewünschten Testfallsatz Knoten selektieren und die Menüaktion Testfälle nochmal ausführen im Menü Bearbeiten oder über das Kontextmenü auswählen. Alternativ können sie die Testfälle aus der Fehlerliste über den Kontextmenüeintrag Testfälle der selektierten Knoten nochmal ausführen auswählen.

Im daraufhin erscheinenden Dialog können Sie die nochmal auszuführenden Testfälle auswählen sowie in der Auswahlbox `Protokollmodus` bestimmen, ob die Ergebnisse dieser Wiederholung in das originale Protokoll des Laufes eingepflegt werden sollen oder nicht. Es werden folgende Auswahlmöglichkeiten geboten:

Auswahlmöglichkeit	Bedeutung
Testfälle ersetzen	Im Protokoll werden die Ergebnisse des vorherigen Testlaufes durch die neuen Ergebnisse ersetzt. Die vorherigen Ergebnisse fallen also aus dem Protokoll heraus. Das vorherige Protokoll wird als Sicherungskopie gespeichert und behalten.
Protokolle zusammenführen	Die neuen Ergebnisse werden in die bestehende Struktur eingefügt.
Protokoll anhängen	Die neuen Ergebnisse werden zusätzlich an das Originalprotokoll gehängt. Die Testfallsatzstruktur wird nicht berücksichtigt.
Protokolle getrennt lassen	Die neue Ausführung bekommt ihr eigenes Protokoll. Das Originalprotokoll bleibt unangetastet.

Tabelle 25.1: Auswahlmöglichkeiten für die Protokollierung einer Wiederausführung

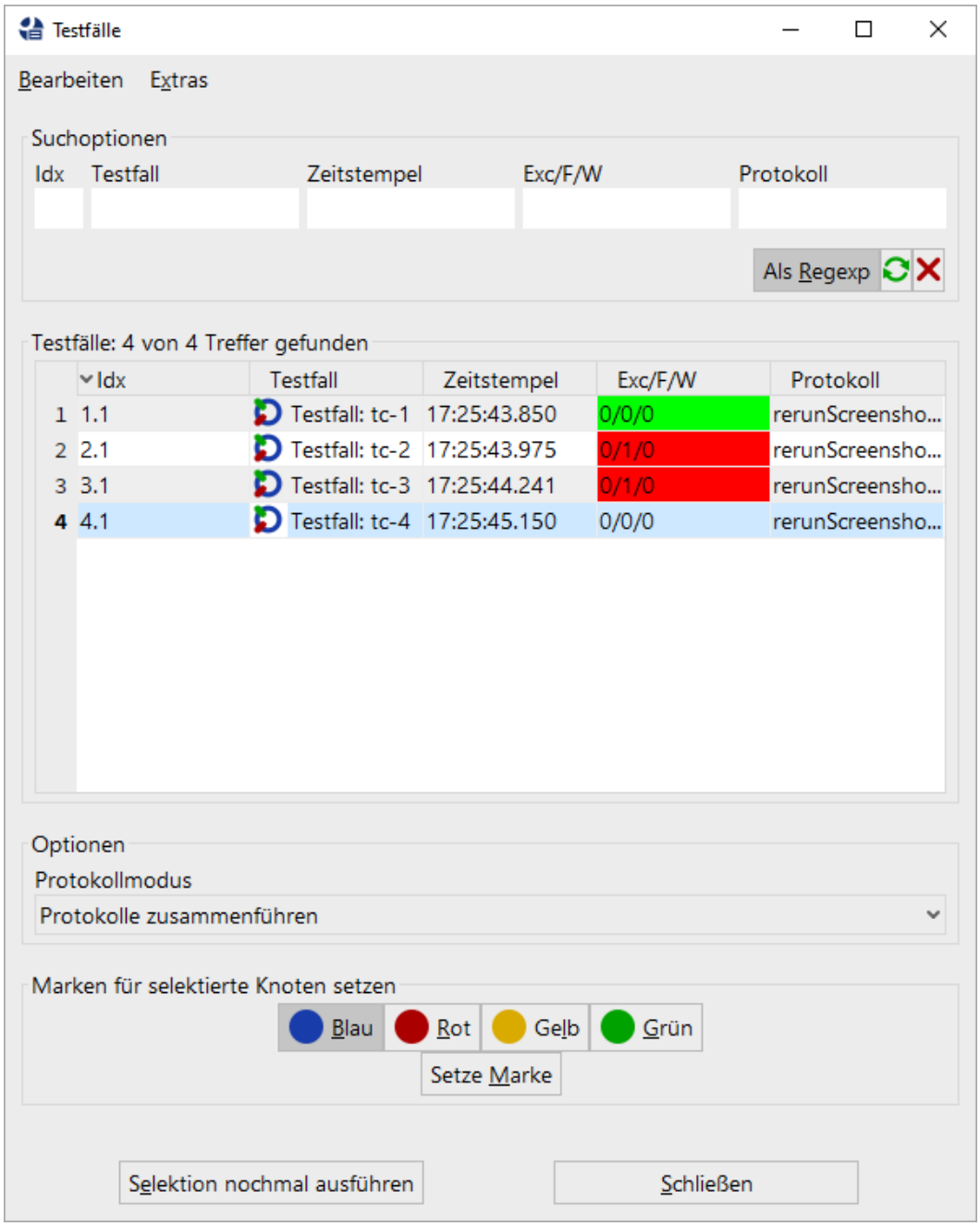


Abbildung 25.1: Dialog zur Wiederausführung von Testfällen

Diese wiederholte Ausführung verwendet für jeden Testfall dessen Variablenstände aus dem vorherigen Protokoll. Da dort nur die String-Werte der Variablen gespeichert werden, stehen die Werte der Variablen bei der wiederholten Ausführung aus dem Protokoll auch nur als Zeichenketten zur Verfügung. Die Variable `${qftest:isInRerunFromLog}` erhält für eine solche Wiederholung den Wert

`true`, womit im Test zwischen normalem Lauf und Wiederholung unterschieden werden kann.

Hinweis

Die Zusammenführung der Protokolle basiert zur Zeit auf den Namen der Testfälle und Testfallsätze. Daher müssen deren Namen eindeutig sein. Im Falle von datengetriebenen Testen sollten Sie darauf achten, dass Sie diese Eindeutigkeit der Namen über die Attribute Name für separates Protokoll bzw. Charakteristische Variablen herstellen.

25.3.2 Fehlerhafte Knoten sofort wiederholen

Während der Testautomatisierung kann es vorkommen, dass Testschritte sich volatil verhalten und manchmal das richtige Ergebnis liefern aber in anderen Fällen scheitern. Meist sind solche Fälle abhängig vom Timing und können in QF-Test mit Warten auf Komponente[876] Knoten oder Checks auf Zustände, mit Skripten, Verzögerungen oder Kontrollstrukturen stabilisiert werden. Alternativ oder zusätzlich ermöglicht QF-Test es Ihnen, solche Knoten im Fehlerfall automatisch nochmals auszuführen.

Für die Art der automatischen Ausführung im Fehlerfall können Sie bei jedem ausführbaren Knoten in der Testsuite einen Doctag im Attribut Bemerkung verwenden. Dieser Doctag kann im Wesentlichen wie folgt aussehen:

```
@rerun attempts=3;errorlevel>=ERROR;newerrorlevel=WARNING;
          handler=handlers.errorhandler
```

Beispiel 25.18: Beispiel für eine Rerun Definition

Das obige Beispiel bedeutet, dass ein Knoten im Falle eines Fehlers oder einer Exception maximal dreimal wiederholt wird bis kein Fehler und keine Exception mehr auftreten. Die Versuche, die mit Fehler oder Exception endeten, werden im Protokoll als Warnung markiert und nicht mehr als Fehler. Nach jedem Versuch, der mit Fehler oder Exception endet, wird die Prozedur `handlers.errorhandler` aufgerufen. Besteht nach dem dritten Versuch immer noch ein Fehler oder eine Exception, so wird dieser letzte Zustand protokolliert.

Falls Sie den aktuellen Versuch des Reruns herausfinden möchten, können Sie die Variable `reruncounter` aus der `qftest` Variablengruppe verwenden, siehe Abschnitt 6.8[127].

Für das @rerun Doctag können verschiedene Parameter wie `attempts` oder `errorlevel` mit folgender Bedeutung verwendet werden:

attempts

Die maximale Anzahl der Wiederholungen.

errorlevel (optional)
Gibt an, auf welche Arten von Fehlverhalten reagiert werden soll. Hier können Sie entweder die Fehlerstufen `EXCEPTION`, `ERROR` oder `WARNING` angeben, entweder exakt (mit =) oder mit > oder >=. Mit `errorlevel=ERROR` wird der Knoten nur bei Fehlern neu ausgeführt wird, bei `errorlevel>=ERROR` bei Fehlern und Exceptions. Falls Sie nichts angeben wird der Wert `errorlevel>=ERROR` als Standard herangezogen.

newerrorlevel (optional)
Legt die Fehlerstufe für die Protokollierung des initialen Laufs und eventueller weiterer fehlgeschlagener Versuche fest. Zulässige Werte sind wiederum `EXCEPTION`, `ERROR` oder `WARNING` sowie zusätzlich `NOLOG` und `KEEP`. Die Stufe `NOLOG` bedeutet, dass die fehlerhaften Versuche aus dem Protokoll gelöscht werden, so dass nur der letzte Versuch im Protokoll auftaucht. Der Wert `NOLOG` sollte mit großer Vorsicht eingesetzt werden. Der Wert `KEEP` gibt an, dass die originalen Fehlerstufen im Protokoll behalten werden. Falls Sie nichts angeben wird der Wert `WARNING` als Standard herangezogen.

handler (optional)
Definiert den Namen einer Prozedur, welche aufgerufen werden soll, falls das Fehlverhalten auftritt. Diese Prozedur wird nach jedem fehlgeschlagenen Durchlauf ausgeführt.

reusevariables (optional, default=true)
Hier geben Sie an, ob der Variablenstand vom Beginn des ersten Durchlaufs wiederverwendet werden soll. Steht der Parameter auf `false` wird der aktuelle Variablenstand genutzt.

logmessages (optional, default=true)
Ist dieser Parameter auf `true` gesetzt, wird am Anfang eines jeden Versuchs sowie nach erfolgreicher bzw. fehlerhafter Ausführung eine Nachricht in das Protokoll geschrieben. Zusätzlich wird bei jedem Versuch eine Anmerkung im Protokoll für diesen Knoten gesetzt.

logmessagesintoreport (optional, default=true)
Ist dieser Parameter und der Parameter `logmessages` auf `true` gesetzt, werden die Meldungen auch in den Report geschrieben.

keepfirst (optional, default=false)
Falls dieser Wert auf `true` gesetzt wird, wird das erste gefangene Fehlerverhalten mit dem Originalzustand protokolliert. Im Falle von weiteren Fehlern wird der Parameter `newerrorlevel` herangezogen.

exceptiontype (optional)
Falls Sie einen speziellen Exceptiontype fangen wollen, können Sie

diesen Typ hiermit angeben, z.B. `CheckFailedException` oder nur `ClientNotConnected` für eine `ClientNotConnectedException`. Dieser Parameter sollte nur verwendet werden, wenn Sie `Exception` als Wert für `errorlevel` gesetzt haben. Details zu Exceptions finden Sie beim Catch(707) Knoten.

exceptionmessage (optional)

Falls Sie eine bestimmte Exception mit bestimmter Nachricht abfangen wollen, können Sie diese Nachricht hier angeben. Dieser Parameter sollte nur verwendet werden, wenn Sie `Exception` als Wert für `errorlevel` gesetzt haben. Details zu Exceptions finden Sie beim Catch(707) Knoten.

exceptionregex (optional)

Wenn dieser Wert `true` ist, dann stellt der Parameter `exceptionmessage` einen regulären Ausdruck dar. Dieser Parameter sollte nur verwendet werden, wenn Sie `Exception` als Wert für `errorlevel` gesetzt haben und einen Wert für `exceptionmessage` angegeben haben. Details zu Exceptions finden Sie beim Catch(707) Knoten.

exceptionlocalized (optional)

Wenn dieser Wert `true` ist, dann steht der Parameter `exceptionmessage` für die lokalisierte Fehlermeldung der Exception. Dieser Parameter sollte nur verwendet werden, wenn Sie `Exception` als Wert für `errorlevel` gesetzt haben und einen Wert für `exceptionmessage` angegeben haben. Details zu Exceptions finden Sie beim Catch(707) Knoten.

Kapitel 26

Verteilte Entwicklung von Tests

Die vorhergehenden Kapitel waren alle auf das Erstellen und Bearbeiten einzelner Testsuiten ausgerichtet. Für das Testen von umfangreichen Applikationen ist dies eventuell nicht ausreichend. Es gibt mindestens zwei Szenarios, für die das Aufteilen von Tests in mehrere Testsuiten von entscheidender Bedeutung ist:

- Mehrere Entwickler arbeiten simultan an der Erstellung von Tests. Um Redundanz zu vermeiden, sollten separat erstellte Testsuiten nach Möglichkeit gemeinsame Prozeduren und Komponenten verwenden. Eine Testsuite kann zu einem bestimmten Zeitpunkt aber immer nur von einer Person bearbeitet werden.
- Tests werden schlicht und einfach zu groß und unhandlich. Bei ausgedehnten Testläufen reicht eventuell der verfügbare Speicher für Protokolle nicht aus. Eine Testsuite mit einer Vielzahl von Tests kann sehr unübersichtlich werden. Außerdem kann es wünschenswert sein, dass gewisse Tests sowohl als Teil des gesamten Tests, als auch für sich alleine ausgeführt werden können.

QF-Test bietet einige nützliche Hilfsmittel für die verteilte Entwicklung mit deren Hilfe Sie Tests auf mehrere Testsuiten verteilen können. Mehrere Entwickler können damit an einzelnen Teilen von Tests arbeiten und später ihre Entwicklungen koordinieren um die Komponenten ihrer Testsuiten zusammenzuführen und Bibliotheken von gemeinsam genutzten Prozeduren zu erstellen.

Dieses Kapitel erläutert zunächst die verschiedenen Mechanismen zur verteilten Entwicklung und deren Zusammenspiel. Der abschließende Abschnitt enthält eine knappe und präzise Schritt-für-Schritt Anleitung, um große Testprojekte mit QF-Test umzusetzen.

26.1 Der Aufruf einer Prozedur in einer anderen Testsuite

Es ist möglich, Prozeduren[672] und Komponenten[930] außerhalb der aktuellen Testsuite zu referenzieren. Diese Referenzen können explizit oder implizit über Includedateien angegeben werden.

- Explizite Referenzen verwenden die gleiche Syntax, die auch in URLs zum Einsatz kommt, um eine bestimmte Stelle auf einer Webseite anzuspringen: Der Name der Datei wird, durch ein '#'-Zeichen getrennt, dem Name der Prozedur[676] Attribut eines Prozeduraufruf[675] Knotens oder dem Attribut QF-Test ID der Komponente eines von einer Komponente abhängigen Knotens vorangestellt. Aus `PackagePfad.Prozedur` wird somit `Suite#PackagePfad.Prozedur`
- Implizite Referenzen bedienen sich des Inkludierte Dateien[596] Attributs des Testsuite[595] Knotens. Kann ein Knoten in der aktuellen Suite nicht ermittelt werden, sucht QF-Test nach einer passenden Prozedur oder Komponente in allen direkt oder indirekt eingebundenen Testsuiten (eine Testsuite wird als indirekt eingebunden bezeichnet, wenn Sie als Includedatei in einer direkt von der aktuellen Suite eingebundenen Datei aufgeführt ist).

Eine Testsuite, die einen Knoten in einer anderen Testsuite referenziert, wird von dieser Testsuite abhängig. Bei einer Änderung des Namens der Prozedur oder der QF-Test ID der Komponente muss die verweisende Testsuite angepasst werden, ansonsten ist der Bezug falsch und die Testsuite funktioniert nicht mehr richtig. QF-Test führt solche Anpassungen automatisch durch, sofern es von der Beziehung weiß. Idealerweise sollten beide Testsuiten dem selben Projekt angehören, denn QF-Test verwaltet automatisch alle Include-Beziehungen und alle expliziten Referenzen innerhalb eines Projekts. Andernfalls muss die aufrufende Suite im Attribut Abhängige Dateien (umgekehrte Includes)[597] des Testsuite Knotens der referenzierten Testsuite aufgeführt sein.

Zwar sind implizite Referenzen im Normalfall flexibler und praktischer, es kann damit aber auch schwierig sein, den Überblick zu behalten, wo sich eine referenzierte Prozedur oder Komponente eigentlich befindet. Feststellen kann man dies durch die über das Kontextmenü erreichbaren Funktionen "Prozedur finden" (Strg-P) und "Komponente finden" (Strg-W). Zusätzlich bietet QF-Test die Menüeinträge Operationen→Referenzen explizit machen und Operationen→Referenzen implizit machen, mit deren Hilfe Sie schnell zwischen den beiden Darstellungen umschalten können, ohne die tatsächlich referenzierten Knoten zu verändern.

Im expliziten wie im impliziten Fall kann die referenzierte Testsuite entweder ein relativer oder ein absoluter Dateiname sein. Relative Dateinamen werden zunächst relativ zur

aufrufenden Suite aufgelöst. Schlägt dies fehl, werden die Dateien des Bibliothekspfades (vgl. option Verzeichnisse mit Testsuite-Bibliotheken[503]) der Reihe nach durchsucht. Verwenden Sie grundsätzlich - auch unter Windows - das '/'-Zeichen als Verzeichnistrenner. QF-Test verwendet dann zur Laufzeit das korrekte Trennzeichen für das jeweilige System. Dadurch bleiben die Testsuiten auf verschiedenen Systemen lauffähig.

Hinweis

Ihre Package und Prozedur Namen sollten die Zeichen '\' und '#' nicht enthalten. Wenn doch, müssen diese im Prozeduraufruf geschützt werden. Näheres zu diesem Thema finden Sie in Abschnitt 49.5[1025].

Wenn Sie die Prozedur für einen Prozeduraufruf oder die Komponente für einen Event in dem Auswahldialog festlegen, bietet QF-Test alle derzeit geöffneten Testsuiten zur Auswahl an. Wenn Sie eine Prozedur oder eine Komponente aus einer anderen Testsuite selektieren, erzeugt QF-Test automatisch die korrekte Referenz. Bei einem späteren Ablauf des Tests wird die referenzierte Testsuite gegebenenfalls automatisch nachgeladen, falls sie sich noch nicht im Speicher befindet.

Während der Ausführung eines Tests verwaltet QF-Test alle beteiligten Testsuiten in einem Stapel. Wird eine Prozedur in einer anderen Testsuite aufgerufen, kommt die neue Suite nach oben auf den Stapel. Ist die Prozedur abgearbeitet, wird die Testsuite wieder vom Stapel entfernt. Wann immer während der Ausführung der Prozedur ein Fenster oder eine Komponente über ihre QF-Test ID angesprochen werden, durchsucht QF-Test den Stapel der Testsuiten von oben nach unten, d.h. zuerst in der aufgerufenen Suite, dann in der aufrufenden, wobei Includedateien berücksichtigt werden. Dieser Prozess ist nicht unkompliziert und Sie sind gut beraten, Includes nicht zu tief zu verschachteln. Für den Fall dass Sie bei auftretenden Problemen ein tieferes Verständnis für dieses Thema benötigen, finden Sie in Abschnitt 49.6[1026] eine ausführliche Erklärung.

26.2 Die Verwaltung von Komponenten

Wie bereits in Kapitel 5[47] mehrfach betont wurde, sind die Komponenten der zentrale Bestandteil einer Testsuite. Bei Änderungen am SUT sind diese am ehesten betroffen. Wenn die Änderungen ein solches Ausmaß annehmen, dass QF-Test sich nicht mehr automatisch darauf einstellen kann, müssen die Komponenten von Hand angepasst werden. Aus diesem Grund sollten Sie bei den Komponenten noch mehr als bei jedem anderen Teil der Tests auf das Vermeiden von Redundanz achten.

Wenn Sie Ihre Tests auf mehrere Testsuiten verteilen, sollten Sie daher versuchen, die Komponenten in einer zentralen Testsuite vorzuhalten und diese in den anderen Testsuiten als Includedatei einzubinden. Für sehr große Applikationen kann es sinnvoll sein, die Komponenten Hierarchie in Teile zu zerlegen, die jeweils einen zusammengehörenden Teil der Oberfläche des SUT repräsentieren.

Diese zentrale Bibliothek von Komponenten zu verwalten ist nicht trivial. Die dabei auf-

tretenden Probleme können wie folgt mit QF-Test gelöst werden:

- Wenn mehrere Entwickler von Tests gleichzeitig neue Komponenten aufnehmen, können diese nicht sofort in die zentrale Suite integriert werden, da immer nur ein Anwender diese gleichzeitig bearbeiten kann. Stattdessen müssen Komponenten später in die zentrale Suite importiert werden, wenn die neuen Tests stabil laufen. Dies wird im folgenden Abschnitt erläutert.
- Wenn sich das SUT ändert, müssen eventuell Komponenten in der zentralen Suite angepasst werden. Werden dabei QF-Test IDs von Komponenten geändert, werden dadurch Referenzen auf diese Komponenten aus anderen Testsuiten ungültig. Um das zu vermeiden, passt QF-Test diese Referenzen automatisch an, vorausgesetzt, die Testsuiten, die von der zentralen Suite abhängen sind im Moment geladen, gehören zum selben Projekt oder sind im Attribut Abhängige Dateien (umgekehrte Includes)(597) des Testsuite(595) Knotens der zentralen Suite aufgeführt.

26.3 Verschmelzen von Testsuiten

Testsuiten können durch Importieren einer Testsuite in eine andere miteinander verschmolzen werden. Sie erreichen diese Funktion über den Menüeintrag Datei→Importieren....

Sie können auswählen, welche Bereiche der Testsuite zusammengeführt werden sollen.

Um die Konsistenz von Aufrufen sicherzustellen, sollten Sie auf eine korrekte Konstellation von Includes- und Umgekehrten Includes achten. Mehr zum Arbeiten mit mehreren Testsuiten finden Sie im Kapitel 37(463).

26.3.1 Importieren von Komponenten

Beim Importieren von Komponenten werden die beiden Komponentenhierarchien miteinander verschmolzen. Hierfür werden alle Fenster und Komponenten der importierten Suite in die Hierarchie der Komponenten der importierenden Suite eingefügt. Komponenten, die bereits vorhanden sind, werden nicht kopiert. QF-Test ID Konflikte, zum Beispiel wenn identische Komponenten in beiden Testsuiten verschiedene QF-Test IDs oder verschiedene Komponenten die selben QF-Test IDs haben, löst QF-Test automatisch, indem es die QF-Test ID der importierten Komponente verändert.

Anschließend werden alle Fenster und Komponenten aus der importierten Suite entfernt. Knoten der importierten Suite, die sich auf diese Komponenten bezogen haben, werden

automatisch angepasst. Idealerweise sollte die importierte Suite die importierende Suite als Includedatei einbinden, so dass dabei keine expliziten Referenzen entstehen.

26.3.2 Importieren von Prozeduren und Testfällen

3.3+

Analog zum Importieren von Komponenten können auch Prozeduren, Packages, Abhängigkeiten und Testfälle sowie Testfallsätze importiert werden, indem Sie 'Prozeduren' oder 'Tests' im Dialog auswählen. Achten Sie allerdings hierbei wieder auf die Konsistenz Ihrer Testsuite, z.B. macht es kaum Sinn, Prozeduren ohne benutzte Komponenten zu importieren.

Falls Sie nur eine bestimmte Prozedur oder einen Testfall importieren wollen, so können Sie den 'Einzelimport' Knopf auf den Importdialog auswählen und im erscheinenden Detaildialog den gewünschten Knoten auswählen.

26.4 Verteilte Entwicklung von Tests

Es gibt keinen *goldenen Weg*, um die Entwicklung von Tests zu organisieren, aber eine Vorgehensweise, die sich bewährt hat, ist die folgende:

- Beginnen Sie mit einer zentralen Testsuite, welche die nötige Funktionalität zum Starten und Stoppen des SUT sowie einen grundlegenden Satz von Tests und Prozeduren umfasst. Diese Suite wird Ihre Mastersuite, die alle Komponenten enthalten wird.
- Stellen Sie sicher, dass Ihre Entwickler die Wichtigkeit von `setName()` verstanden haben und dass, wo erforderlich, eindeutige Namen nach einheitlichem Schema vergeben werden. Wenn `setName()` nicht verwendet werden kann, setzten Sie `ComponentNameResolver` ein, um dies zu erreichen (vgl. Abschnitt 54.1.7(1163)). Sie sollten in der Lage sein, neue Sequenzen ohne viel Aufwand aufzunehmen und ohne dass dabei kleine Änderungen am SUT die Hierarchie der Komponenten völlig durcheinanderbringen.
- Verlagern Sie möglichst viel Funktionalität in Prozeduren, insbesondere häufig genutzte Sequenzen und die Vorbereitungs- und Aufräumsequenzen für das SUT.
- Um neue Tests zu erstellen, starten Sie mit einer leeren Testsuite. Binden Sie die Mastersuite über das Attribut Inkludierte Dateien(596) des Testsuite(595) Knotens ein. Erstellen Sie Vorbereitung und Aufräumen Knoten zum Starten und Stoppen des SUT indem Sie die entsprechenden Prozeduren in der Mastersuite aufrufen.

- Erstellen Sie die erforderlichen Tests. Beim Aufnehmen von Sequenzen werden wenn möglich die Komponenten in der Mastersuite verwendet. Neue Komponenten werden der neuen Suite hinzugefügt, so dass die Mastersuite zu diesem Zeitpunkt nicht angefasst werden muss.
- Wo möglich, verwenden Sie die Prozeduren der Mastersuite für allgemeine Operationen.
- Wenn Ihre neuen Tests komplett sind und zufriedenstellend funktionieren, importieren Sie alle erforderlichen Knoten der neuen Testsuite in die Mastersuite. Vor allem sollten Sie die Komponenten importieren. Damit stellen Sie sicher, dass alle Komponente Knoten, die neu aufgenommen wurden, in die Mastersuite integriert werden. Die bestehenden Komponenten der Mastersuite werden dabei nicht verändert, so dass keine anderen von der Mastersuite abhängigen Testsuiten betroffen sind.
- Nach erfolgtem Komponenten-Import können Sie nun auch alle bzw. einzelne Prozeduren in die Mastersuite importieren.
- Es gibt jetzt mehrere Möglichkeiten, wie Sie die Sequenzen aus Events und Checks organisieren können, die die eigentlichen Tests bilden. In jedem Fall ist es eine gute Idee, alles in Prozeduren und Packages zu verpacken, die entsprechend Ihrem Testplan strukturiert sind. Danach sollten die Testfallsatz bzw. Testfall Knoten der obersten Ebene in der Mastersuite und der neuen Suite nur noch aus der gewünschten Struktur von Testfallsatz, Testfall, Testschritt und Sequenz Knoten bestehen, welche die Prozeduraufrufe der eigentlichen Testfälle enthalten. Ein solcher Aufbau hat mehrere Vorteile:
  - Alle Tests sind klar strukturiert.
  - Sie können sehr einfach Testläufe von unterschiedlicher Komplexität und Laufzeit zusammenstellen.
  - Sie haben die Möglichkeit, Testfälle in separate Testsuiten auszulagern. Diese ”Test-Bibliotheken” sollten die Mastersuite als Include Datei einbinden und selbst keine Komponenten enthalten. Damit können Sie die Tests so organisieren, dass Sie wahlweise über die Mastersuite alle Tests, oder nur die einzelnen Testsuiten alleine ausführen können.
  - Die Tests können von verschiedenen Personen unabhängig voneinander gepflegt werden, sofern Änderungen an der Mastersuite miteinander abgestimmt werden.
- Wenn Sie Ihre neu erstellten Tests in der neuen Testsuite belassen wollen, anstatt sie in die Mastersuite zu übernehmen, sorgen Sie dafür, dass beide Testsuiten zum selben Projekt gehören oder tragen Sie die neue Suite im

Abhängige Dateien[597] Attribut des Testsuite Knotens der Mastersuite ein, um QF-Test explizit auf diese neue Abhängigkeit hinzuweisen.

- Um einen bestehenden Test zu ändern oder zu erweitern, verfahren Sie nach dem selben Schema. Nehmen Sie zusätzliche Sequenzen nach Bedarf auf, importieren Sie die neuen Komponenten in die Mastersuite.
- Wenn sich Ihr SUT auf eine Weise ändert, die Anpassungen an den Komponenten erfordert, sollten Sie Ihre Tester koordinieren. Stellen Sie zunächst sicher, dass alle Testsuiten, die die Mastersuite direkt oder indirekt einbinden, zum selben Projekt wie die Mastersuite gehören oder im Attribut Abhängige Dateien des Testsuite Knotens der Mastersuite aufgeführt sind. Wenn bei den Anpassungen QF-Test IDs von Komponenten geändert werden, muss QF-Test die abhängigen Testsuiten entsprechend modifizieren. Daher sollten diese zu diesem Zeitpunkt nicht von Anderen bearbeitet werden.
- Das Dateiformat der Testsuiten von QF-Test ist XML und damit normaler Text. Dieses Format eignet sich ausgezeichnet zur Verwaltung in Systemen zum Versionsmanagement. Änderungen an einigen QF-Test ID der Komponente Attributen der abhängigen Testsuiten lassen sich normalerweise problemlos mit anderen Änderungen integrieren, so dass die Koordination der Entwickler bei Einsatz eines solchen Werkzeugs nicht zwingend erforderlich ist.

Natürlich kann dieses Schema auch auf mehrere Mastersuites erweitert werden, um verschiedene Teile oder Aspekte einer Applikation zu testen. Dabei kann es sich auszahlen, wenn sich die Komponentenhierarchien dieser Testsuiten möglichst wenig überschneiden, denn dadurch reduziert sich der Pflegeaufwand, wenn sich die grafische Oberfläche des SUT signifikant verändert.

26.5 Statische Validierung von Testsuiten

3.1+

Im Laufe eines Projektes wird es immer wieder zu Veränderungen, Refactoring oder Löschungen von Knoten in Ihrer Testsuite-Struktur kommen. Sie werden sicher manchmal Prozeduren umbenennen oder diese löschen, falls diese nicht mehr benötigt werden.

26.5.1 Ungültige Referenzen vermeiden

In solchen Fällen ist es äußerst wichtig, dass Sie dabei alle Aufrufe der entsprechenden Prozedur anpassen, um Ihre Testsuiten weiterhin lauffähig zu halten. Für diesen Zweck passt QF-Test beim Umbenennen oder Verschieben von Knoten alle Referenzen auf Wunsch automatisch an.

Wenn Sie nun sicherstellen wollen, dass Ihre Teststruktur keine Verweise auf nicht existierende Prozeduren enthält, können Sie dies mit dem Kommando ”Referenzen analysieren” bewerkstelligen. Diese statische Validierung wird Ihnen nach einer Analyse auch die Ergebnisse in einem Dialog zeigen, aus dem ersichtlich ist, welche Referenzen noch funktionieren und welche nicht mehr.

Sie können diese Analyse starten, indem Sie nach einem Rechtsklick den Menüeintrag Weitere Knotenoperationen→Referenzen analysieren... oder den entsprechenden Eintrag aus dem Hauptmenü unter Operationen auswählen. Diese Validierung ist auch im Batchmodus möglich.

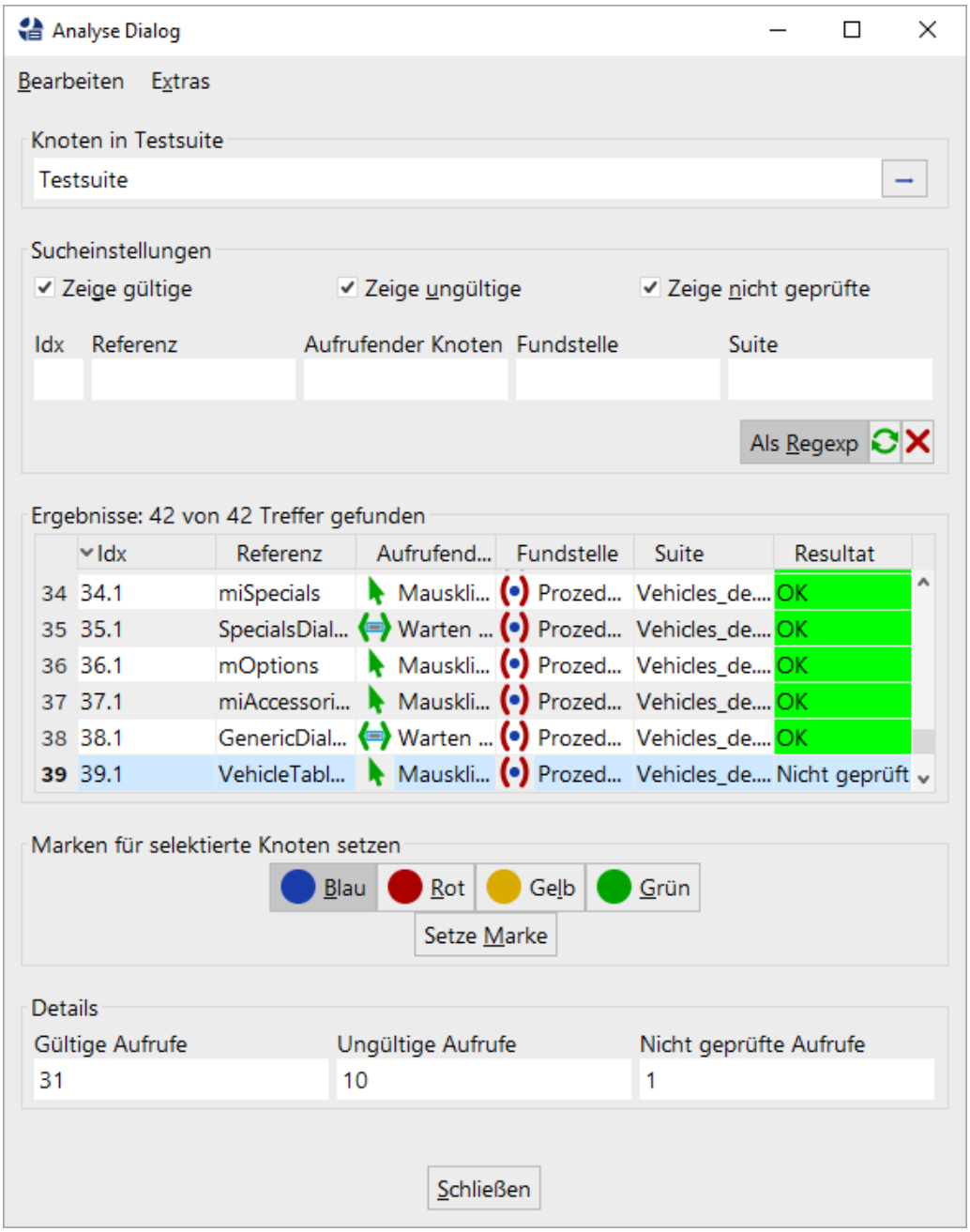


Abbildung 26.1: Ergebnis einer Analyse

3.5+ QF-Test bietet Ihnen außerdem die Möglichkeit, Ihre Testsuiten auf Duplikate zu überprüfen oder nach leeren Packages oder Prozeduren zu suchen. Es ist auch möglich, Knoten auf ungültige Zeichen in deren Namen zu überprüfen.

Diese Art der statischen Validierung ist für Prozeduren, Abhängigkeiten, Testfälle, Testfallsätze und Komponenten und deren Referenzen verfügbar.

26.5.2 Ungenutzte Prozeduren finden

4.0.3+

Während der Testentwicklung kann es immer wieder vorkommen, dass Prozeduren, die in früheren Versionen der Tests verwendet wurden, in neueren Version nicht mehr benötigt werden. Wenn Sie solche Prozeduren nicht sofort löschen, kann Ihre Testsuite wachsen. In manchen Fällen könnten Sie hier nun das Gefühl bekommen, dass Sie die Übersicht über die Prozeduren verloren haben könnten. Es gibt nun die Möglichkeit ungenutzte Prozeduren und Abhängigkeit zu finden. Hierfür klicken Sie mit der rechten Maustaste auf Testsuite oder Prozeduren und wählen Weitere Knotenoperationen→Ungenutzte aufrufbare Knoten finden... aus. Diese Operation liefert nun einen Report über alle ungenutzten Prozeduren und Abhängigkeiten. Jetzt können Sie entscheiden, was Sie damit tun wollen.

Manchmal ist es auch vom Vorteil solche ungenutzten Knoten einfach zu löschen, hierfür können Sie im Menü auch Weitere Knotenoperationen→Ungenutzte aufrufbare Knoten entfernen auswählen.

Kapitel 27

Automatisierte Erstellung von Basisprozeduren

3.0+

27.1 Einführung

Am Beginn eines typischen QF-Test Projektes werden die ersten Schritte aufgezeichnet und auch meistens erfolgreich ausgeführt. Nach den ersten Erfolgen und einigen solcher Aufnahmen bemerkt man, dass einfaches Aufnehmen oder ein Kopieren/Einfügen Ansatz einige Probleme bereiten, die Tests wartbar zu halten. Denken Sie nur an eine Änderungen im Haupt-Panel des Workflows. In diesem Fall muss beinahe jeder Test angepasst werden. Wegen solcher Probleme empfehlen wir, die Tests modularisiert zu entwickeln und Prozeduren und Variablen einzusetzen. Für eine genauere Beschreibung modularisierter Testentwicklung siehe Abschnitt 8.5(157).

In Projekten, die eine große Menge an Dialogen und grafischen Elementen beinhalten, ist es mitunter empfehlenswert, solche Prozeduren in komponentenspezifische Prozeduren, z.B. ”Klicke Button OK”, und in eigenständige Workflow Prozeduren, z.B. ”Ein Fahrzeug anlegen”, aufzuteilen. Dieser Ansatz ermöglicht es, dass neue Testfälle relativ schnell angelegt werden können, jedoch müssen vorher diese Basisprozeduren erstellt werden.

QF-Test liefert den Procedure Builder, der genau diese Basisprozeduren automatisiert erstellt. Wenn Sie diesen Procedure Builder verwenden, wird sich die Zeit für Aufnahme und Erstellung der Prozeduren für die grafischen Elemente drastisch verkürzen. Damit kann sich der Tester auf seine eigentliche Aufgabe, Testfälle zu erzeugen und zu planen bzw. Testdaten bereitzustellen, konzentrieren.

27.2 Die Verwendung vom Procedure Builder

Um die Basisprozeduren automatisiert zu erzeugen, müssen Sie folgende Schritte ausführen:

- Starten Sie das SUT von QF-Test aus.
- Gehen Sie im SUT zum Fenster oder der Komponente, für welche Sie die Basisprozeduren erzeugen wollen.
- Drücken Sie in QF-Test den 'Prozeduren erstellen'  Knopf, wählen Sie den entsprechenden Menüeintrag aus oder nutzen Sie den Hotkey für Prozedurenaufnahme[529].
- Führen Sie einen Rechtsklick auf die entsprechende Komponente im SUT durch.
- Wählen Sie den entsprechenden Aufnahmemodus aus.
- Stoppen Sie die Prozedurenaufnahme mittels Klick auf den 'Prozeduren erstellen' Knopf, mittels Auswahl des entsprechenden Menüeintrages in QF-Test oder über den Hotkey für Prozedurenaufnahme[529].

Nun sollten Sie die neu erstellten Packages unter den Prozeduren[682] Knoten der aktuellen Testsuite, in der Sie die Aufzeichnung gestoppt haben, finden. Diese Packages beinhalten die erstellten Prozeduren für die Komponenten. Standardmäßig wird ein Package `procbuilder` erzeugt. Falls dieses schon existiert, wird ein Package `procbuilder1` erzeugt und so weiter. Wenn Sie dieses Package öffnen, finden Sie weitere Packages für bestimmte Funktionen wie `check`, `get`, `select`, `wait` etc.. Diese beinhalten die Prozeduren für die Komponenten, welche Sie aufgenommen haben. Unter dem `check` Package finden Sie noch eine weitere Ebene für unterschiedliche Checkarten. Diese Struktur ist die Standardstruktur, welche Sie allerdings anders konfigurieren können. Eine Beschreibung hierfür finden Sie im nächsten Abschnitt.

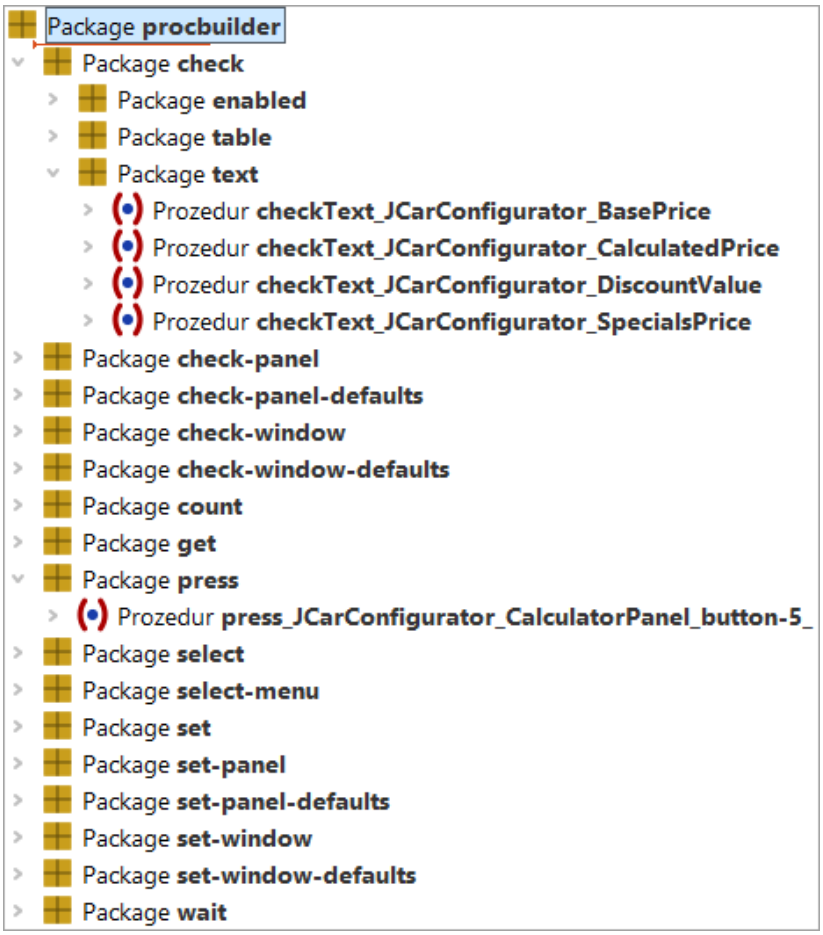


Abbildung 27.1: Aufgezeichnete Prozeduren

27.3 Konfiguration des Procedure Builder

Die Erstellung der Prozeduren wird über eine Template Testsuite gesteuert, diese Testsuite finden Sie unter `qftest-9.0.0/include/procbuilderdef.qft`. Diese Datei sollte in ein projektspezifisches Verzeichnis kopiert werden, falls Sie diese anpassen möchten. Den Speicherort dieser Datei können Sie dann in den QF-Test Einstellungen unter Konfigurationsdatei für die Prozedurenaufnahme [529] konfigurieren.

Die Template Testsuite beinhaltet Prozedurvorlagen für die am meisten verwendeten GUI Elemente und deren Aktionen. Wenn Sie eine eigene Aktion definieren möchten, können Sie einfach eine entsprechende Prozedur einfügen.

Die Testsuite selbst hat eine bestimmte Struktur. Diese Struktur ist im Abschnitt 27.3.1 [371] beschrieben. Die Definition erlaubt es den Tester, Prozedurvorlagen für

Komponenten einer bestimmten Klasse zu erstellen oder für Prozeduren, die sich auf alle Elemente eines Fenster bzw. Dialoges beziehen, eigene Prozeduren zu definieren.

Sie finden auch einige Beispielkonfigurationen unter `qftest-9.0.0/demo/procbuilder`.

27.3.1 Die Procedure Builder Definitionsdatei

Die automatisierte Erstellung der Basisprozeduren liefert je nach Komponenten unterschiedliche Prozeduren. Ein Textfeld benötigt eine Setzen-Methode für den Text, ein Button braucht eine Drück-Methode und ein Fenster benötigt eine Setzen-Methode, die die Setzen-Methoden aller enthaltenen Elemente aufruft etc..

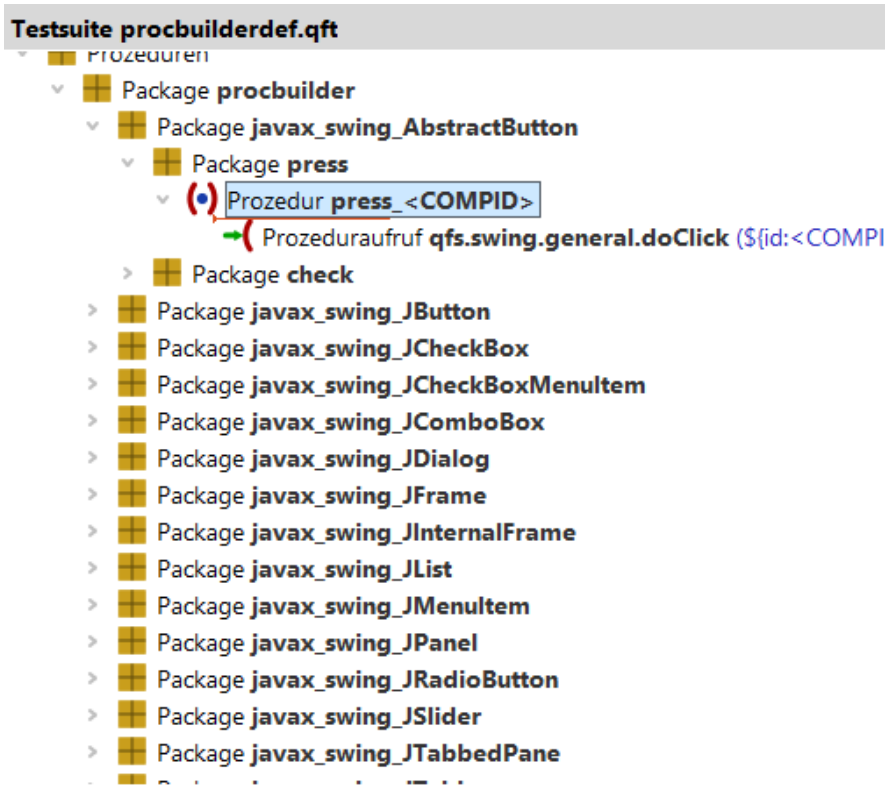


Abbildung 27.2: Die Procedure Builder Definitionsdatei

Das oberste Package im Prozeduren[682] Knoten ist der Name des neuerstellen Packages. Der Defaultname hierfür ist `procbuilder`. Falls das Package `procbuilder` bereits besteht, wird ein Package `procbuilder1` erzeugt usw..

Die nächste Ebene ist die Klassenebene. Hier können Sie ein Package pro Klasse definieren. Das Package benötigt den kompletten Klassenname, wobei jedoch alle '.' durch

'_' ersetzt werden, weil Punkte in Packagenamen nicht erlaubt sind. Der Procedure Builder erzeugt die Prozeduren auch für die abgeleiteten Klassen. Falls Ihre Klasse einen '_' beinhalten sollte, dann müssen Sie diesen mit '_' kennzeichnen.

Die darauffolgenden Ebenen können frei definiert werden, weil diese die Packagestruktur der Prozeduren vorgeben.

Am Ende müssen Sie jedoch eine Prozedur definieren, welche die entsprechenden Schritte beinhaltet.

Sie können auch einige Variablen, z.B. `<COMPID>`, in der Definition verwenden.

Variablen, z.B. die aktuelle QF-Test ID der Komponente oder der Komponentenname, können für den Prozedurnamen verwendet werden. Des Weiteren können Sie auf die aktuellen Werte eines Textfeldes oder den aktuellen Status einer Checkbox zugreifen. Es ist sogar möglich, die gesamte Packagestruktur variabel zu gestalten. Für eine Übersicht über alle verwendbaren Variablen siehe Kapitel 56(1297).

Kapitel 28

Anbindung an Testmanagementtools

3.0+

QF-Test bietet einige pragmatische Ansätze um Testfälle zu verwalten. Sie können eine Übersicht über alle Testfälle generieren oder auch Testfälle in QF-Test dokumentieren. In größeren Projekten allerdings könnte es Bedarf für ein eigenes Testverwaltungssystem geben, um z.B. den Entwicklungsstatus der einzelnen Tests zu sehen oder die Tests bzw. Ergebnisse mit Anforderungen oder Testfallbeschreibungen zu verbinden. Neben der Testplanung und Testergebnisverwaltung könnte auch ein eigenes Testausführungssystem im Einsatz sein, welches z.B. die Auslastung der Testsysteme während verschiedener Testläufe überwacht.

QF-Test bietet nicht so viele Möglichkeiten wie dererlei spezialisierte Werkzeuge, jedoch verbessert es sich von Version zu Version. Die Integration von QF-Test in ein solches Testverwaltungssystem ist mittels dem Batch-Modus oder den Daemon-Modus sehr einfach. Für mehr Informationen über Batch bzw. Daemon-Modus, siehe Kapitel 25(340).

Die folgenden Kapitel beschreiben einige Musterlösungen, welche wir für etablierte Testverwaltungssysteme liefern. Wenn Sie das Testverwaltungssystem, welches Sie im Einsatz haben, nicht finden können, kontaktieren Sie unser Support-Team um Hinweise und Tipps für eine mögliche Integration zu bekommen.

28.1 HP ALM - Quality Center

28.1.1 Einführung

Die aktuelle Integration von QF-Test und HP ALM - Quality Center baut auf dem von Quality Center gelieferten VAPI-XP-TEST Typ auf.

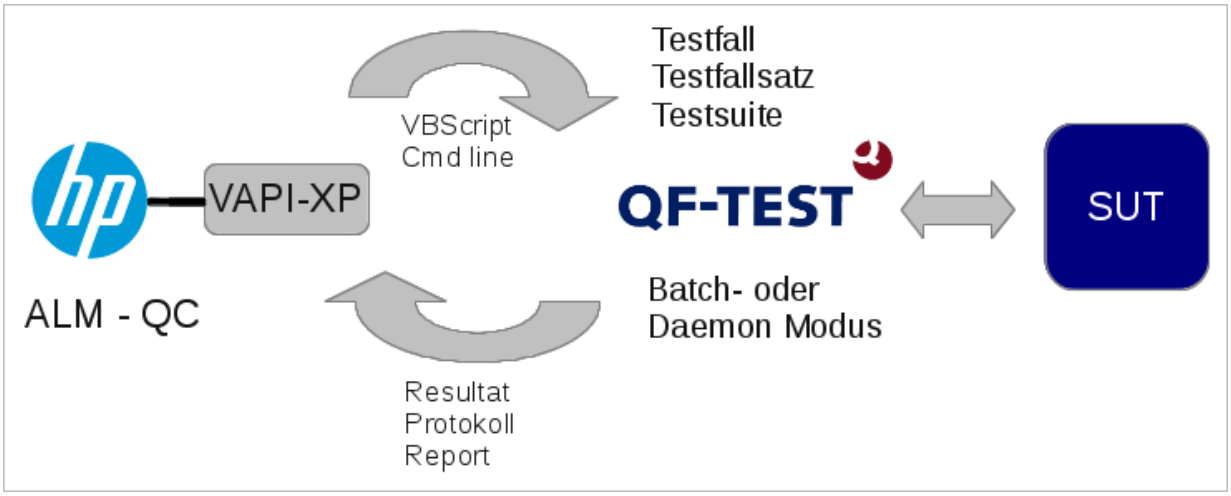


Abbildung 28.1: Integration mit ALM - Quality Center

Dieser Testtyp ist für den Aufruf von automatisierten Testskripten gedacht. Eine Vorlage für ein VAPI-XP-TEST Skript, welches QF-Test an Quality Center anbindet, finden Sie in der Datei `qcVapiXPTemplate.txt` im QF-Test Installationsverzeichnis, siehe `qftest-9.0.0/ext/qualitycenter`. Diese Vorlage kann für alle QF-Test Tests, welche in Quality Center aufgerufen werden, benutzt werden. Eine detaillierte Schritt-für-Schritt Anleitung dazu finden Sie in Abschnitt 28.1.2(375).

Die QF-Test Vorlage für das VAPI-XP-TEST-Skript benutzt ein externes VBScript-Skript `qcTemplate.vbs`, welches die Ausführung der Tests übernimmt. Dieses Skript ist ebenfalls Bestandteil des Lieferumfangs von QF-Test (siehe `qftest-9.0.0/ext/qualitycenter`). Je nach Bedarf sollten Sie dieses Skript anpassen und in ein projektspezifisches Verzeichnis kopieren.

Der VAPI-XP-TEST in Quality Center startet QF-Test im Batchmodus lokal auf jedem einzelnen Testsystem. Dies bedeutet also, dass das externe Skript auf jedem Testsystem verfügbar bzw. erreichbar sein muss und entsprechend müssen auch die Testsuitedateien auf jedem Testsystem verfügbar sein, weshalb wir empfehlen, diese Dateien auf ein gemeinsames Netzlaufwerk zu legen bzw. in Ihr Versionsmanagementsystem aufnehmen.

Nach Ausführung des Tests wird das QF-Test Protokoll als Anhang zur Testinstanz hochgeladen und auch der Status des Tests entsprechend gesetzt.

Sie können das externe Skript auch dahingehend ändern, dass anstatt des Batchmodus ein Daemonaufruf (siehe Kapitel 55(1276)) stattfindet. In diesem Fall baut QF-Test die Verbindung zum Testsystem auf und nicht mehr Quality Center. Im Falle des normalen Batchaufrufes baut Quality Center die Netzwerkverbindung auf und ruft dann lokal QF-Test auf. Wenn Sie den Daemonaufruf benutzen wollen, dann muss sich das externe Skript auf dem Rechner, auf dem Quality Center installiert ist befinden, jedoch müssen

die Testsuitedateien weiterhin auf jedem einzelnen Testsystem erreichbar sein.

Falls Sie eine andere unterstützte Skriptsprache als VBScript, z.B. JScript, in Ihrem Projekt einsetzen sollten, können Sie die QF-Test Vorlagen natürlich in diese Skriptsprache portieren.

In der folgenden Abbildung sehen Sie das VAPI-XP-TEST-Skript in Quality Center:

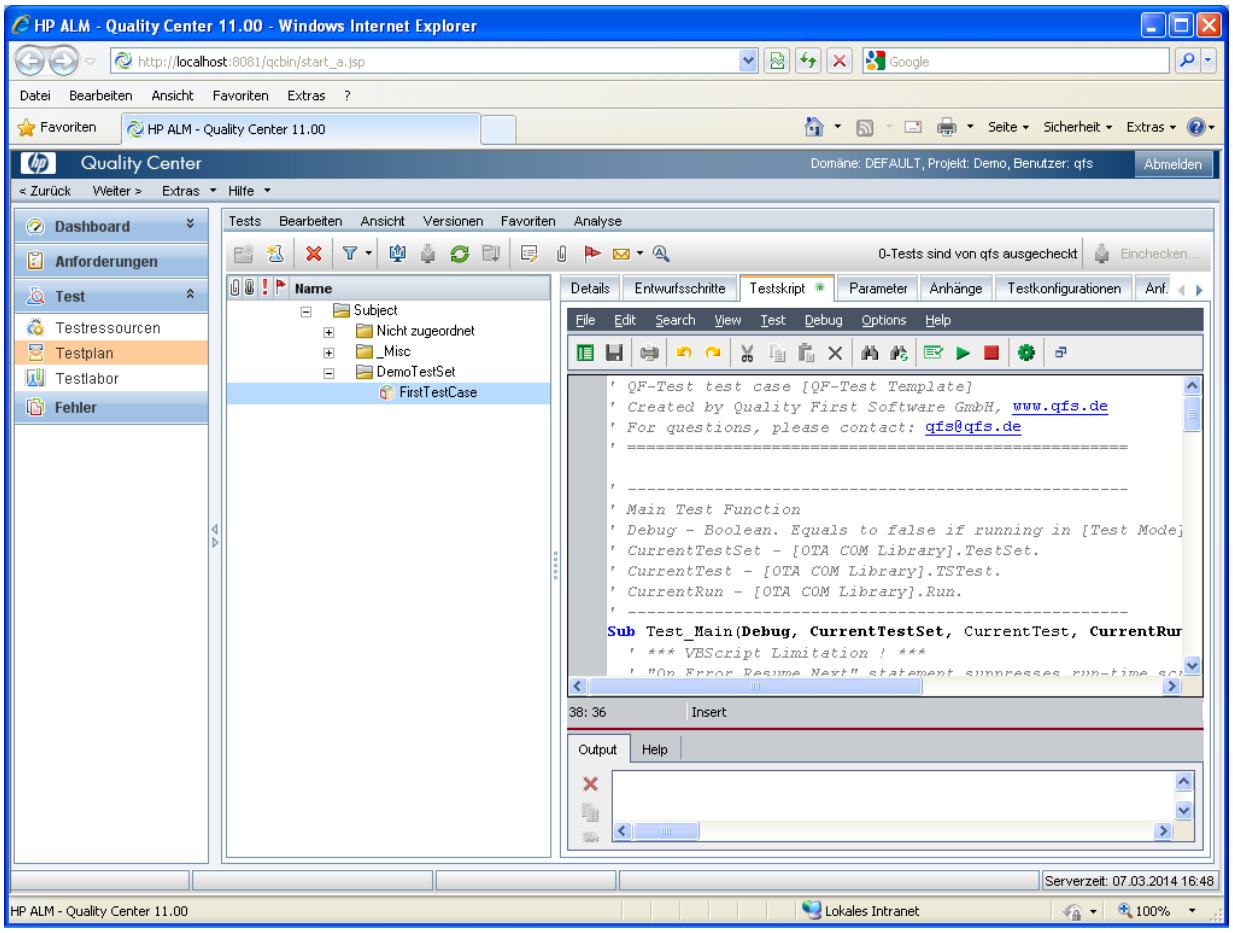


Abbildung 28.2: QF-Test VAPI-XP-TEST Testfall in HP ALM - Quality Center

28.1.2 Schritt für Schritt Anleitung

Allgemeine Schritte die auf dem Testsystem auszuführen sind:

1. Kopieren Sie das externe Skript `qftest-9.0.0/ext/qualitycenter/qcTemplate.vbs` in ein projektspezifisches Verzeichnis und ändern Sie dessen Namen. Wir empfehlen

Ihnen, auf jedem Testsystem denselben Pfad oder alternativ ein Netzlaufwerk zu verwenden.

2. Innerhalb des externen Skripts können Standardoptionen definiert werden, z.B. ob der Batch- oder Daemon-Modus zu Ausführung verwendet werden soll und wie der Name der Protokolldatei lauten soll. Dies kann aber auch zu einem späteren Zeitpunkt geschehen, was beim initialen Einrichten der Anbindung empfehlenswert sein kann, um die Dinge erst mal überschaubarer zu halten.

Schritte in ALM - Quality Center, um einen Beispieltest anzulegen:

1. Starten Sie ALM/Quality Center und loggen Sie sich in Ihr Projekt ein.
2. Im Bereich ”Testplan” können Sie als ersten Schritt einen neuen Testfallsatz z.B. ’DemoTestSet’ anlegen.

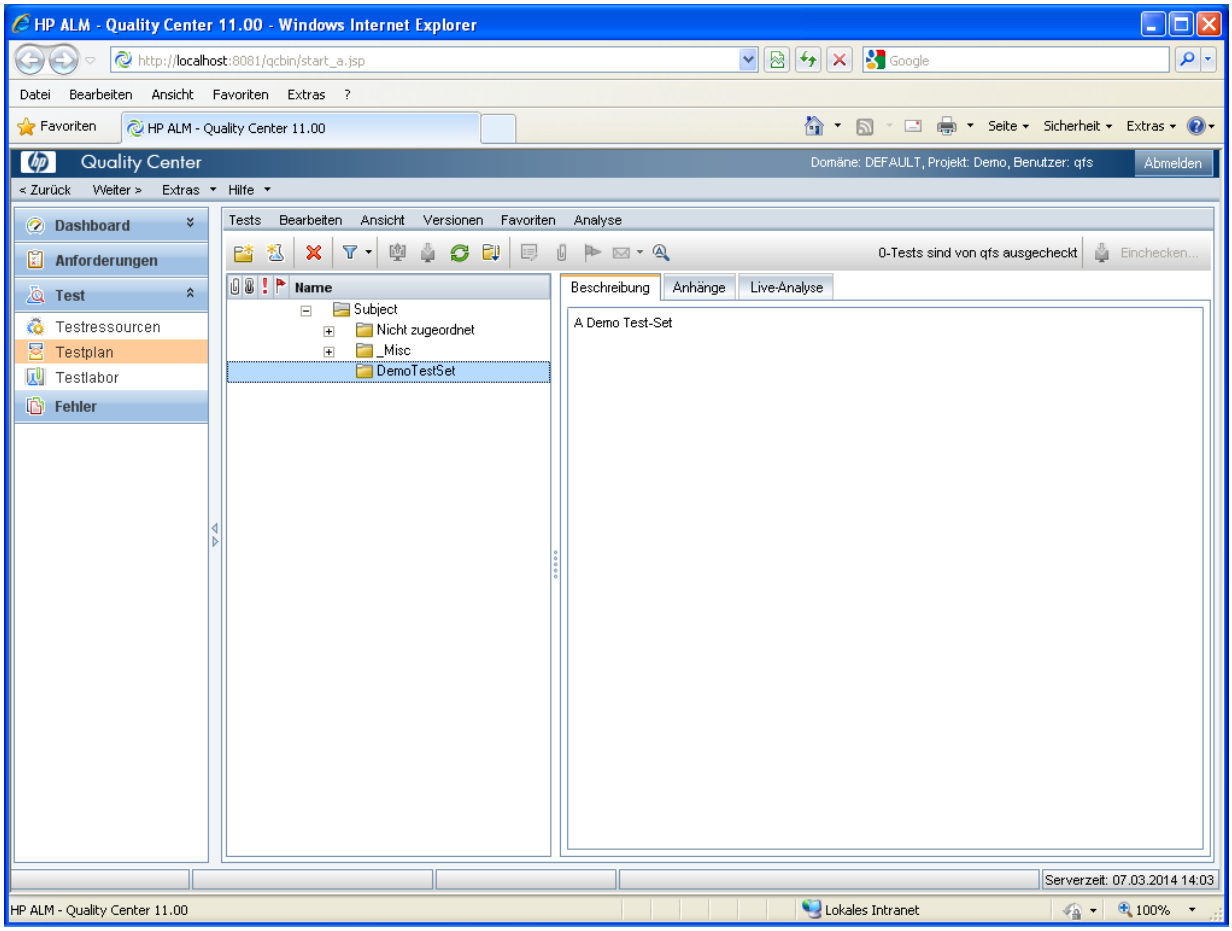


Abbildung 28.3: Im Testplan einen neuen Testfallsatz anlegen

3. In diesem Testfallsatz erstellen Sie einen neuen Test vom Typ VAPI-XP.

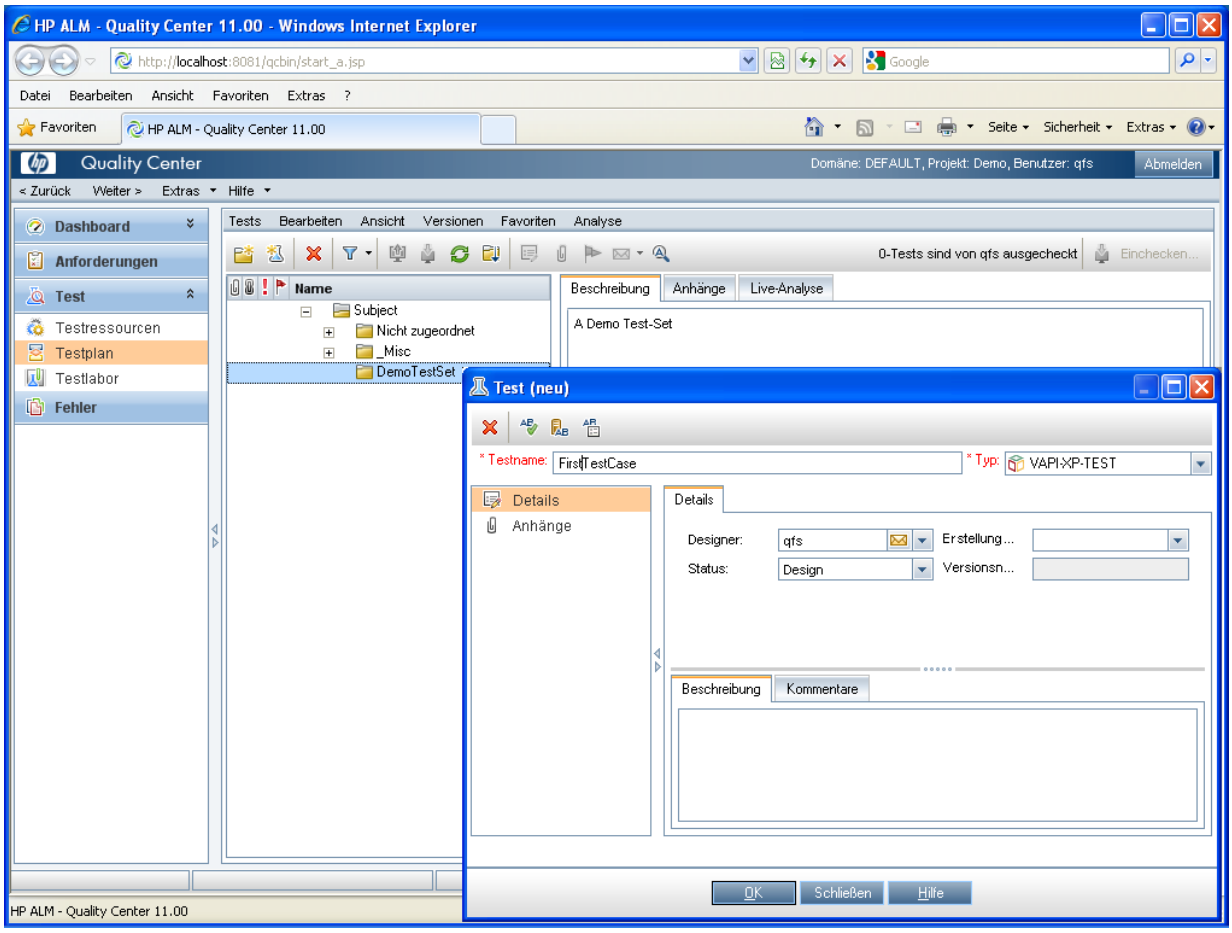


Abbildung 28.4: Test vom Typ VAPI-XP-TEST anlegen

Hinweis

4. Im HP VAPI-XP Wizard Fenster drücken Sie einfach ohne Änderungen auf Finish. (Das bedeutet Sie haben VBScript als Skriptsprache und COM/DCOM Server Test als Testtyp).

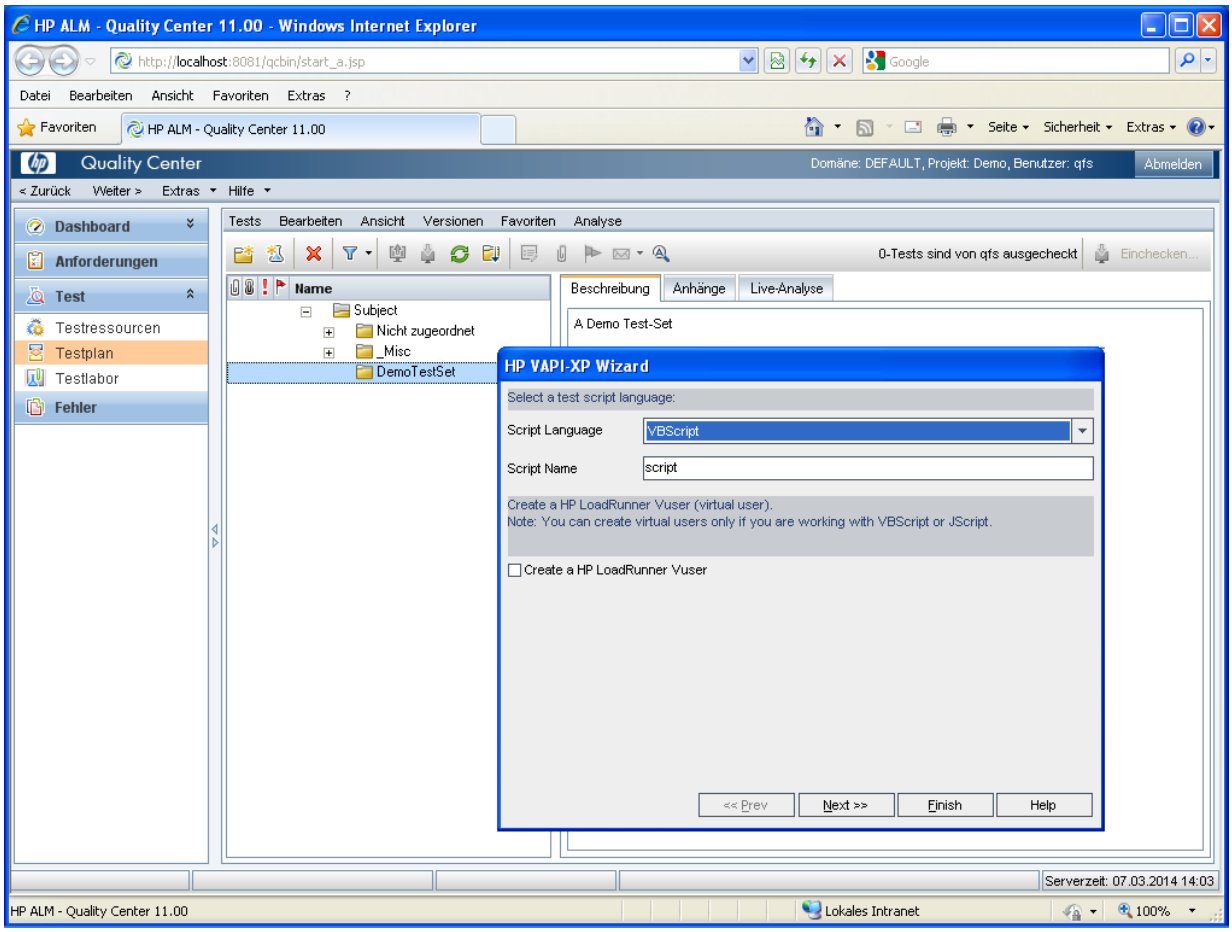


Abbildung 28.5: HP VAPI-XP Wizard

5. Der erhaltene Test sieht wie folgt aussieht.

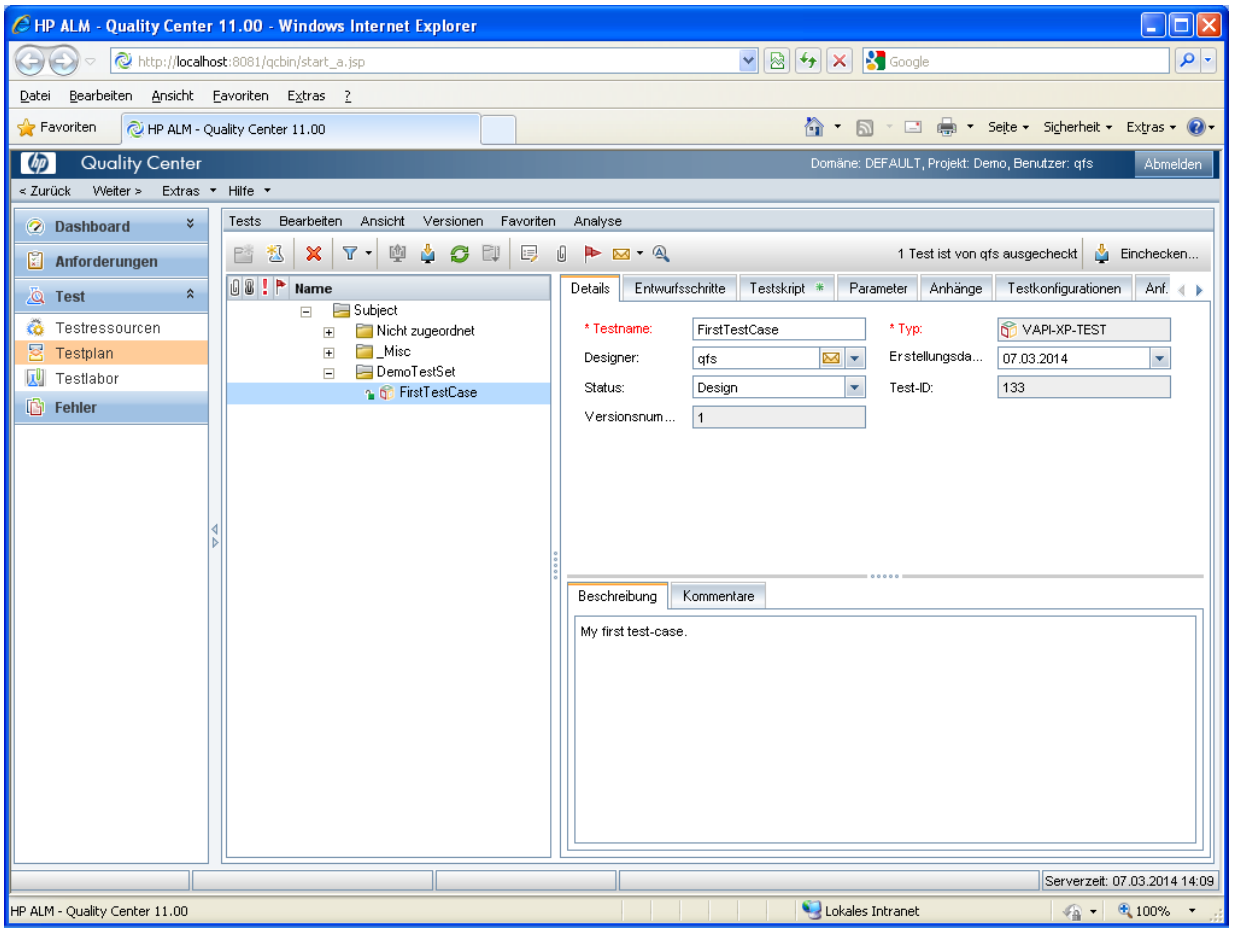


Abbildung 28.6: VAPI-XP-TEST Details

6. Wechseln Sie zum Reiter ”Testskript” und kopieren Sie den Inhalt der QF-Test Vorlage `qftest-9.0.0/ext/qualitycenter/qcVapiXPTemplate.txt` in den Textbereich des Skripteditors.

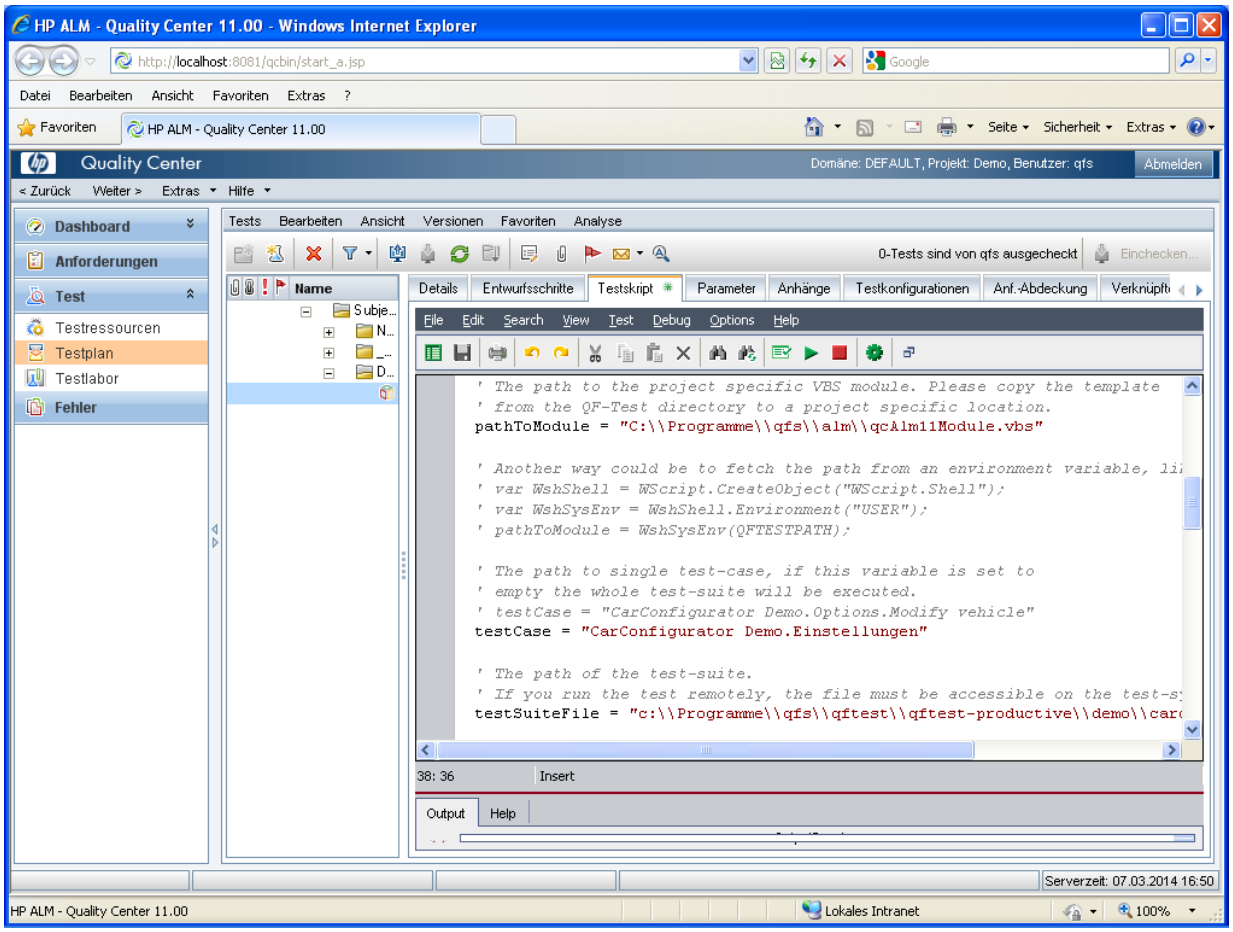


Abbildung 28.7: Template in Textbereich des Skripts kopieren

7. Im Testskript führen Sie bitte folgende Anpassungen durch:

   - Ändern Sie die `pathToModule` Variable auf den Pfad, in welchen Sie das externe Skript `qcTemplate.vbs` kopiert haben.
   - Passen Sie die Variable `testSuiteFile` entsprechend der zur verwendenden Testsuite an.
   - Wenn Sie einen einzelnen Testfall aus der Testsuitedatei ausführen wollen, passen Sie auch die Variable `testCase` entsprechend an.

Bitte beachten Sie auch die Kommentare in den QF-Test Vorlagen, in denen Sie Hinweise zu weiteren Einstellungsoptionen finden.

Schritte in ALM - Quality Center zum Ausführen des Beispieltests:

1. Wechseln Sie in den Bereich ”Testlabor” von ALM/Quality Center.

2. Vermutlich werden Sie auch hier einen Testfallsatz anlegen wollen.

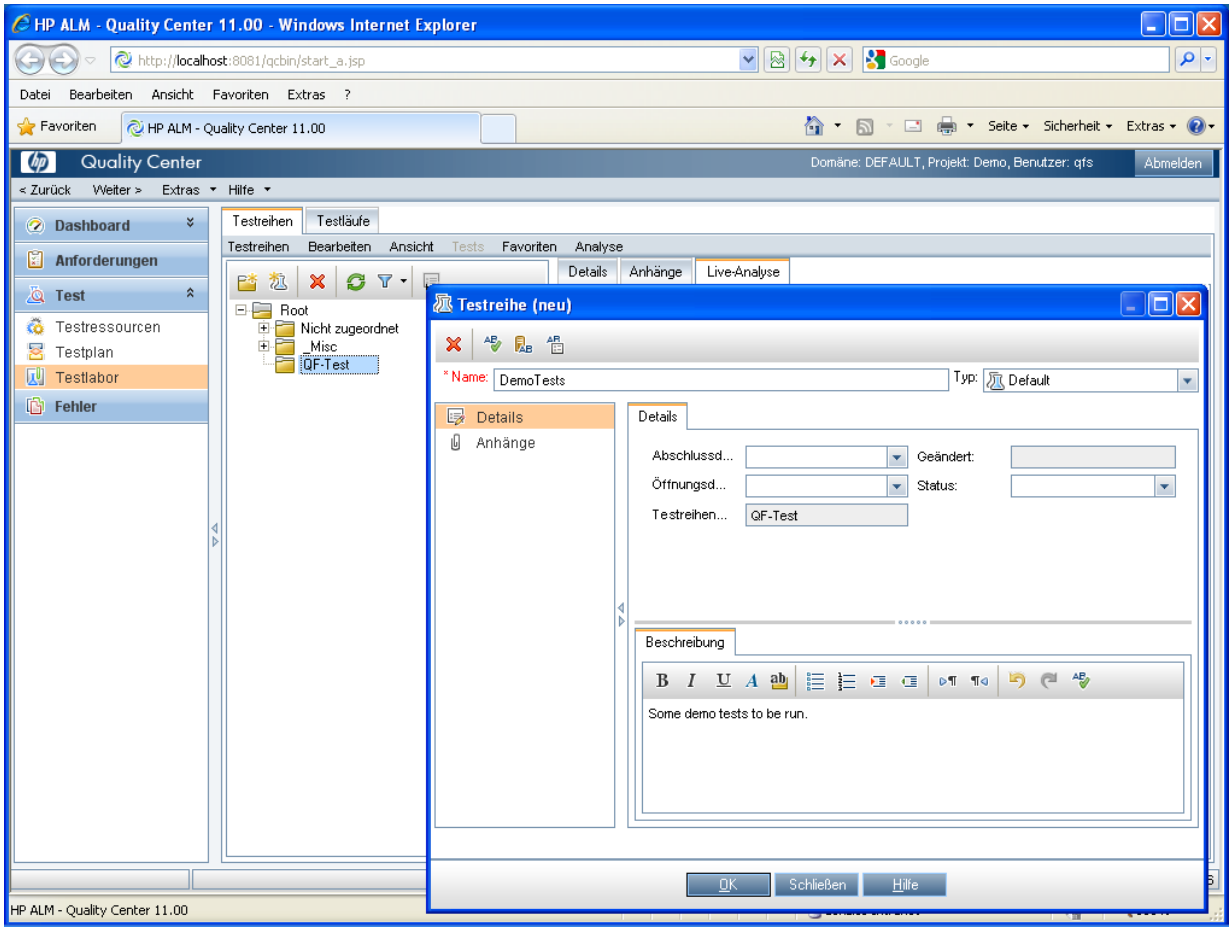


Abbildung 28.8: Neuer Testfallsatz im Testlabor

3. Fügen Sie dort den im vorigen Teil erstellten VAPI-XP-TEST durch passendes Auswählen zur Ausführung hinzu.

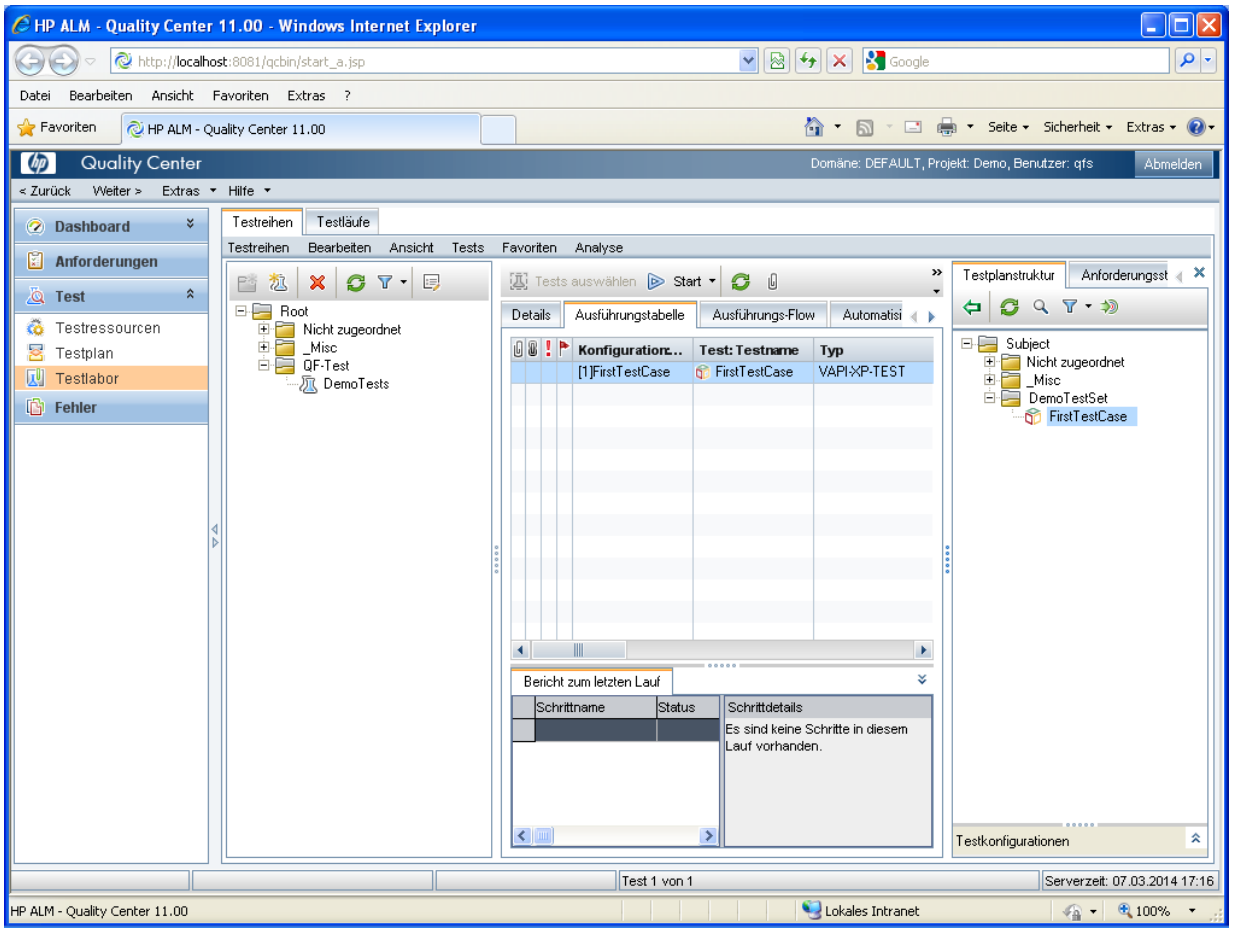


Abbildung 28.9: Test zu Ausführung hinzufügen

4. Nun können Sie den Test starten. Achten Sie darauf, dass Sie das Häkchen bei ”Alle Tests lokal ausführen” setzen, außer Sie haben bereits ein verteiltes System eingerichtet.

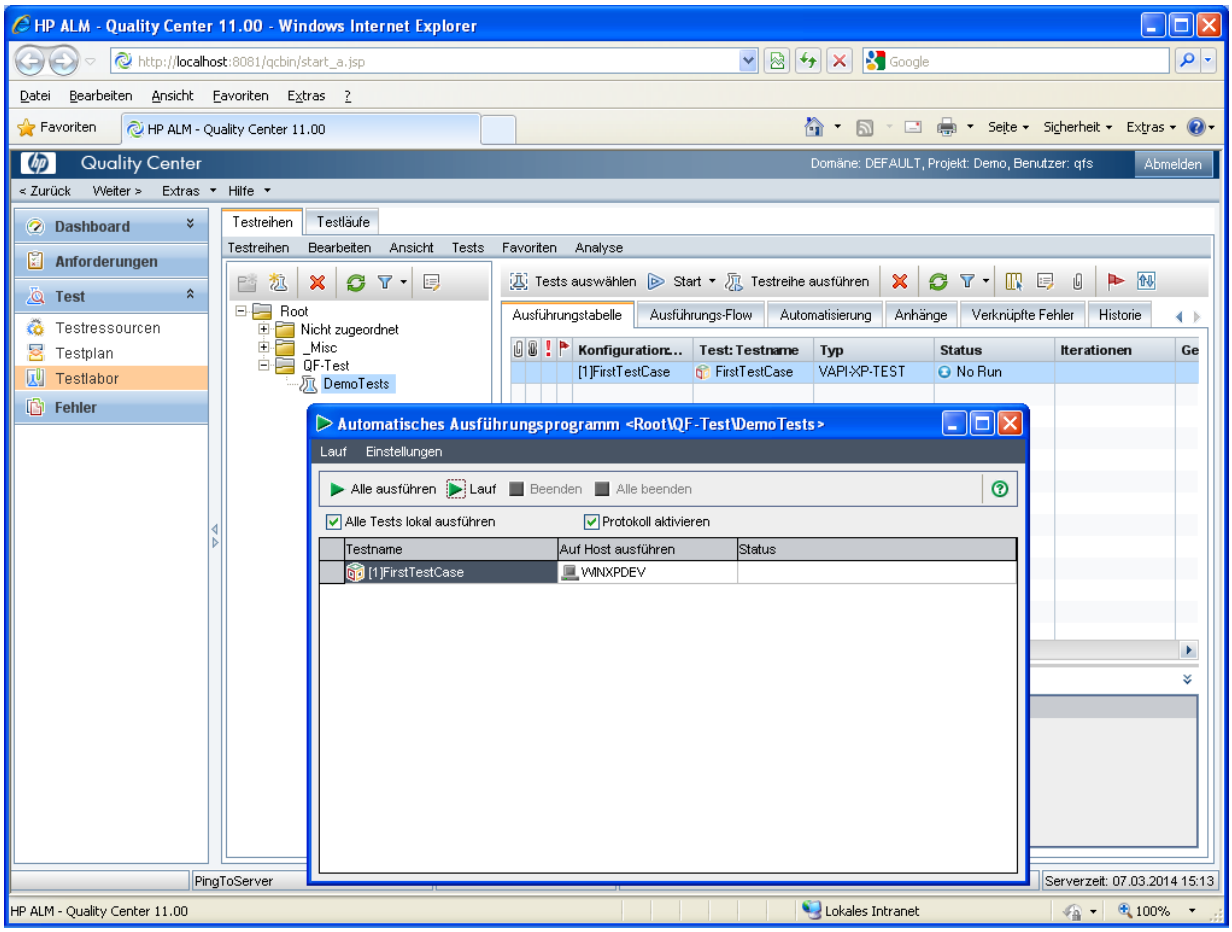


Abbildung 28.10: Ausführung des Tests

5. Quality Center sollte nun den Test ausführen - ggf. auf Ihrem Rechner, dann taucht das SUT auf, Aktionen werden durchgeführt und das SUT wird wieder beendet. Nach dem Beenden wird das Resultat für den Testlauf eingetragen: Passed oder Failed.

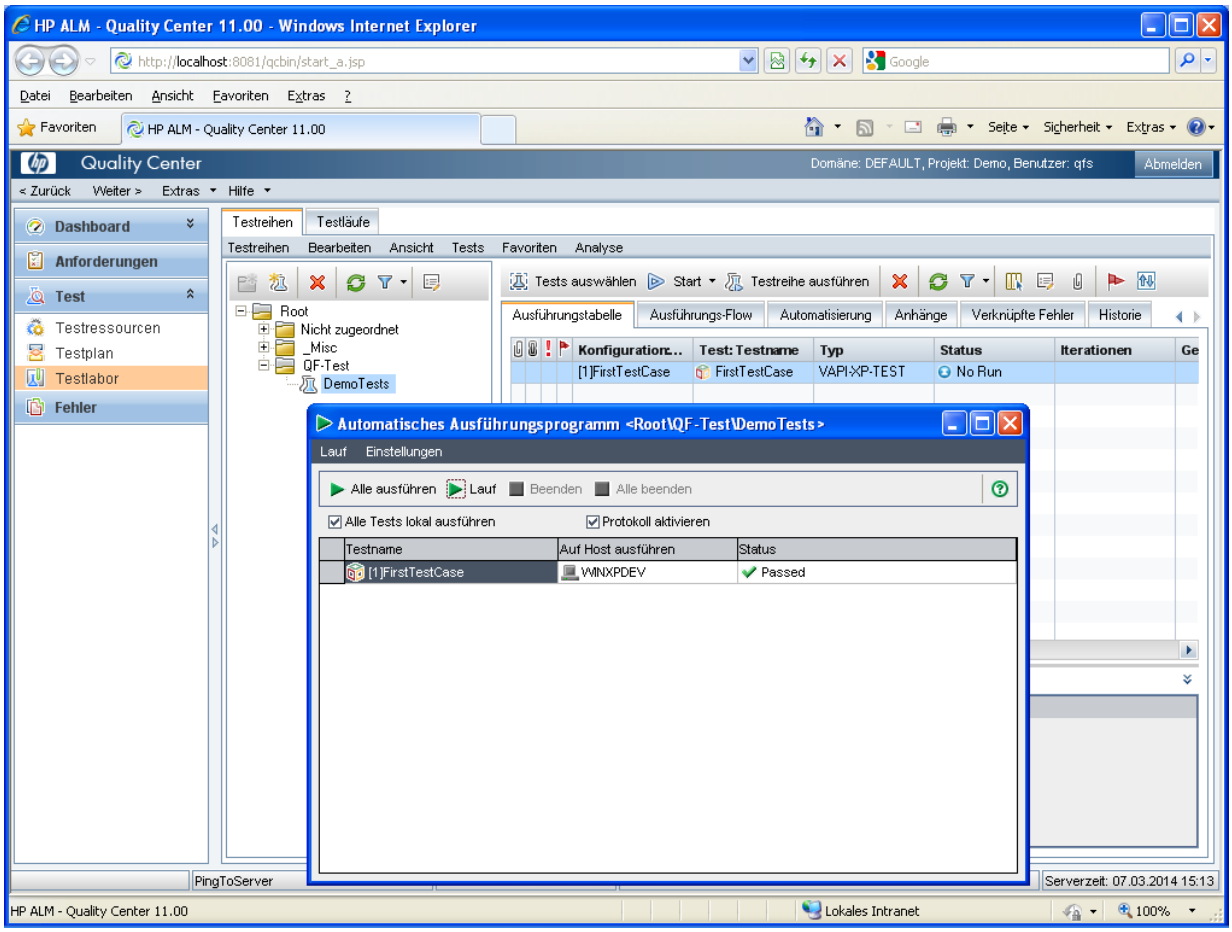


Abbildung 28.11: Testresultat

6. Nachdem der Test ausgeführt wurde, wird das Protokoll des Testlaufs als Anhang zur Testinstanz hochgeladen und der Status des Testfalles auf das entsprechende Resultat gesetzt.

7. Zusätzlich zum Ergebnis wird auch das QF-Test Protokoll in Quality Center hochgeladen und beim Testlauf abgelegt. Um das Protokoll anzusehen, führen Sie einen Doppelklick auf den Test im Ausführungsbereich aus, wechseln Sie dann auf ”Läufe” und klicken nochmals doppelt auf die Büroklammer (Anhangssymbol) des entsprechenden Testlaufs.

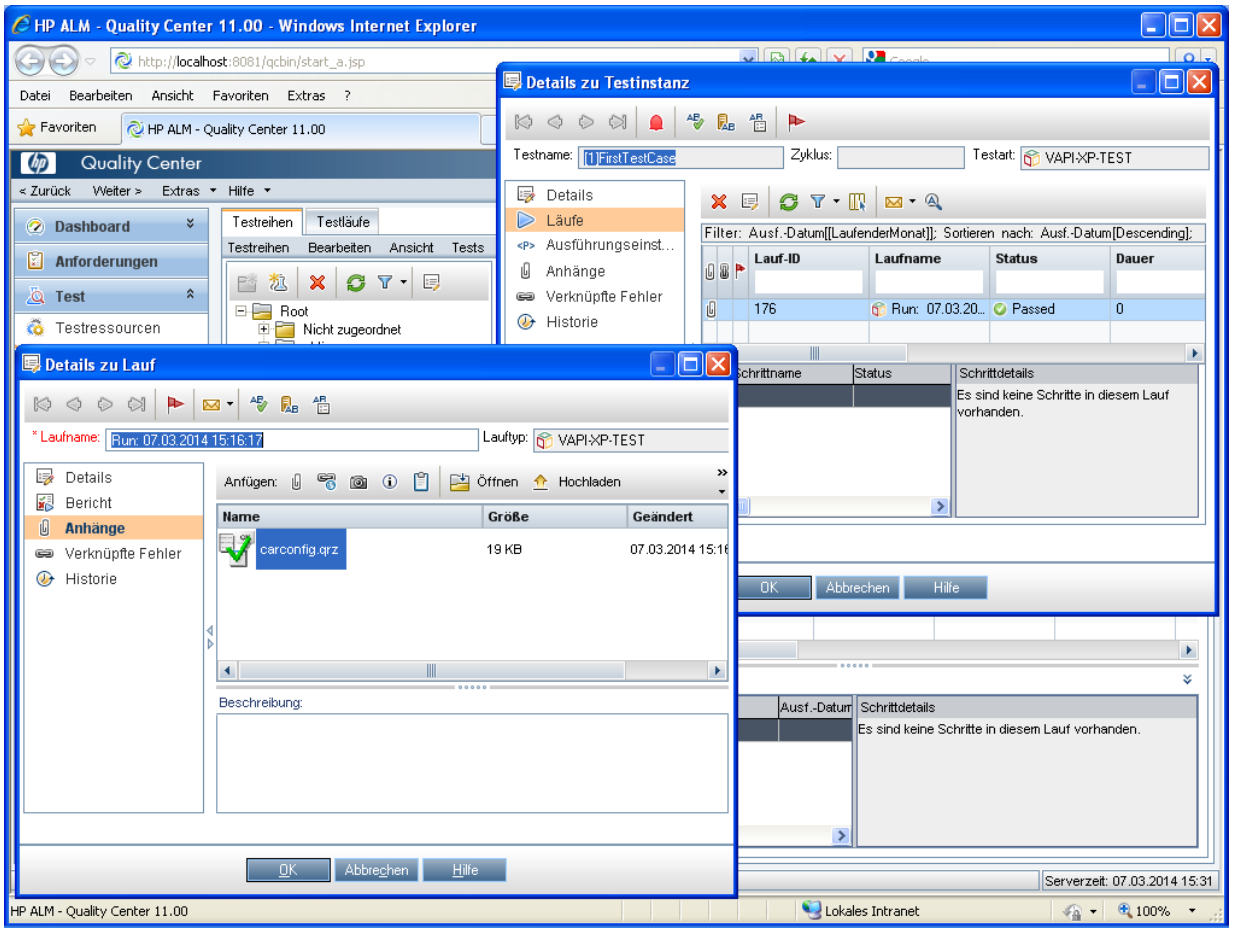


Abbildung 28.12: Hochgeladenes Protokoll

28.1.3 Fehlersuche

Als ersten müssen wir vorausschicken, dass wir selbst keine Quality Center Experten sind. Deshalb mag es weitere und bessere Wege zu Fehlersuche geben. Trotzdem wollen wir die Methoden, die wir selbst angewendet haben, hier beschreiben.

Leider ist die Prozessausgabe während der Testausführung nur für wenige Sekunden sichtbar und erlaubt somit keine direkte Analyse. Deshalb muss man eine Alternative suchen.

Der Texteditor des VAPI-XP-TEST-Testknotens im Bereich ”Testplan” erlaubt das direkte Ausführen des Skripts. Die Ausgabe ist dann permanent im Bereich unter dem Skript sichtbar und enthält hoffentlich etwas hilfreiches.

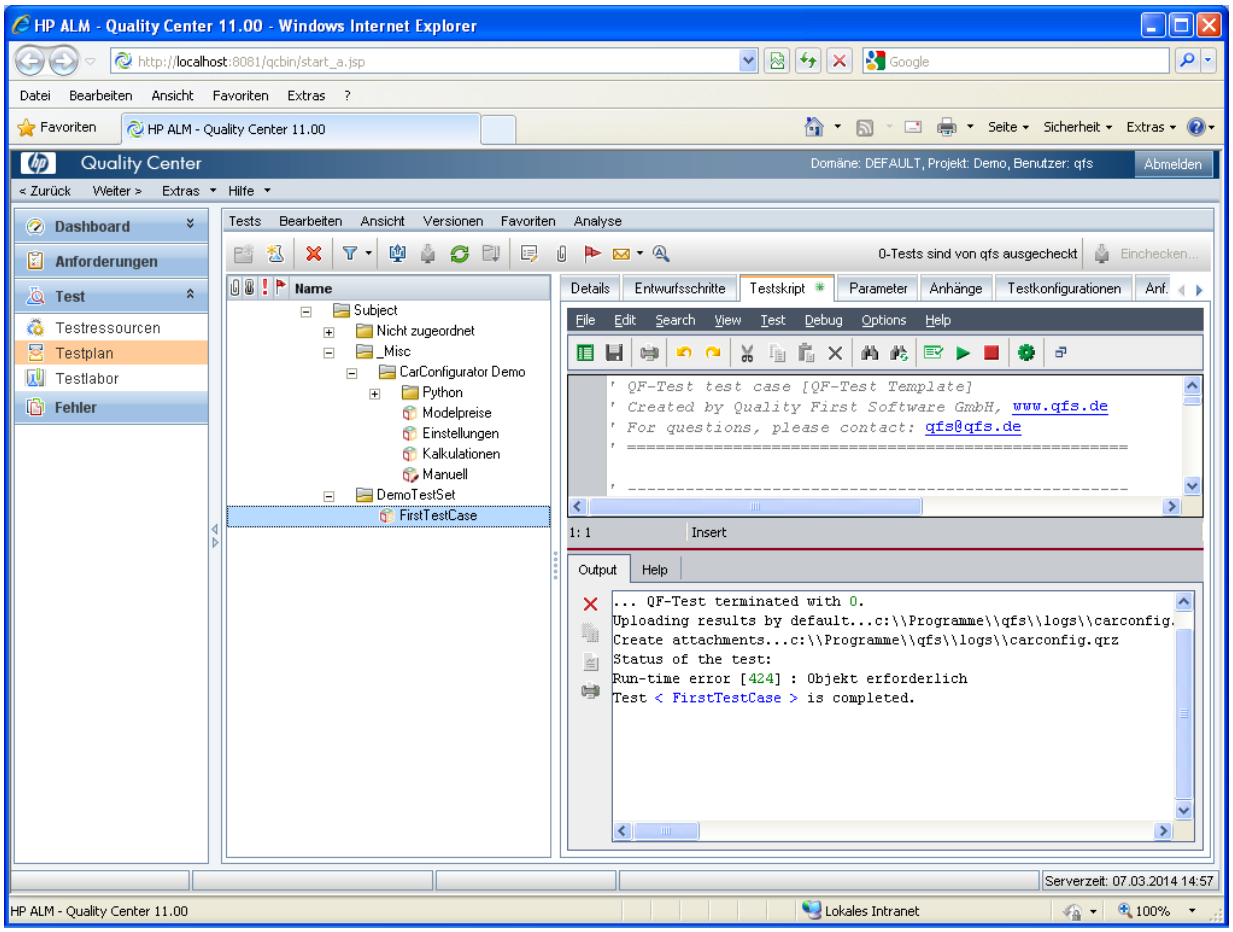


Abbildung 28.13: Skript Debuglauf

Die direkte Ausführung eines Skripts muss jedoch mit Vorsicht betrachtet werden. Es handelt sich ja nicht um einen echten Testlauf. Deshalb kann auch das Protokoll nicht hochgeladen werden und es wird ein entsprechender Run-time error ”Objekt erforderlich” ausgegeben. Davon sich also bitte nicht verwirren lassen!

Für weiteres Debugging kann man zusätzliche Ausdrücke der der Form `TDOutput.Print "Some text"` in die Skripte und auch das externe Skript einbauen. Mit diesen kann man sehen, wie weit die Ausführung des Skripts funktioniert und wo ggf. ein Fehler auftritt.

Der Skripteditor hat auch einen ”Syntax check” Button, der hilfreich zur Prüfung nach jeder Änderung eingesetzt werden kann.

28.2 Imbus TestBench

28.2.1 Einführung

Die aktuelle Integration von QF-Test und der TestBench der Firma Imbus besteht aus zwei Teilen:

- Generieren von QF-Test Vorlagen-Testsuiten aus den TestBench Interaktionen.
- Importieren der QF-Test Resultate in die TestBench.

Sie finden alle benötigten Testsuiten sowie Demoimplementierungen im Ordner `qftest-9.0.0/ext/testbench/Version_1.1_TestBench_2.3`. Bitte achten Sie darauf, dass Sie alle Testsuiten vor dem eigentlichen Einsatz in einen projektspezifischen Ordner kopieren und diese dort modifizieren.

Das folgende Kapitel gibt einen Überblick über das Konzept der Integration.

28.2.2 Generieren von QF-Test Vorlage-Testsuiten aus den Interaktionen

Nachdem Sie den Testfall in der TestBench geplant und die entsprechenden Interaktionen angelegt haben, können Sie mittels des QF-Test Export Plug-ins eine QF-Test Datei generieren. Imbus liefert die erforderlichen Informationen, wie Sie dieses Plugin einspielen müssen.

In dieser QF-Test Datei finden Sie alle Interaktionen als Prozeduren. Ihre Struktur wird durch Packages abgebildet. Nun können Sie mit der Aufnahme der einzelnen Interaktionen beginnen und die Prozeduren mit Leben füllen.

Nach Fertigstellung der Aufnahme speichern Sie diese Testsuite in einem projektspezifischen Ordner ab. Diese Datei muss später bei der Ausführung in der Ausführungstestsuite inkludiert werden, um so die Interaktionen ausführen zu können. Wir empfehlen diese Datei entweder in das Versionsmanagementsystem mitaufzunehmen oder auf ein Netzlaufwerk zu legen.

28.2.3 Importieren der Resultate

Für die Ausführung der Tests benötigen Sie spezielle Testfälle und Prozeduren, welche Sie in den mitgelieferten Testsuiten finden können. Diese befinden sich im Ordner `qftest-9.0.0/ext/testbench/Version_1.1_TestBench_2.3/suite`. Bitte

achten Sie darauf, dass Sie alle Testsuiten vor dem eigentlichen Einsatz in einen projektspezifischen Ordner kopieren und diese dort modifizieren.

Sie finden im Ordner `qftest-9.0.0/ext/testbench/Version_1.1_TestBench_2.3/suite/demo` eine Beispielimplementierung für den CarConfigurator. Hier ist es wichtig, dass Sie eine Testsuite benötigen, die die Testsuite `TestBench_Automation.qft` inkludiert. Falls Sie Ihre Prozeduren über den iTEP Export erstellt haben (siehe auch Abschnitt 28.2.2(387)), sollten sie auch diese Testsuite inkludieren.

Nun sollten sie noch die Konfiguration der Ausgabedateien in den Dateien `testaut.properties` und `user.properties` vornehmen.

Jetzt können Sie den Testfall `Standalone test executor` aus `TestBench_Automation.qft` aufrufen.

Nach der Testausführung können Sie die gesamten Ergebnisse in die TestBench importieren. Verwenden Sie dazu das iTEP bzw. das iTORX Import Plug-in. Die einzelnen QF-Test Protokolle werden zusätzlich an die Testinstanzen angehängt.

28.3 QMetry

28.3.1 Einführung

Die aktuelle Integration zwischen QF-Test und QMetry beinhaltet die Unterstützung der Planung von Testfällen bzw. Testschritten innerhalb von QMetry und anschließend die Weiterleitung der Ausführungsinformationen an QF-Test. Nachdem QF-Test die Tests ausgeführt hat, werden das Protokoll und der HTML-Report in den Testergebnisbereich von QMetry hochgeladen sowie der Status des Testfalles auf das entsprechende Ergebnis gesetzt.

Für die Testausführung müssen Sie Ihr Testsystem wie folgt vorbereiten:

- Im 'Admin'-Bereich der QMetry-Testmanagement-Ansicht, müssen Sie einen Ausführungsagenten in den 'Agent'-Einstellungen einrichten.
- Laden Sie die entsprechenden Agentendateien sowie die Konfigurationsdateien auf Ihr Testsystem.
- Installieren Sie den QF-Test QMetry Launcher auf Ihrem Testsystem.
- Installieren Sie eine Testplattform in den 'Platform'-Einstellungen, der ebenfalls im 'Admin'-Bereich zu finden ist.
- Setzen Sie in der Datei `QMetryAgent.properties` die benötigten Umgebungsvariablen für den QF-Test Wrapper von QMetry.

- Setzen Sie in der Datei `QMetryConfig.properties` den korrekten und vollständigen Pfad zur QF-Test Ausführungsdatei.
- Setzen Sie in der Datei `QMetryConfig.properties` ggf. erwünschte Ausführungsparameter, siehe auch nächsten Abschnitt.
- Starten Sie den QMetry Agenten. Unter Windows sollten Sie den Agenten nicht als Dienst / Service starten, damit das SUT nicht in der Service-Session läuft. Falls Sie den Agenten doch als Service starten möchte, dann müssen Sie die QF-Test Tests mittels QF-Test Daemon ausführen, welcher wiederum nicht als Dienst gestartet werden darf.

Nachdem Sie den QMetry Agenten und Launcher konfiguriert haben, müssen Sie die Testausführung planen. Für die Testausführung von QF-Test Testfällen gibt es mehrere Varianten in QMetry. Sämtliche Varianten sind im QMetry IntegrationGuide für QF-Test beschrieben. Für die schnellste Variante gehen Sie wie folgt vor:

- In der Testmanagement Ansicht wechseln Sie in den 'Test Cases' Bereich und erstellen die Testfälle dort.
- Bei jedem Testfall müssen Sie im Attribut 'Test Script Name' den vollständigen Pfad zur benötigten QF-Test Testsuite setzen.
- Der Name des Testfalles sollte genau dem vergebenen QF-Test ID Attribut in QF-Test entsprechen.
- Fügen Sie den Testfall zu einer ausführbaren Testsuite in der 'Test Suites' Ansicht hinzu.

Nun können Sie die Testfälle ausführen:

- Öffnen Sie die 'Test Suites' Ansicht und selektieren die auszuführende Testsuite.
- Wählen Sie den 'Execute TestSuite' Reiter.
- Starten Sie direkt oder planen Sie die Ausführung eines Testlaufes mittels Klick in die 'Automation' Spalte. Wählen Sie dort auch den entsprechenden Agenten aus.
- Wenn der QMetry Agent das nächste Mal die Ausführungsinformationen vom QMetry Server abholt, wird die Ausführungsinformation des Testfalles an den QMetry Agenten gesandt.
- Nach Beendigung des Testlauf wird das QF-Test Protokoll und der HTML-Report als Resultat im Reiter 'Execution History' der ausgeführten Testsuite in QMetry hochgeladen und der entsprechende Status des Testlaufes gesetzt.

Die unten stehende Abbildung zeigt den 'Execution History' Reiter in der 'Test Suites' Ansicht inkl. QF-Test Protokoll:

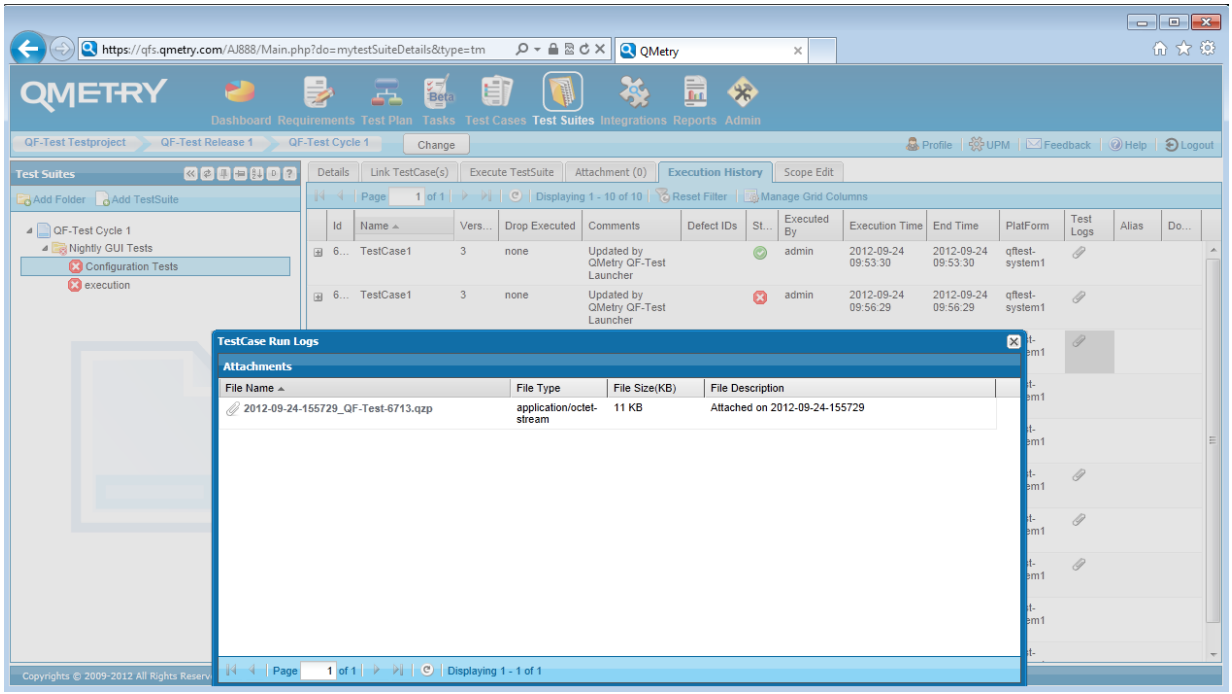


Abbildung 28.14: QF-Test Protokoll in QMetry

Sie finden detaillierte Informationen sowie weitere Varianten für die QF-Test Integration im Handbuch und IntegrationGuide Dokument von QMetry.

28.3.2 Demokonfiguration Beispiel

Für eine einfache Integration empfehlen sich folgende Einstellungen in der Datei `QMetryConfig.properties`:

- Setzen Sie den Wert von `generic.adapter.success.code` auf `0,1`.
- Setzen Sie `qftest.additional.arguments` auf `-test ${QMTestCaseName}` im Falle einer lokalen Ausführung.
- Im Falle einer Ausführung per QF-Test Daemon, setzen Sie `qftest.additional.arguments` auf `-test ${QMTestCaseName} -calldaemon -daemonhost <Ihr-Testrechner> -daemonport <Ihr-DaemonPort> .`

Wie bereits im vorigen Kapitel erwähnt, muss bei dieser Integrationsvariante der Name des Testfalles in QMetry den Wert des QF-Test ID Attributes des Testfall Knotens in QF-Test entsprechen.

Weitere Varianten der Integration zwischen QMetry und QF-Test finden Sie im Integrationguide Dokument von QMetry.

28.4 Klaros

28.4.1 Einführung

Klaros ist ein Testmanagementtool, das von der Firma verit Informationssysteme GmbH, Kaiserslautern, Deutschland entwickelt und vertrieben wird.

Klaros ist in zwei Editionen erhältlich, einer freien Community-Edition und einer Enterprise-Edition mit einen erweiterten Funktionsumfang, individuellen Konfigurationsmöglichkeiten und umfassenden Support.

Die aktuelle Integration von QF-Test mit Klaros umfasst:

- Import von QF-Test Ergebnissen in Klaros.

28.4.2 Importieren von QF-Test Ergebnissen in Klaros

Nach dem Erzeugen eines XML-Reports, wie in Kapitel 24(330) beschrieben, kann dieser in Klaros importiert werden. Ein Beispiel für eine QF-Test Import-URL könnte wie folgt aussehen, wobei die Ergebnisdatei im HTTP-Request Body enthalten ist.

```
http://localhost:18080/klaros-web/seam/resource/rest/importer?
                config=P00001&env=ENV00001&sut=SUT00001&type=qftest&
                time=01.03.2011_12:00&username=me&password=secret
```

Beispiel 28.1: Importieren von Testergebnissen in Klaros

Das `curl`-Kommandozeilenprogramm kann auf Linux und Windows/Cygwin den Import direkt anstoßen.

```
curl -v -H "Content-Type: text/xml" -T "my_qftest_report.xml" \
"http://localhost:18080/klaros-web/seam/resource/rest/importer\
?config=P00001&env=ENV00001&sut=SUT00001&type=qftest\
&time=05.02.2013_12:00&user=me&password=secret"
```

Beispiel 28.2: Importieren von Testergebnissen in Klaros mit Hilfe des `curl`-Kommandos

Weitere Informationen finden Sie im Klaros Online-Handbuch unter https://www.klaros-testmanagement.com/files/doc/html/User-Manual.Import-Export.html.

28.5 TestLink

28.5.1 Einführung

Die aktuelle Integration von QF-Test und dem Opensource Tool TestLink besteht aus zwei Teilen:

- Generieren von QF-Test Vorlagen-Testsuiten aus den geplanten Testfällen in TestLink.
- Importieren der QF-Test Resultate als TestLink Resultate.

3.5.1+ Seit TestLink 1.9.4 kann die TestLink API für die Integration verwendet werden. Diese Art der Integration benötigt einen gültigen Development-Schlüssel. Dieser kann in den Einstellungen von TestLink unter 'Meine Einstellungen' im Bereich 'API Schnittstelle' erzeugt werden. Hierfür klicken Sie einfach auf den Button 'Neuen Schlüssel erzeugen'.

Für ältere TestLink Versionen bis 1.9.3 wird für die Integration direkt auf die Datenbank von TestLink zugegriffen. Diese Technik erfordert einen JDBC Datenbanktreiber, damit die mitgelieferten Skripte verwendet werden können. Die Datenbanktreiber finden Sie auf den Internetseiten der jeweiligen Datenbankhersteller und können diese dort auch herunterladen.

Das Exportieren der geplanten Testfälle, mitsamt Testschritten, aus TestLink unterstützt den Testautomatisierer den Testfall wie geplant zu implementieren.

Das Importieren der Testresultate nach TestLink erlaubt eine bessere Übersicht über die ausgeführten automatischen und manuellen Tests mit einem Werkzeug.

Hinweis Testresultate können Sie auch unabhängig vom Export von QF-Test in TestLink importieren, hierfür sollten Sie allerdings beachten, dass im Namen der Testfälle in QF-Test immer die ID des Testfalles von Testlink im Namen steht. Ein Testfall sollte nach folgendem Schema benannt werden: `<TestLink-ID>: Name des Testfalles`.

28.5.2 Generieren von QF-Test Vorlagen-Testsuiten aus den Testfällen

Um eine einheitliche Struktur von automatisierten Testfällen in QF-Test und der Testplanung in TestLink sicher zu stellen, bietet QF-Test die Möglichkeit, Testsuite Dateien mit derselben Struktur wie innerhalb eines Testprojektes von TestLink zu generieren.

In dieser QF-Test Datei finden Sie einen Testfall Knoten pro Testfall und einen Testfallsatz Knoten pro Testsuite aus TestLink. Falls Sie die Felder ”Auszuführende Schritte” und ”Erwartetes Resultat” eines Testfalles ausgefüllt haben, wird auch ein leerer Testschritt pro auszuführenden Schritt in den jeweiligen Testfällen angelegt. Das erwartete Resultat finden Sie im Bemerkung Attribut des Testschritt Knotens.

Diese Vorlagen-Testsuite muss der Testautomatisierer nun mit entsprechenden Leben füllen, indem er die erstellten Testschritte mit den benötigten Prozeduraufrufen und Aufzeichnungen füllt.

3.5.1+ Falls Sie mit TestLink 1.9.4 oder einer neueren Version arbeiten, führen Sie bitte folgende Schritte durch:

1. Stellen Sie sicher, dass Testautomatisierung in TestLink eingeschaltet ist. Dafür setzen Sie in der Konfigurationsdatei `config.inc.php` den Wert `enable_test_automation` auf `ENABLED`.
2. Kopieren Sie das Verzeichnis `qftest-9.0.0/ext/testlink/api` in einen projektspezifischen Ordner.
3. Öffnen Sie dort das Startskript, welches Sie verwenden möchten, mit einem Texteditor, d.h. `exportTests.bat` für Windows bzw. `exportTests.sh` für Linux.
4. Passen Sie dort die Pfade für die Variablen `JAVA`, `QFTDIR` und `TESTLINKINTEGRATOR` an.
5. Öffnen Sie die Datei `TestLinkUserSpecifics.py` mit einem Texteditor.
6. Passen Sie die Variablen `serverurl` und `devkey` entsprechend an.
7. Falls Sie benutzerdefinierte Felder aus TestLink in den Export mit einbeziehen wollen, passen Sie auch die Variable `custom_fields` an.
8. Rufen Sie nun das angepasste Exportskript, wie im Beispiel unten, auf.

```
exportTests.bat --testproject projectname
                --targetsuite /path/to/testsuite.qft
```

Beispiel 28.3: Beispielaufruf für das Exportieren von Testfällen ab 1.9.4

Falls Sie Testfälle für eine Version bis TestLink 1.9.3 exportieren möchten, führen Sie bitte folgende Schritte durch:

1. Kopieren Sie das Verzeichnis `qftest-9.0.0/ext/testlink/export` in einen projektspezifischen Ordner.
2. Öffnen Sie dort das Startskript, welches Sie verwenden möchten, mit einem Texteditor, d.h. `exportTestLinkToQFT.bat` für Windows bzw. `exportTestLinkToQFT.sh` für Linux.
3. Passen Sie dort die Pfade für die Variablen `JAVA`, `QFTDIR` und `TESTLINKINTEGRATOR` an.
4. Öffnen Sie nun die Datei `TestLinkDBIntegrator.py` mit einem Texteditor.
5. Passen Sie dort die Variablen `dbdriver`, `conncetionstr`, `dbuser` und `dbpass` entsprechend Ihrer Datenbank an.
6. Falls Sie benutzerdefinierte Felder aus TestLink in den Export mit einbeziehen wollen, passen Sie auch die Variable `custom_fields` an.
7. Rufen Sie nun das angepasste Exportskript, wie im Beispiel unten, auf.

```
exportTestLinkToQFT.bat --testproject projectname
                --targetsuite /path/to/testsuite.qft
```

Beispiel 28.4: Beispielaufruf für das Exportieren von Testfällen bis 1.9.3

28.5.3 Ausführung der Testfälle

Die Ausführung der Testfälle kann wie gewohnt stattfinden. Sie sollten allerdings einen XML-Report erstellen, wenn Sie die Resultate nach TestLink laden wollen. Wichtig ist hierfür der Parameter '-report.xml' bei der Testausführung. Beim Erstellen des Reports über das GUI sollte die Checkbox 'XML Report erstellen' angehakt sein.

Hinweis

Falls Sie Ihre Testfälle nicht aus TestLink exportiert haben, sollten Sie die TestLink-ID in die Namen der QF-Test Testfälle mitaufnehmen. Ein Testfall sollte nach folgendem Schema benannt werden: `<TestLink-ID>: Name des Testfalles`.

```
qftest -batch -report.xml reportFolder testsuite.qft
```

Beispiel 28.5: Beispielausführung um XML-Reports zu erzeugen

28.5.4 Importieren der QF-Test Resultate nach TestLink

Nach Erstellen der XML-Reportdatei können Sie diese Resultate nun nach TestLink hochladen.

Standardmäßig wird für jede Ausführung ein neuer Build in TestLink angelegt. Die TestLink-Buildnummer wird aus der Run-ID des Reports gebildet. Diese können Sie bei Bedarf mit der Option '-runid' bei der Testausführung überschreiben. Allerdings können Sie den Buildnamen auch mit der Option '-build' beim Importieren der Resultate setzen.

3.5.1+ Falls Sie TestLink 1.9.4 oder neuer verwenden, führen Sie bitte folgende Schritte durch:

1. Stellen Sie sicher, dass Testautomatisierung in TestLink eingeschaltet ist. Dafür setzen Sie in der Konfigurationsdatei `config.inc.php` den Wert `enable_test_automation` auf `ENABLED`.
2. Kopieren Sie das Verzeichnis `qftest-9.0.0/ext/testlink/api` in einen projektspezifischen Ordner.
3. Öffnen Sie das Startskript, welches Sie verwenden möchten, mit einem Texteditor, d.h. `importResults.bat` für Windows bzw. `importResults.sh` für Linux.
4. Passen Sie dort die Pfade für die Variablen `JAVA`, `QFTDIR` und `TESTLINKINTEGRATOR` an.
5. Öffnen Sie die Datei `TestLinkUserSpecifics.py` mit einem Texteditor.
6. Passen Sie die Variablen `serverurl` und `devkey` entsprechend an. (Falls Sie dies bereits für den Export gemacht haben, können Sie dieselben Werte verwenden.)
7. Führen Sie das angepasste Importskript, wie im unten stehenden Beispiel, aus.

```
importResults.bat --testproject projectname
                --resultfile qftestReport.xml --testplan testplanname
            --platform system1
```

Beispiel 28.6: Importieren der Resultate ab 1.9.4

Falls Sie einen Build gezielt überschreiben wollen, können Sie dies mit dem `-build` Parameter erreichen.

```
importResults.bat --testproject projectname
                --resultfile qftestReport.xml --testplan testplanname
            --platform system1 --build myBuild
```

Beispiel 28.7: Importieren der Resultate ab 1.9.4 mit eigenem Build

Falls Sie mit TestLink 1.9.3 oder einer älteren Version arbeiten, führen Sie bitte folgende Schritte durch:

1. Kopieren Sie das Verzeichnis `qftest-9.0.0/ext/testlink/import` in einen projektspezifischen Ordner.
2. Öffnen Sie das Startskript, welches Sie verwenden möchten, mit einem Texteditor, d.h. `importToTestLink.bat` für Windows bzw. `importToTestLink.sh` für Linux.
3. Passen Sie dort die Pfade für die Variablen `JAVA`, `QFTDIR` und `TESTLINKINTEGRATOR` an.
4. Öffnen Sie nun die Datei `ReportParser.py` mit einem Texteditor.
5. Passen Sie dort die Variablen `dbdriver`, `conncetionstr`, `dbuser` und `dbpass` entsprechend Ihrer Datenbank an.
6. Erstellen Sie bei der Ausführung der QF-Test Tests einen XML-Report mit dem Parameter '-report.xml' und wahlweise auch mit einer eigenen Run-ID mit der Option '-runid'.
7. Führen Sie das angepasste Importskript, wie im unten stehenden Beispiel, aus.

```
importToTestLink.bat --testproject projectname
                --resultfile qftestReport.xml --testplan testplanname
                --tester tester
```

Beispiel 28.8: Importieren der Resultate bis 1.9.3

Kapitel 29

Integration mit Entwickler-Tools

Automatisierte GUI-Tests sind nur ein Schritt im Erstellungsprozess eines Software-Entwicklungsprojektes. Die grundlegende Anforderung, den Ablauf aller erforderlichen Schritte (Kompilierung, Testen, Erzeugung von Dokumentation, Auslieferung) zu automatisieren, hat zu einer Vielzahl von Entwickler-Tools (z.B. `Eclipse`) und Build-Tools (z.B. `make`, `ant`, `maven`) und so genannten Continuous-Integration-Tools geführt (z.B. `Jenkins`, `Cruise Control`, `Continuum`).

Generell sollte die Integration mit solchen Werkzeugen durch Verwendung der Kommandozeilen-Schnittstelle von QF-Test, die in Kapitel 25(340) und Kapitel 44(971) ausführlich dokumentiert ist, ohne Schwierigkeiten möglich sein.

Hinweis GUI Tests benötigen eine aktive Benutzersession um korrekt ausgeführt werden zu können. Sie finden im Kapitel Aufsetzen von Testsystemen(474) nützliche Tipps und Tricks für die Einrichtung Ihrer Testsysteme. Der technische Hintergrund ist in FAQ 14 beschrieben.

Die folgenden Abschnitte enthalten Beispiele für Integrationen mit einigen der oben genannten Tools.

29.1 Eclipse

Eclipse (http://eclipse.org) ist eine Open Source Entwicklungsumgebung zur Erstellung von Java-Applikationen.

QF-Test bietet mit dem Eclipse Plugin die Möglichkeit, während der Entwicklung einer Java-Applikation diese aus Eclipse heraus zu starten und Tests darauf auszuführen. Dabei können beliebige QF-Test Knoten gestartet werden: z.B. ganze Testfallsätze, einzelne Testfälle oder auch nur ein Mausklick.

Video Video-Anleitung:

’Das QF-Test Eclipse Plugin’
https://www.qftest.com/de/yt/eclipse-42.html

29.1.1 Installation

Für die Installation des Eclipse Plugins kopieren Sie bitte die Datei `de.qfs.qftest_9.0.0.jar` aus dem Unterverzeichnis `qftest-9.0.0/misc/` des QF-Test Installationsverzeichnisses in das Unterverzeichnis ”dropins” des Eclipse Installationsverzeichnisses. Nach dem (Neu-)Start von Eclipse steht das Plugin zur Verfügung.

29.1.2 Konfiguration

Öffnen Sie den Konfigurationsdialog über den Eclipse Menüpunkt Run→Run Configurations. Geben Sie die gewünschten Werte im Reiter ”Main” und bei Bedarf in den Reitern ”Settings” und ”Initial Settings” ein. (Bei den Reitern ”Environment” und ”Common” handelt es sich um Eclipse Standardreiter, die für die Konfiguration des QF-Test Plugins nicht benötigt werden.)

Anschließend übernehmen Sie die Konfiguration mit ”Apply”. Gestartet wird mittels ”Run”.

Reiter ”Main”

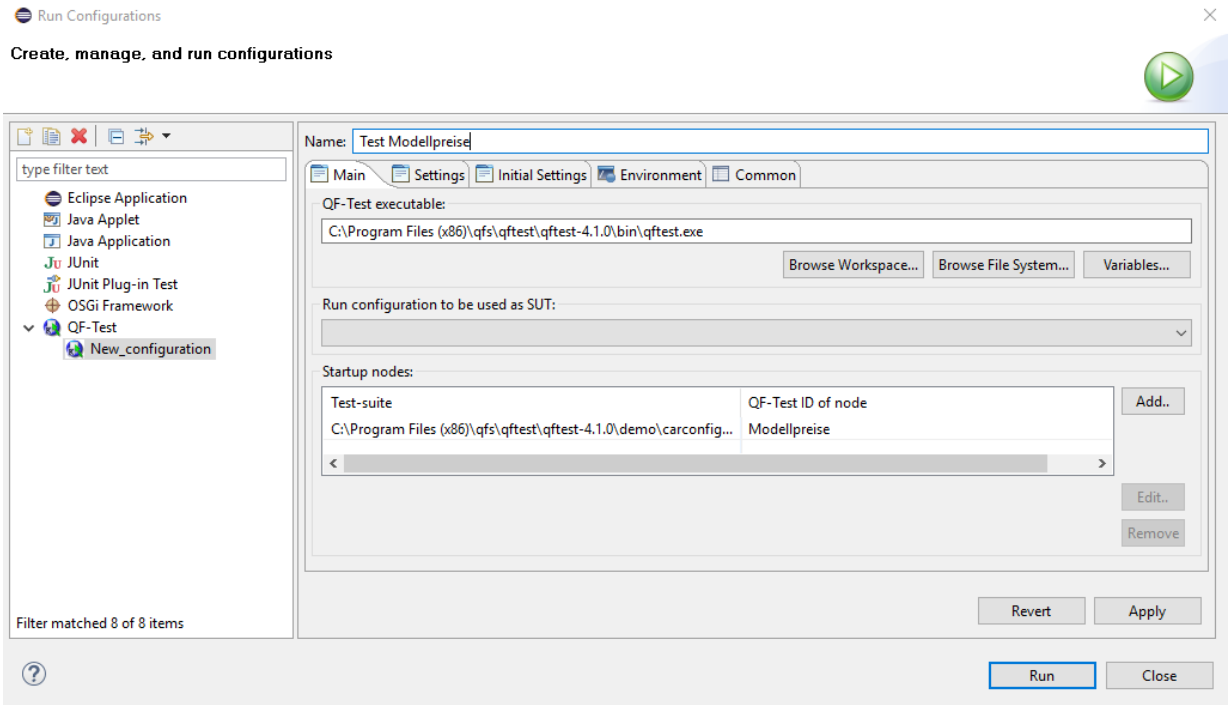


Abbildung 29.1: Eclipse Plugin Konfiguration - Reiter ”Main”

Im Feld ”QF-Test executable” geben Sie ”qftest.exe” inklusive Pfad an. Z.B. `C:\Program Files (x86)\qfs\qftest\qftest-4.1.0\bin\qftest.exe`.

Das Feld ”Run configuration to be used as SUT” ist optional. Hier kann eine in Eclipse angelegte ”Run Configuration” angegeben werden, über die die zu testende Applikation gestartet wird. Bei Start der Applikation wird eine Verbindung zu QF-Test hergestellt, so dass Tests auf der Applikation abgespielt und auch Aufnahmen von Tests gemacht werden können. Dies ist hilfreich, wenn die unter ”Startup nodes” konfigurierten QF-Test Knoten keine Startsequenz enthalten um die Applikation zu starten. Bitte beachten Sie dabei, dass die Ausführung der eingetragenen Run Configuration, die als SUT genutzt werden soll, angestoßen wird und sofort im Anschluss daran die Ausführung der aufgelisteten ”Startup nodes” beginnt. Das hat zur Folge, dass im Test als erstes darauf gewartet werden sollte, dass der Client gestartet ist. D.h. entweder sollte im ersten ”Startup node” der erste ausgeführte Knoten ein Warten auf Client sein oder der erste ”Startup node” sollte direkt einen Warten auf Client Knoten in QF-Test ausführen.

In der Tabelle ”Startup nodes” werden alle QF-Test Knoten eingetragen, die der Reihe nach ausgeführt werden soll. Ein Knoten werden anhand seiner QF-Test ID identifiziert sowie der Testsuite, in der er sich befindet. Bitte beachten Sie, dass die QF-Test ID

ein eigenständiges Attribut des Knotens ist und nicht dessen Name. Dieses Attribut ist standardmäßig leer und muss eigens gesetzt werden.

Reiter ”Settings”

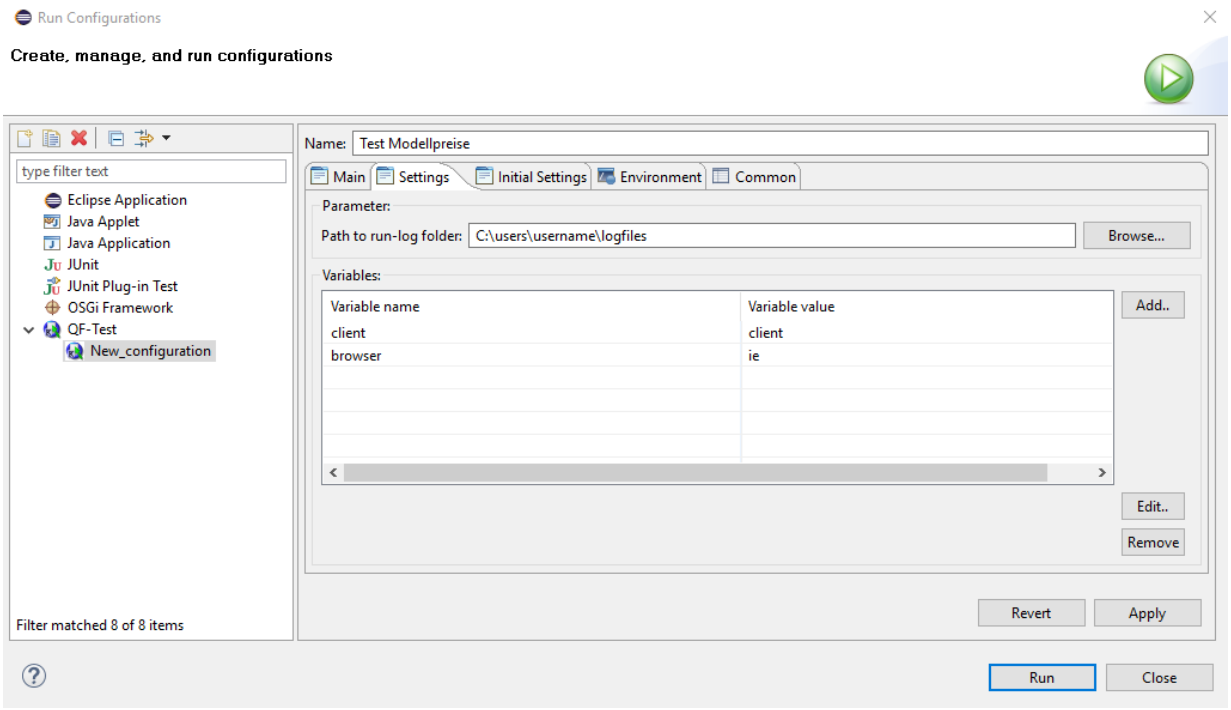


Abbildung 29.2: Eclipse Plugin Konfiguration - Reiter ”Settings”

Variablen in diesem Reiter werden vor jeder Ausführung der ”Run Configuration” neu eingelesen.

Im Feld ”Path to run log folder” kann das Verzeichnis angegeben werden, in dem die Protokolle für Testläufe aus dieser Run Configuration abgelegt werden. Die Angabe ist optional. Falls kein Wert angegeben wird, greift der Wert, der in der QF-Test Systemkonfiguration gesetzt wurde.

In der Tabelle ”Variables” können Variablen spezifiziert werden, die bei der Ausführung der Run Configuration auf Kommandozeilenebene an QF-Test übergeben werden. Damit können Default-Variablenwerte überschrieben werden, nicht jedoch im Test gesetzte globale oder lokale Variablen.

Reiter ”Initial Settings”

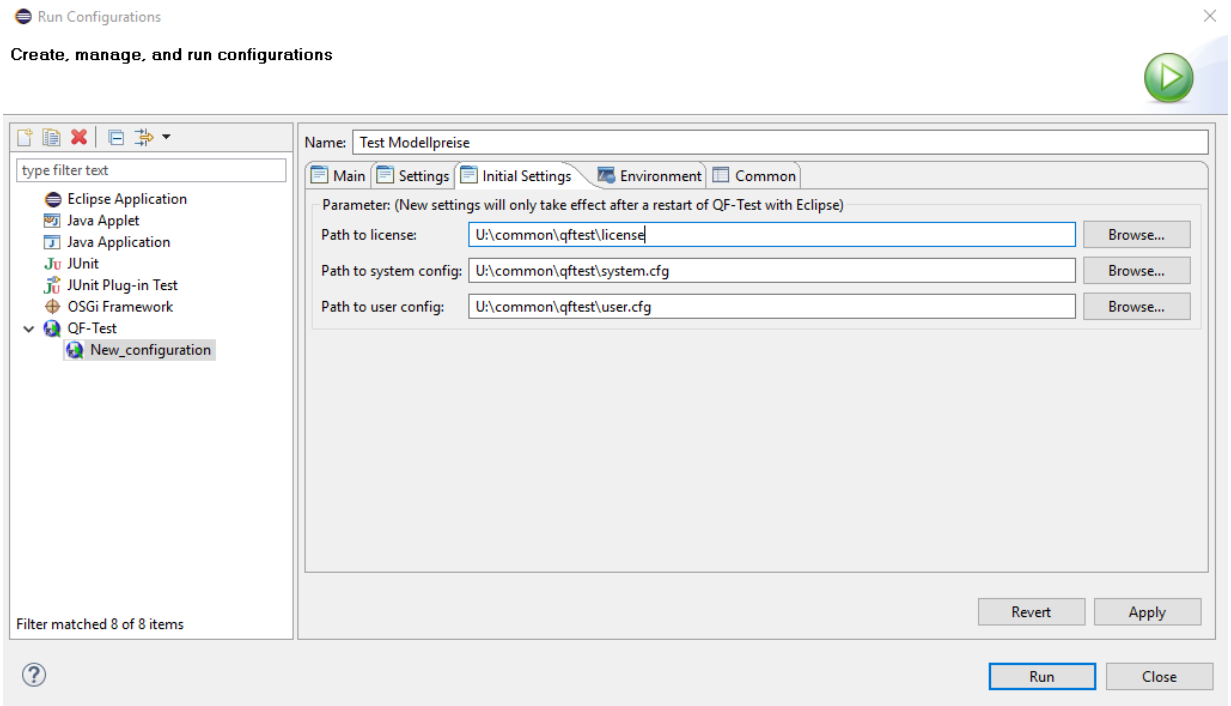


Abbildung 29.3: Eclipse Plugin Konfiguration - Reiter ”Initial Settings”

Die Werte dieses Reiters sind optional und werden beim Start von QF-Test ausgewertet. Eine Änderung der Werte erfordert einen Neustart von QF-Test aus Eclipse heraus bevor sie greifen.

Path to license file: Pfad der zu verwendenden Lizenzdatei.

Path to qftest system config file: Pfad der zu verwendenden qftest.cfg Datei.

Path to qftest user config file: Pfad der zu verwendenden benutzerspezifischen Konfigurationsdatei.

29.2 Ant

Wer Apache Ant (http://ant.apache.org) als Buildsystem einsetzt, kann die Testausführung mit QF-Test in die Builddatei integrieren:

```
<project name="QF-Test" default="runtest">
    <property name="qftest"
  location="c:\Program Files\qfs\qftest\qftest-9.0.0\bin\qftest.exe" />
    <property name="logdir" value="c:\mylogs" />
    <target name="runtest" description="Run a test in batchmode">
        <echo message="Running ${suite} ..." />
        <exec executable="${qftest}" failonerror="false"
  resultproperty="returncode">
            <arg value="-batch" />
            <arg value="-compact" />
            <arg value="-runlog" />
            <arg value="${logdir}\+b" />
            <arg value="${suite}" />
        </exec>
        <condition property="result"
  value="Test terminated successfully.">
            <equals arg1="${returncode}" arg2="0" />
        </condition>
        <condition property="result"
  value="Test terminated with warnings.">
            <equals arg1="${returncode}" arg2="1" />
        </condition>
        <condition property="result"
  value="Test terminated with errors.">
          <equals arg1="${returncode}" arg2="2" />
        </condition>
        <condition property="result"
  value="Test terminated with exceptions.">
            <equals arg1="${returncode}" arg2="3" />
        </condition>
        <echo message="${result}" />
    </target>
</project>
```

Beispiel 29.1: Ant Builddatei `build.xml` zur Ausführung einer Testsuite

Im obigen Beispiel wird davon ausgegangen, dass die auszuführende Testsuite beim Aufruf von `ant` als Property definiert wird: `ant -Dsuite="...\qftest-9.0.0\demo\carconfigSwing\carconfigSwing_en.qft"`.

29.3 Maven

Wer Apache Maven (http://maven.apache.org) als Buildsystem einsetzt, kann die Testausführung mit QF-Test in die Builddatei integrieren. Dies geschieht mittels Verwendung des antrun Plugins von Maven. Eine Beispiel `pom.xml` Datei, bei der die Tests in der

`test` Phase des Builds ausgeführt werden, könnte wie folgt aussehen:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
 http://maven.apache.org/maven-v4_0_0.xsd">
 <modelVersion>4.0.0</modelVersion>
 <artifactId>testant</artifactId>
 <packaging>jar</packaging>
 <name>testant</name>
 <groupId>de.qfs</groupId>
 <version>1</version>
 <properties>
  <qf.exe>"C:\Program Files\qfs\qftest\qftest-9.0.0\bin\qftest.exe"</qf.exe>
  <qf.reportfolder>qftest</qf.reportfolder>
  <qf.log>logFile.qrz</qf.log>
  <qf.suite>"c:\path\to\testsuite.qft"</qf.suite>
 </properties>
 <build>
     <plugins>
         <plugin>
             <groupId>org.apache.maven.plugins</groupId>
             <artifactId>maven-antrun-plugin</artifactId>
             <executions>
                 <execution>
                     <phase>test</phase>
                     <configuration>
                         <tasks>
                             <exec executable="${qf.exe}">
                                 <arg value="-batch"/>
                                 <arg value="-report"/>
                                 <arg value="${qf.reportfolder}"/>
                                 <arg value="-runlog"/>
                                 <arg value="${qf.log}"/>
                                 <arg value="${qf.suite}"/>
                             </exec>
                         </tasks>
                     </configuration>
                     <goals>
                         <goal>run</goal>
                     </goals>
                 </execution>
             </executions>
         </plugin>
     </plugins>
 </build>
</project>
```

Beispiel 29.2: Maven Builddatei `pom.xml` zur Testausführung

In Ihrem Projekt könnte es notwendig werden, die Tests nicht in der `test` Phase auszuführen. Hierfür sollten Sie das Plugin wie in der Mavendokumentation beschrieben konfigurieren.

29.4 Jenkins

3.3+

Video

Das Video

'QF-Test Jenkins Plugin'
https://www.qftest.com/de/yt/jenkins-plugin-40.html

zeigt die Einrichtung und Konfiguration des Plugins.

Bei Jenkins (jenkins-ci.org) handelt es sich um ein Continuous-Integration-Build-Tool. Es ist dazu gedacht, die Ausführung des Build-Prozesses eines Software-Projektes zu steuern und zu überwachen sowie die Ergebnisse zu verwalten. Ein wichtiger Schritt in diesem Build-Prozess sind automatisierte Tests, u.a. auch GUI-Tests mit QF-Test.

Eine Verwendung von QF-Test in Kombination mit Jenkins kann folgende Vorteile bringen:

- Falls bereits ein Continuous-Integration-Prozess mit Jenkins für die fortlaufende Erstellung der Software verwendet wird, kann das automatisierte Testen sehr einfach in den bestehenden Ablauf integriert werden.
- Einfache Verwaltung von zeitgesteuerten Starts der Tests sowie Benachrichtigung über das Ergebnis über Email oder RSS.
- Komfortable Übersicht und Kontrolle der ausgeführten Testläufe in einer webbasierten Oberfläche.
- Mit dem HTML Publisher Plugin können QF-Test HTML-Reports direkt in die Oberfläche von Jenkins/Jenkins integriert werden.
- Resultate, wie Protokolle und Reports, die während des Testlaufs erstellt werden, können automatisch archiviert werden. Somit ist es nicht mehr nötig, eine eigene Verzeichnisstruktur zu pflegen.

29.4.1 Jenkins installieren und starten

Hinweis

Für GUI Tests darf Jenkins nicht als Service eingerichtet sein sondern muss innerhalb einer echten Benutzer-Sitzung laufen. Unter Windows konfiguriert der Jenkins `.msi` Installer leider ohne Rückfrage direkt den Betrieb als Service, weshalb wir von diesem

abraten. Bitte stellen Sie sicher, dass Jenkins als echter Benutzerprozess startet, wie es unten beschrieben wird.

Die Installation von Jenkins beschränkt sich auf das Herunterladen des `war` Archivs von jenkins-ci.org/latest/jenkins.war und das Starten mittels `java -jar jenkins.war`.

Sobald Jenkins erfolgreich läuft, kann mittels eines Browsers über die URL `http://localhost:8080` auf die Weboberfläche zugegriffen werden, welche sich wie folgt darstellt:

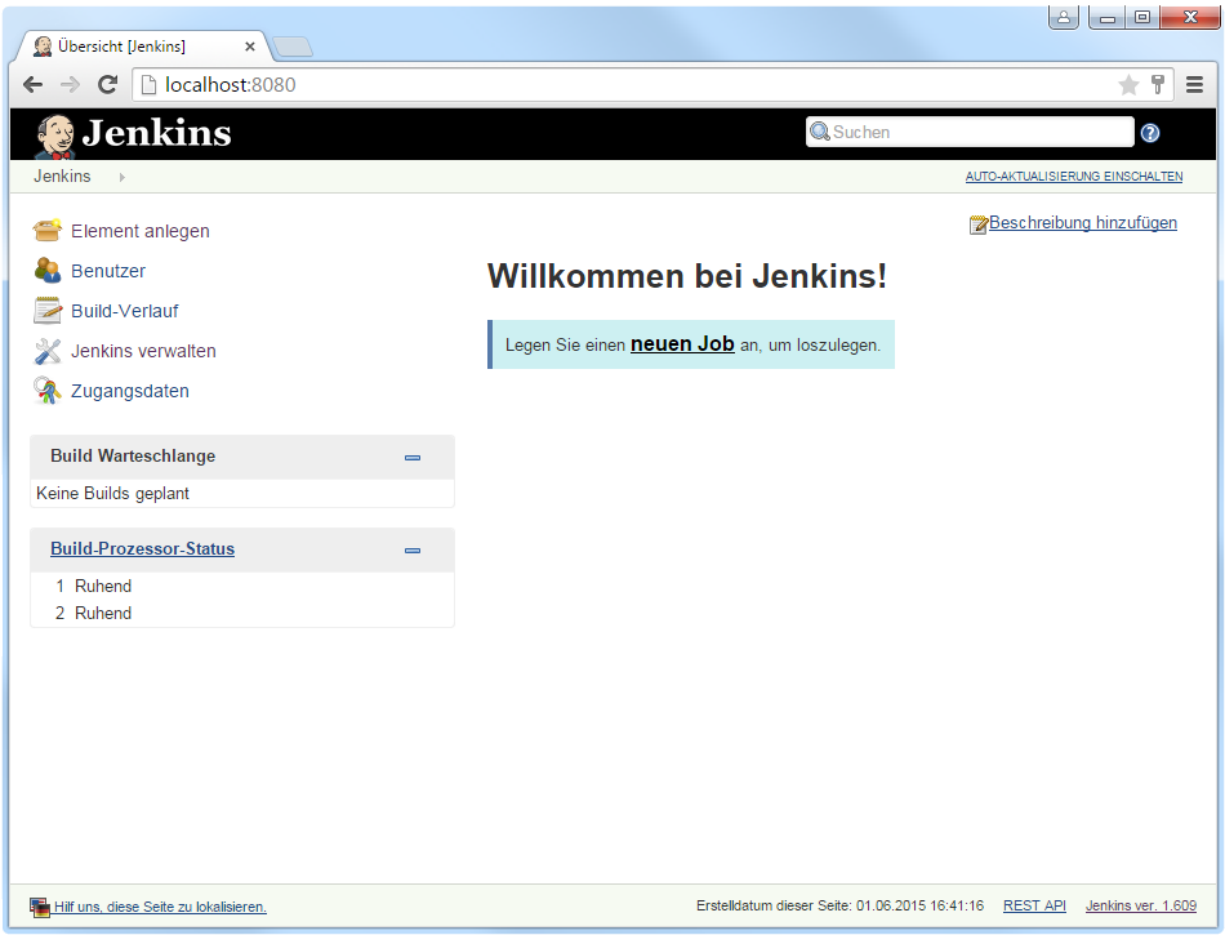


Abbildung 29.4: Jenkins nach dem Start.

29.4.2 Voraussetzungen für GUI-Tests

GUI-Tests benötigen einen ungesperrten, aktiven Desktop mit einer aktiven Benutzer-Session. So verhält sich das SUT genauso wie bei einem normalen Benutzer. Deshalb ist es nicht möglich, den Agenten als Windows-Service zu starten, sondern ein realer

(Test-) Benutzer muss angemeldet sein (z.B. mittels Auto-Login) und Start des Agenten über Windows-Autostart. Bildschirmsperren müssen deaktiviert sein. Sie finden im Kapitel Aufsetzen von Testsystemen[474] nützliche Tipps und Tricks für die Einrichtung des Jenkins-Prozesses.

Jenkins erlaubt die Ausführung von Aufgaben auf verteilten Rechnern. Dies ist natürlich auch für GUI-Tests relevant. Aufgrund Ihrer Natur werden GUI-Tests typischerweise nicht direkt auf dem zentralen Buildserver ausgeführt. Zusätzlich sollen häufig Tests für verschiedene Umgebungen, Betriebssysteme und Versionen des SUTs durchgeführt werden.

Auf einem dezentralen Rechner muss ein Jenkins Agent gestartet werden, um sich zum zentralen Jenkins Server zu verbinden und dann auf auszuführende Aufgaben zu warten. Wie in der Dokumentation von Jenkins beschrieben, gibt es verschiedene Möglichkeiten diesen Agenten zu starten, aber damit die GUI-Tests vollständig funktionieren können, müssen die Agenten via Java Web Start gestartet werden.

Hinweis Weitere technische Hintergrundinformation gibt FAQ 14.

29.4.3 Installation des QF-Test Plugins

Das QF-Test Plugin ermöglicht die Ausführung von QF-Test Suiten in Jenkins. Um das Plugin zu installieren, öffnen Sie das Jenkins Dashboard und gehen auf ”Jenkins verwalten”, gefolgt von ”Plugins verwalten”. Wählen Sie das QF-Test Plugin im ”Verfügbar” Tab aus und klicken Sie auf den Installieren-Button. Das QF-Test Plugin wird auch das JUNIT und das HTML-Publisher Plugin herunterladen, falls diese noch nicht installiert sind. Schließlich muss Jenkins neugestartet werden um die Installation abzuschließen. Das QF-Test Plugin erscheint nun in dem ”Installiert” Tab, wie in Abbildung 20.2 gesehen werden kann.

Hinweis Jenkins benutzt automatisch die zuletzt installierte Version von QF-Test. Falls eine andere Version genutzt werden soll, kann der Pfad in der QF-Test Sektion in den Jenkins Einstellungen angegeben werden (Jenkins verwalten -> System konfigurieren).

Abbildung 29.5: QF-Test Plugin installiert.

Sobald das QF-Test Plugin erfolgreich installiert wurde kann man die Testausführung mit QF-Test in den Jenkins-Buildprozess integrieren. Eine detaillierte Handreichung dazu findet sich in der Dokumentation des QF-Test Plugins unter https://www.qftest.com/en/jenkins.

29.5 JUnit 5 Jupiter

7.0+

In Kapitel 12[215] wurde beschrieben, wie sich JUnit-Tests komfortabel in eine QF-Test Testsuite einbinden lassen und ein gemeinsames Protokoll die Ergebnisse aus den übrigen Testfällen mit denen der Unit-Tests kombiniert. Mit Hilfe der Java-Annotation `@QFTest.Test` ist es möglich, das umgekehrte Szenario zu realisieren - QF-Test Testsuiten als Teil eines JUnit 5 Testfalls einzubinden und die Ergebnisse des QF-Test Testlaufs mit den Ergebnissen der übrigen JUnit-Testfälle zu kombinieren. Dies vereinfacht die Einbindung von QF-Test Testläufen sowohl in bestehende Maven- oder Gradle-Builds, als auch in Entwicklungsumgebungen wie Eclipse oder IntelliJ IDEA.

Konkret muss dafür in der Test-Klasse, welche die Ausführung von einer oder mehrerer QF-Test Testsuiten umfassen soll, eine Methode hinzugefügt werden, welche mit der Annotation `de.qfs.apps.qftest.junit5.QFTest.Test` markiert ist. Diese Methode muss ein Objekt des Typs `de.qfs.apps.qftest.junit5.QFTest` zurückgeben, welches durch Aufruf der statischen Methode `QFTest.runSuite` oder `QFTest.runSuites` erzeugt wird. Bei Bedarf kann dieses Objekt weiter konfiguriert werden, um zum Beispiel QF-Test Optionen oder Variablen zu spezifizieren. Die Dokumentation der dafür bereitgestellten Methoden findet sich in der Datei

doc/javadoc/qftest-junit5.zip innerhalb der QF-Test Installation.

```
import de.qfs.apps.qftest.junit5.QFTest;
import java.io.File;
public class QFTestDemoTest
{
  @QFTest.Test
  QFTest demoTest() throws Exception {
    // Demo-Testsuite lokalisieren
    final File qftestVerdir = QFTest.getVersionDir();
    final File demo = new File(qftestVerdir,
            "demo/carconfigSwing/carconfigSwing_de.qft");
    return QFTest.runSuite(demo)
                 .withVariable("buggyMode","True")
                 .withArgument("-verbose")
                 .withReportOpen();
  }
}
```

Beispiel 29.3: Beispiel einer JUnit 5-Testklasse, die einen QF-Test Testlauf einbezieht.

Zur Ausführung des Testes ist es notwendig die folgenden Bibliotheken aus der QF-Test Installation im Classpath einzubinden:

- lib/truezip.jar
- qflib/qflib.jar
- qflib/qfshared.jar
- qflib/qftest.jar

Wenn das Projekt mit Gradle gebaut wird, so kann man das de.qfs.qftest Gradle-Plugin anwenden, welches die Abhängigkeiten automatisch auflöst. Weitere Informationen dazu finden sich auf der Plugin Homepage.

```
plugins {
    id 'java'
    id 'de.qfs.qftest' version '1.1.0'
}
repositories {
    mavenCentral()
}
test {
    useJUnitPlatform()
}
```

Beispiel 29.4: Ausschnitt aus einer gradle.build-Datei, welche im JUnit-Testlauf QF-Test aufruft.

29.6 TeamCity CI

QF-Test kann leicht in TeamCity CI integriert werden, sodass Tests automatisch von TeamCity CI ausgeführt werden und Testergebnisse, Protokolle und HTML-Reports direkt aus der TeamCity-Benutzeroberfläche heraus eingesehen werden können.

Eine Schritt-für-Schritt-Anleitung hierzu finden Sie in unserem Blogartikel Integration von QF-Test mit TeamCity in drei einfachen Schritten .

Kapitel 30

Integration mit Robot Framework

6.0+

30.1 Einführung

Robot Framework ist ein sehr beliebtes Framework zur Testautomatisierung und robotergesteuerten Prozessautomatisierung (Robotic Process Automation - RPA). Es basiert auf Python und kommt mit einer Vielzahl an gebrauchsfertigen Bibliotheken für diverse Testszenarien. Meist wird die Entscheidung zwischen entweder QF-Test oder Robot Framework fallen, aber es gibt Situationen für welche die Integration absolut Sinn ergibt: Wenn Sie einerseits bereits existierende Infrastruktur für Robot Framework haben oder Tester in Ihrem Team über Spezialwissen zu Robot Framework verfügen und andererseits die besonderen Fähigkeiten von QF-Test in der UI-Automatisierung benötigen.

30.2 Voraussetzungen und Installation

Es muss eine aktuelle Version von Python 3 installiert sein.

Falls noch nicht vorhanden, kann Robot Framework via `pip install robotframework` installiert werden. Es wird Robot Framework Version 4 oder höher benötigt.

Die Integration benötigt eine Brücke zwischen Python und Java. JPype erfüllt diese Rolle ausgezeichnet. Es muss via `pip install JPype1` installiert werden.

QF-Test enthält eine Robot Framework Bibliothek namens `qftest`, die Robot Framework bekannt gemacht werden muss. Sie befindet sich im Verzeichnis `.../qftest-9.0.0/ext/robotframework`. Sie können entweder dieses Verzeichnis zu Ihrer PYTHONPATH Environment Variable hinzufügen, oder eine Datei namens `qftest_robot.pth` in den site-packages Ihrer Python 3 Installation anlegen - also `.../python3/Lib/site-packages/qftest_robot.pth` - die genau eine

Zeile enthält: Den vollständigen Pfad zu diesem Verzeichnis.

30.3 Erste Schritte

Robot Framework kommuniziert mit dem QF-Test Daemon, daher müssen Sie zunächst QF-Test, wie in Kapitel 55(1276) beschrieben, mit aktiviertem Daemon-Modus starten. Zum Erstellen von Tests ist es ideal, den interaktiven Daemon-Modus zu nutzen, da Sie dann den QF-Test Debugger aktiveren und dazu nutzen können, auf der Ebene von QF-Test in Einzelschritten durch Ihre Keywords zu gehen, unabhängig von und in Ergänzung zu dem Debugger der IDE, mit der Sie Ihre Robot Framework-Skripte ausführen. Starten Sie also bitte QF-Test von der Kommandozeile mit

```
qftest -daemon -daemonport 5454 -keystore=
```

Port 5454 ist nur ein Beispiel, Sie können diesen beliebig wählen, müssen ihn aber, wie unten beschrieben, in Ihrer robot Datei angeben.

Wie beim Kommandozeilenargument `-keystore <Keystore-Datei>`(983) beschrieben, dient `-keystore=` dazu, den Daemon ohne abgesicherte Kommunikation zu starten, was den Verbindungsaufbau beschleunigt und für den internen Gebrauch auf Ihrer Maschine akzeptabel sein sollte. Das dritte Argument der qftest Bibliothek muss in diesem Fall ”false” sein und ”true”, wenn ein Keystore genutzt wird.

Bevor Sie Ihre eigenen Robot Framework Tests mit QF-Test erstellen, sollten Sie zunächst das mit QF-Test bereitgestellte Demo-Skript ausführen, um sicherzustellen, dass Ihre Umgebung vollständig eingerichtet ist. Es befindet sich im Verzeichnis `.../qftest-9.0.0/demo/robotframework`. Bitte wechseln Sie dorthin und führen Sie folgenden Befehl aus:

```
robot carconfigSwing_de.robot
```

Dieses Skript sollte die Swing Carconfig Demo-Anwendung starten und darin ein paar Klicks und Checks durchführen. Wenn Sie dieses mehrfach ausführen sehen Sie einen weiteren großen Vorteil dieser Integration: Durch die Nutzung des QF-Test Daemon zum Start den Anwendung wird deren Lebensdauer unabhängig von der des Python Prozesses, der das Robot Framework-Skript ausführt. Es können also weitere Skripte ausgeführt werden und die bereits laufende Anwendung nutzen - dank des QF-Test Konzepts von Abhängigkeiten (Abschnitt 42.3(630)) trotzdem immer in einem definierten Zustand.

SmartIDs (vgl. Abschnitt 5.6(81)) sind ideal zur Angabe der Zielkomponenten von Robot Framework Keywords geeignet. Leider leitet das führende ’#’-Zeichen von SmartIDs in Robot Framework einen Kommentar ein, so dass dieses immer geschützt werden müsste, was die Lesbarkeit stark beeinträchtigt. Es gibt aber eine Option in QF-Test, nach deren Aktivieren jede Referenz auf eine Bemerkung auch ohne führendes ’#’ impli-

zit als SmartID behandelt wird, sofern kein Komponente(930) Knoten mit dieser ID existiert. Da diese spezifisch für Robot Framework ist, kann Sie nur auf Skriptebene aktiviert werden, wie in der Prozedur "use smartids without marker" der `robot.qft` Demo-Testsuite gezeigt:

```
rc.setOption(Options.OPT_SMARTID_WITHOUT_MARKER, true)
```

30.4 Nutzung der Bibliothek

Wie Sie der Datei `resource.txt` im Robot Framework Demo-Verzeichnis entnehmen können, muss die `qftest` Bibliothek wie folgt initialisiert werden:

```
Library    qftest    localhost    5454    false    ${SUITE}
```

Die Argumente sind optional. Die ersten drei legen Host und Port für den zu kontaktierenden QF-Test Daemon fest und ob ein Keystore verwendet werden soll oder nicht. Das vierte hat den Standardwert `robot.qft` und bezeichnet die primäre Testsuite zur Ermittlung der Keywords, die Robot Framework nutzen kann.

30.5 Erstellung eigener Keywords

Die Keywords für Robot Framework werden ermittelt, indem die primäre Testsuite, die als Argument bei der Definition der Library angegeben wurde, zusammen mit allen direkt oder indirekt per include referenzierten Testsuiten durchgegangen wird.

Hierbei wird das @keyword Doctag genutzt, um eine Prozedur oder eine ganze Package Hierarchie als Keyword festzulegen. Details hierzu finden Sie in Abschnitt 62.2(1362).

Kapitel 31

Schlüsselwortgetriebenes bzw. Keyword-Driven Testing mit QF-Test

31.1 Einführung

Das Konzept des schlüsselwortgetriebenen Testens, auch Keyword-Driven Testing genannt, erlaubt es Fachtestern ohne QF-Test Kenntnisse Testfälle in einer Metasprache oder einem Testmanagementtool zu spezifizieren. Diese Testfälle werden nun von QF-Test interpretiert und ausgeführt. Die reale Implementierung der Testschritte in QF-Test erfolgt durch Personen, die QF-Test programmieren können.

Die verwendete Art der Beschreibung, also die Schlüsselwörter, kann frei zwischen Fachtestern und QF-Test Experten vereinbart werden. Dabei reicht die Bandbreite bei der Festlegung der Schlüsselwörter von atomaren Aktionen (z.B. Mausklick auf Button) bis hin zu komplexen Vorgängen (z.B. Anlegen eines Vorgangs innerhalb der zu testenden Anwendung). Die Formulierung der Testschritte kann hierbei einerseits in streng definierten Tabelleneingaben (z.B. klickeButton=OK) oder andererseits sogar in Fließtext (z.B. Auf OK klicken, um Dialog zu schließen) erfolgen.

Um Ihnen einen Eindruck von den unterschiedlichen Varianten zu geben, stellen wir jetzt die Varianten anhand eines Testfalles für den QF-Test CarConfigurator dar. Der Testfall soll ein Fahrzeug in dieser Anwendung anlegen. Im Großen und Ganzen kann man von vier unterschiedlichen Arten sprechen:

Variante 1: Fachliche Testschritte (siehe Abschnitt 31.2.1 (416)). Eine Beispielimplementierung finden Sie unter `qftest-9.0.0/demo/keywords/simple_business`.

Testschritt
Anwendung starten, wenn notwendig
Fahrzeugdialog öffnen, unter Einstellungen -> Fahrzeuge
Daten wie Name und Preis eingeben
Auf Neu klicken, um das Fahrzeug anzulegen
Auf OK klicken, um den Dialog zu schließen
Die Anlage des Fahrzeuges prüfen

Tabelle 31.1: Testfall mit fachlichen Schlüsselwörtern

Variante 2: Atomare Testschritte (siehe Abschnitt 31.2.2(421)). Eine Beispielimplementierung finden Sie unter `qftest-9.0.0/demo/keywords/simple_atomic`.

Testschritt
Anwendung starten, wenn notwendig
Menüpunkt ”Einstellungen” auswählen
Menüpunkt ”Fahrzeuge...” auswählen
Textfeld ’Name’ füllen
Textfeld ’Preis’ füllen
Button ’Neu’ klicken, um das Fahrzeug anzulegen
Button ’OK’ klicken, um den Dialog zu schließen
Tabelle der Fahrzeuge prüfen, ob das neue Fahrzeug sich darin befindet

Tabelle 31.2: Testfall mit atomaren Schlüsselwörtern

Variante 3: Behavior-Driven Testing (BDT) (siehe Abschnitt 31.4.1(425)). Eine Beispielimplementierung finden Sie unter `qftest-9.0.0/demo/keywords/behaviordriven`.

Testschritt
Gegeben Anwendung läuft
Gegeben Dialog Fahrzeuge ist geöffnet
Wenn Textfeld Name auf Wert <name> gesetzt
Und Textfeld Preis mit Wert <preis> gefüllt
Und Button Neu geklickt wurde
Und Button OK geklickt wurde
Dann ist Tabelle Fahrzeuge eine neue Zeile mit den Werte <name> und <formatierter-preis> erschienen
Und Spalte Modell enthält den Wert <name>
Und Spalte Preis enthält den Wert <formatierter-preis>

Tabelle 31.3: Testfall mit Behavior-Driven Testing mit technischer Beschreibung

Variante 4: Behavior-Driven Testing (BDT) aus fachlicher Sicht (siehe Abschnitt 31.4.2[428]). Eine Beispielimplementierung finden Sie unter `qftest-9.0.0/demo/keywords/behaviordriven_business`.

Testschritt
Gegeben Anwendung ist bereit, um neue Fahrzeugdaten einzugeben
Wenn ein Fahrzeug mit dem Namen <name> und <preis> angelegt wurde
Dann erscheint das Fahrzeug mit den Werten <name> und <formatierter-preis> in der Fahrzeugübersicht

Tabelle 31.4: Testfall mit Behavior-Driven Testing aus fachlicher Sicht

In den folgenden Abschnitten werden nun diese unterschiedlichen Varianten für schlüsselwortgetriebenes Testen mit QF-Test genauer beschrieben. In einem weiteren Abschnitt wird auch das Konzept des Behavior-Driven Testings (BDT) beschrieben, welches aus QF-Test Sicht eine Sonderform des schlüsselwortgetriebenen Testens darstellt.

Die Beispiele beziehen sich allesamt auf den QF-Test CarConfigurator, welcher Teil der Auslieferung von QF-Test ist. Sie finden sämtliche Beispiele auch im Verzeichnis `qftest-9.0.0/demo/keywords/`. Für die Testplanung von sämtlichen Beispielen haben wir der Einfachheit halber Excel-Dateien mitgeliefert, welche die Testplanung der jeweiligen Testfälle beinhaltet. Natürlich kann in Ihrem Projekt diese Planung auch in Ihrem Testmanagementtool durchgeführt werden. Wir verwenden hier allerdings Excel-Dateien, weil diese Software fast überall verwendbar ist und QF-Test diese einfach lesen kann.

Im Abschnitt Keywords mit dynamischen Komponenten (siehe Abschnitt 31.3[422]) wird auch beschrieben, wie Sie QF-Test nur mehr als Ausführungsroboter für genau geplante

Testschritte einrichten können, um QF-Test nur mehr in Ausnahmefällen anfassen zu müssen.

Bitte achten Sie darauf, dass Sie alle Testsuiten vor dem eigentlichen Einsatz in einen projektspezifischen Ordner kopieren und diese dort modifizieren.

31.2 Einfaches Keyword-Driven Testing mit QF-Test

Die einfachste Form von Keywords stellt die Verwendung bereits bestehender Prozeduren dar. Prozeduren können sowohl fachlicher Natur als auch atomarer Natur sein. Fachliche Prozeduren stellen einen fachlichen Ablauf dar, z.B. das Anlegen eines Fahrzeuges. Atomare Prozeduren stellen einen atomaren Schritt des Testlaufes dar, z.B. Klick auf Button OK.

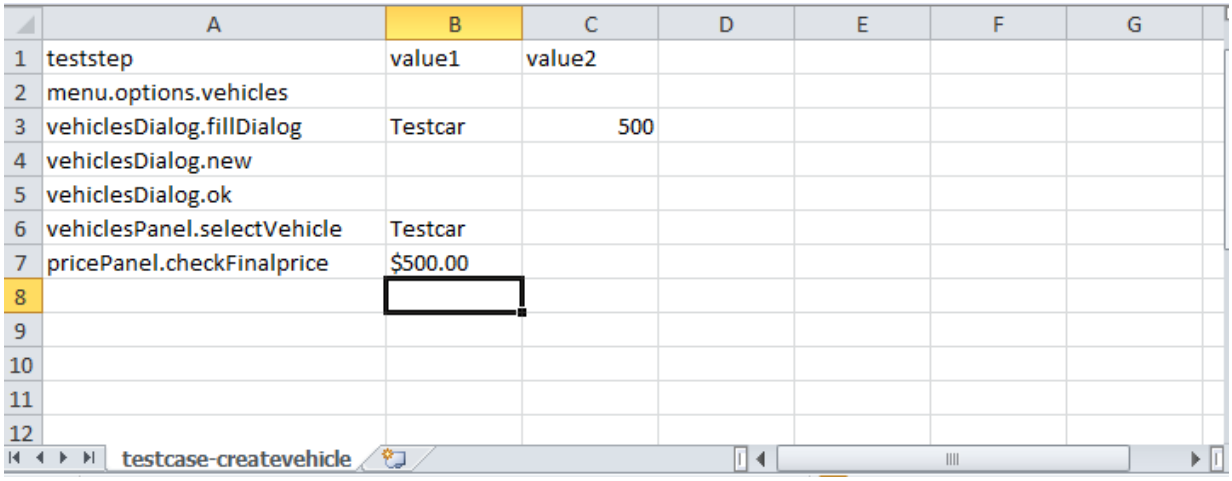
31.2.1 Fachliche Prozeduren

Wie eingangs bereits erwähnt stellen fachliche Prozeduren einen fachlichen Ablauf in Ihrer Anwendung dar. Das dazugehörige Beispiel finden Sie unter `qftest-9.0.0/demo/keywords/simple_business/SimpleKeywords.qft`. Der dazugehörige Testplan liegt unter `qftest-9.0.0/demo/keywords/simple_business/ simple_keywords.xlsx`. Bitte achten Sie darauf, dass Sie den Ordner in ein projektspezifisches Verzeichnis kopieren und diese dort modifizieren.

Als Beispiel können wir uns den Testfall ”Fahrzeug anlegen” für den CarConfigurator näher anschauen. Dieser Testfall besteht aus folgenden Schritten:

1. Anwendung starten, wenn notwendig
2. Fahrzeugdialog öffnen, unter Einstellungen -> Fahrzeuge
3. Daten wie Name und Preis eingeben
4. Auf Neu klicken, um das Fahrzeug anzulegen
5. Auf OK klicken, um den Dialog zu schließen
6. Die Anlage des Fahrzeuges prüfen

Wenden wir uns zuerst der Excel-Datei zu.



	A	B	C	D	E	F	G
1	teststep	value1	value2				
2	menu.options.vehicles						
3	vehiclesDialog.fillDialog	Testcar	500				
4	vehiclesDialog.new						
5	vehiclesDialog.ok						
6	vehiclesPanel.selectVehicle	Testcar					
7	pricePanel.checkFinalprice	$500.00					
8							
9							
10							
11							
12							

testcase-createvehicle

Abbildung 31.1: Excel-Datei fachliche Schlüsselwörter

Nachdem QF-Test Excel-Dateien zeilenweise lesen kann, haben wir uns für diesen Aufbau entschieden. Dieses Einlesen folgt dem Datentreiber Konzept (siehe Abschnitt 42.4(646)). Ein anderer Aufbau der Excel-Datei würde auch funktionieren. Der Vorteil des vorliegenden Beispiels ist, dass es ohne Skripte bzw. If-Abfragen auskommt.

In der ersten Zeile finden wir die Werte `teststep`, `value1` und `value2`. Diese Zeile wird später von QF-Test als Variablennamen interpretiert werden, die pro Zeile in der Excel unterschiedliche Werte annehmen werden. Nun soll QF-Test also schrittweise durch diese Excel-Dateien gehen und die geplanten Testschritte durchführen.

Um dieses Ziel zu erreichen, schauen wir uns nun die QF-Test Datei `SimpleKeywords.qft` an. Diese Datei hat folgenden Aufbau:

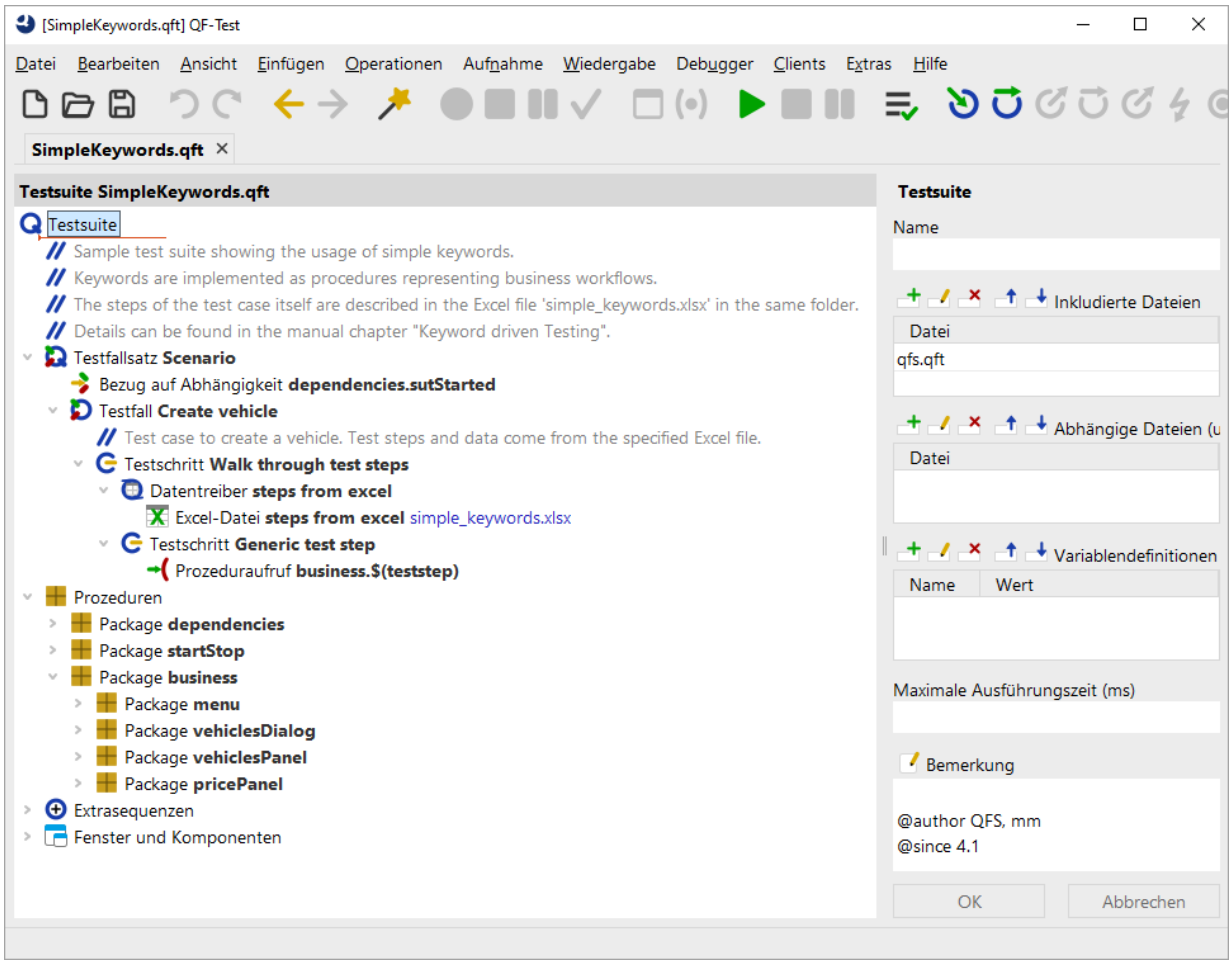


Abbildung 31.2: Testsuite fachliche Schlüsselwörter

Knoten	Zweck
Testfallsatz ”Scenario”	Bildet den äußeren Knoten für die Testausführung, kann theoretisch auch weggelassen werden.
Bezug auf Abhängigkeit dependencies.sutStarted	Das Starten und Stoppen der Anwendung sollten Sie auch bei diesem Ansatz einer QF-Test Abhängigkeit überlassen, weil diese nicht nur eine intelligente Verwaltung hierfür bieten, sondern auch über Mittel verfügen, auf unerwartetes Verhalten zu reagieren, siehe Abschnitt 42.3(630).
Testfall ”Create vehicle”	Dieser Testfall stellt die Implementierung des Testfalles dar.
Testschritt ”Walk through test steps”	Dieser Knoten wird benötigt, um die Excel-Datei mittels Datentreiber Knotens einzulesen.
Datentreiber ”steps from excel”	Dieser Knoten wird die zeilenweise Wiederholung für die Daten aus der Excel-Datei sicherstellen.
Excel-Datei ”steps from excel”	Hier wird auf die Excel-Datei verwiesen.
Testschritt ”Generic test step”	Dieser Testschritt wird während der Ausführung mit den Namen der Testschritte aus Excel gefüllt, um einen lesbaren Report zu erzeugen.
Prozeduraufruf ”business.$(teststep)”	Hier wird der entsprechende Testschritt, welcher in der Excel-Datei definiert ist, aufgerufen. Die Variable `teststep` wird hierbei aufgrund des Datentreiber-Mechanismuses auf die Planungsdaten der Excel-Datei gesetzt.

Tabelle 31.5: Aufbau von SimpleKeywords.qft

Die dafür benötigten Prozeduren sind im Package `business` implementiert. Damit Parameter dieser Prozeduren, die natürlich unterschiedlich sein können, in der Excel-Datei zusammengefasst werden können, werden die Excel-Parameter `value1` bzw. `value2` auf der jeweiligen Prozedur in die prozedurspezifischen Parameter umgewandelt.

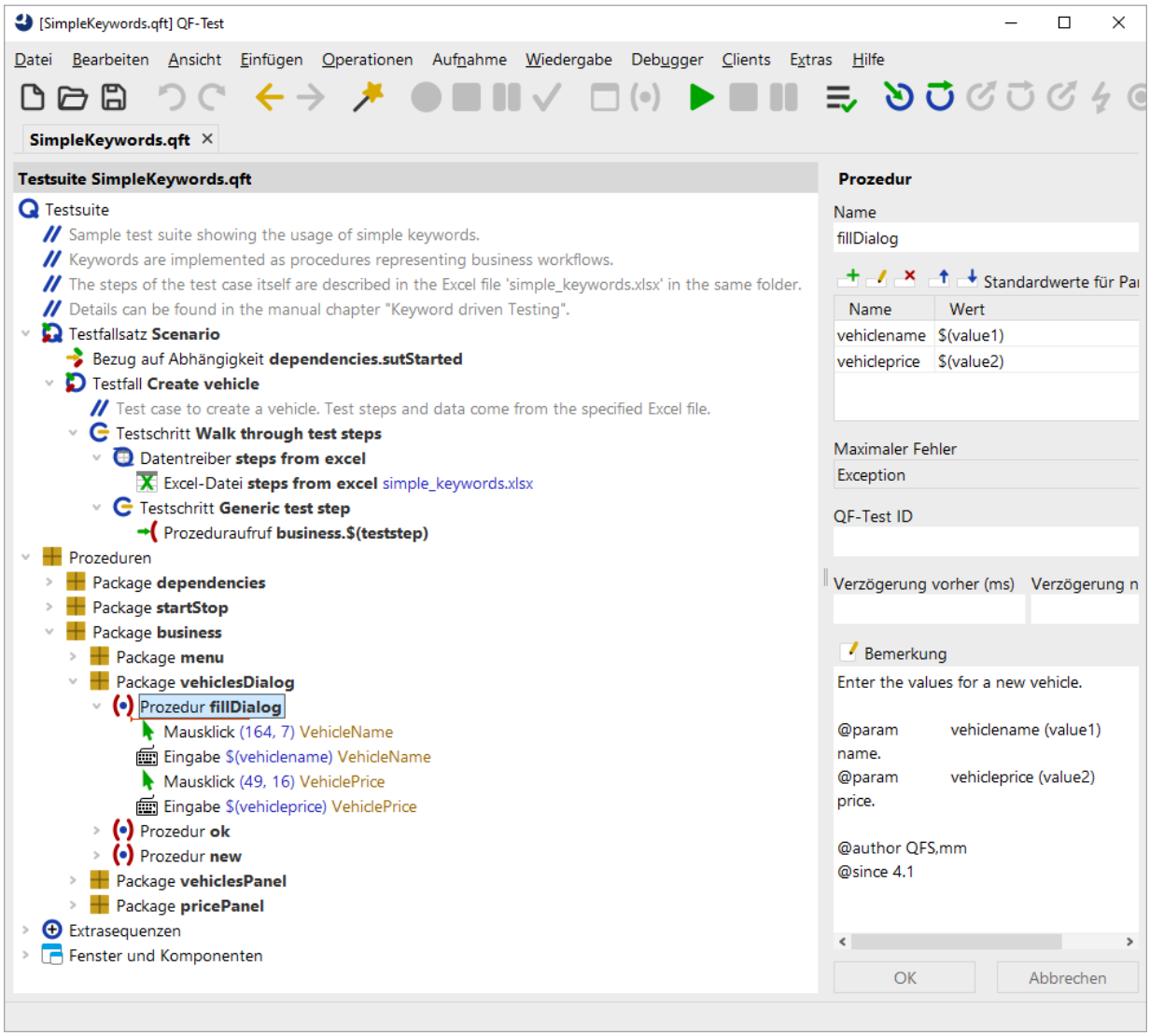


Abbildung 31.3: Prozedur fillDialog

Dieses Konzept erfordert also, dass die zu verwendenden Keywords bereits in QF-Test hinterlegt sind. Die Excel-Datei muss ggf. um mehrere Spalten für die Testdaten erweitert werden, falls Prozeduren mehr als zwei Parameter benötigen. Pro Testfall muss es noch einen Testfall in QF-Test geben. Dies können Sie allerdings auch flexibler gestalten. Eine Beschreibung hierzu finden Sie unter Abschnitt 31.5(429).

31.2.2 Atomare Prozeduren

Neben den vorher beschriebenen fachlichen Prozeduren, die kleinere Workflows darstellen, ist es auch möglich, jede einzelne Aktion zu beschreiben. Sie erhalten damit eine feingranularere Beschreibung der Testfälle. Das dazugehörige Beispiel finden Sie unter `qftest-9.0.0/demo/keywords/simple_atomic/SimpleAtomicKeywords.qft`. Der dazugehörige Testplan liegt unter `qftest-9.0.0/demo/keywords/simple_atomic/simple_atomic_keywords.xlsx`. Bitte achten Sie darauf, dass Sie den Ordner in ein projektspezifisches Verzeichnis kopieren und diese dort modifizieren.

Als Beispiel schauen wir uns wieder den Testfall ”Fahrzeug anlegen” für den CarConfigurator an. Dieser Testfall besteht nun aus folgenden Schritten:

1. Anwendung starten, wenn notwendig
2. Menüpunkt ”Einstellungen” auswählen
3. Menüpunkt ”Fahrzeuge...” auswählen
4. Textfeld Name füllen
5. Textfeld Preis füllen
6. Button Neu klicken, um das Fahrzeug anzulegen
7. Button OK klicken, um den Dialog zu schließen
8. Tabelle der Fahrzeuge prüfen, ob sich das neue Fahrzeug darin befindet

Ähnlich wie bei den fachlichen Prozeduren wird eine entsprechende Excel-Datei erstellt und es müssen die entsprechenden QF-Test Prozeduren hinterlegt werden. Die implementierten Prozeduren finden Sie im Package `atomic` in der Testsuite `qftest-9.0.0/demo/keywords/SimpleAtomicKeywords.qft`. Falls Sie sich für diesen Ansatz entscheiden, können Sie diese atomaren Prozeduren auch automatisch generieren (siehe Kapitel 27(368)).

Im nächsten Abschnitt werden wir sehen, wie wir diese Prozeduren dynamisch anwenden können, d.h. wir können danach Testfälle mit diesem atomaren Ansatz beschreiben, müssen allerdings nicht jede Aktion als Prozedur hinterlegen bzw. jede Komponente aufzeichnen, sondern nur einmal eine bestimmte Aktion, wie klickeButton oder setzeWert beschreiben. Danach können wir diese Aktion mehrfach anwenden.

31.3 Keyword-Driven Testing mit dynamischen/generischen Komponenten

Im vorigen Abschnitt haben wir gesehen, wie schlüsselwortgetriebenene Tests anhand einer Testplanung unterschiedliche Prozeduren aufrufen können. Allerdings ist hierbei die Erkennung der grafischen Komponenten noch in den QF-Test Prozeduren geblieben. Dieser Ansatz erfordert, dass jede Prozedur entsprechend vorab aufgezeichnet bzw. anderweitig erzeugt wird.

Es gibt allerdings noch die Möglichkeit die Information für die Komponentenerkennung auch in die Testplanung mitaufzunehmen und entsprechend in QF-Test zu interpretieren. Das dazugehörige Beispiel finden Sie unter `qftest-9.0.0/demo/keywords/generic/Keywords_With_Generics.qft`. Der dazugehörige Testplan liegt unter `qftest-9.0.0/demo/keywords/generic/keywords-generic.xlsx`. Bitte achten Sie darauf, dass Sie den Ordner in ein projektspezifisches Verzeichnis kopieren und diese dort modifizieren.

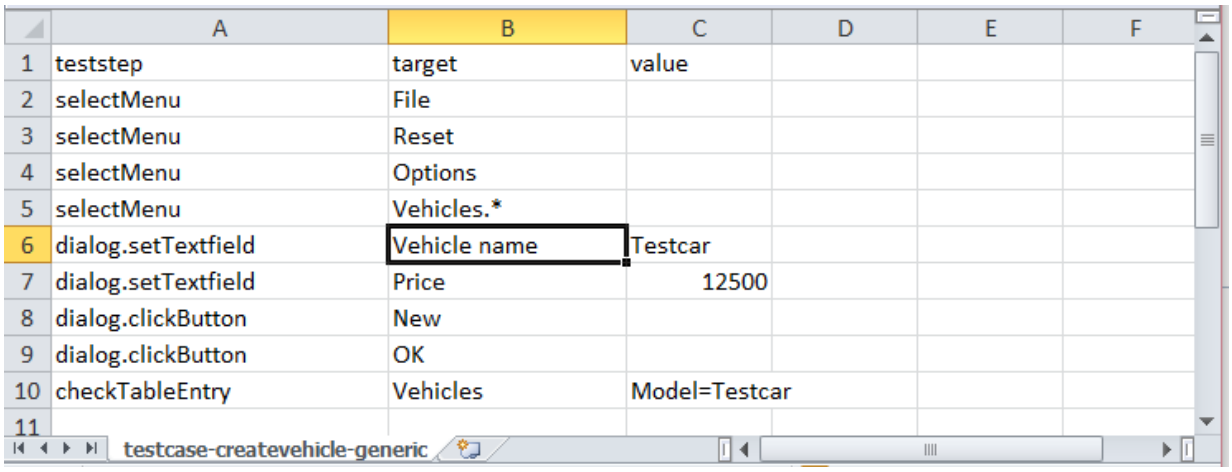
Grundlage dieses Verfahrens ist die generische Komponentenerkennung von QF-Test. Dieses Verfahren erlaubt es Ihnen Variablen in die aufgezeichneten Komponenten einzufügen, und sogar noch Komponenten aus der aufgezeichneten Hierarchie zu lösen., siehe Abschnitt 5.8(91).

Wenden wir uns wieder unserem Beispieltestfall zu. Der Testfall ”Fahrzeug anlegen”, wird nun wie folgt beschrieben:

1. Anwendung starten, wenn notwendig
2. Menüpunkt ”Einstellungen” auswählen
3. Menüpunkt ”Fahrzeuge...” auswählen
4. Textfeld Name füllen
5. Textfeld Preis füllen
6. Button Neu klicken, um das Fahrzeug anzulegen
7. Button OK klicken, um den Dialog zu schließen
8. Tabelle der Fahrzeuge prüfen, ob sich das neue Fahrzeug darin befindet

Wie Sie sehen, wird der Testfall genau so beschrieben, wie bei den atomaren Prozeduren im vorigen Abschnitt.

Die Excel-Datei hierzu sieht wie folgt aus:



	A	B	C	D	E	F
1	teststep	target	value			
2	selectMenu	File				
3	selectMenu	Reset				
4	selectMenu	Options				
5	selectMenu	Vehicles.*				
6	dialog.setTextfield	Vehicle name	Testcar			
7	dialog.setTextfield	Price	12500			
8	dialog.clickButton	New				
9	dialog.clickButton	OK				
10	checkTableEntry	Vehicles	Model=Testcar			
11						

testcase-createvehicle-generic

Abbildung 31.4: Excel-Datei generische Komponenten

In dieser Excel-Datei sehen wir nun, dass die Werte für `teststep` auf Werte wie `selectMenu` oder `dialog.clickButton` gesetzt sind. Des Weiteren ist eine neue Spalte `target` hinzugekommen. Diese Variable werden wir in den nächsten Absätzen genauer erklären. Wie wir in den vorigen Abschnitten schon gelernt haben, sollten hierfür entsprechende Prozeduren in der Testsuite `qftest-9.0.0/demo/keywords/generic/Keywords_With_Generics.qft` bestehen. Diese finden wir unter dem Package `generic`. Bitte achten Sie darauf, dass Sie den Ordner in ein projektspezifisches Verzeichnis kopieren und diese dort modifizieren.

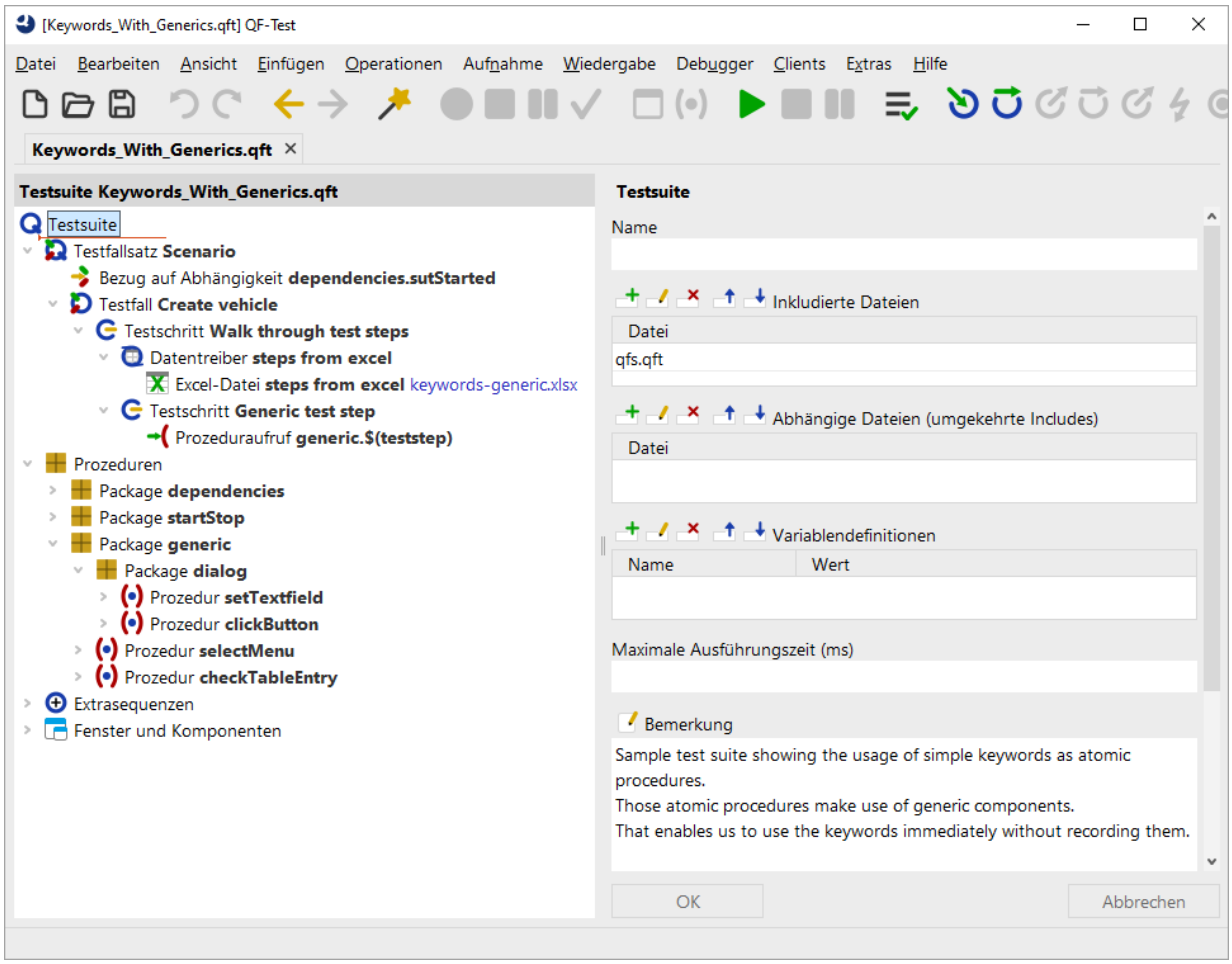


Abbildung 31.5: Testsuite generische Komponenten

Werfen wir kurz einen Blick auf die Prozedur `selectMenu`. Diese Prozedur besteht aus einem Mausklick auf die Komponente `GenericMenuItem`. Wenn wir nun zu der entsprechenden Komponente springen sehen wir, dass diese Komponente nur einen Wert für das Attribut `Klasse` besitzt, nämlich `MenuItem`, keinen Wert für Name und Merkmal, danach einen Wert für das `Weitere Merkmal qfs:label` mit dem Status `Muss übereinstimmen` und dem Wert `$(target)`. Das Attribut `Struktur` ist leer und bei den Werten für `Geometrie` steht ein '-'. Details zum '-', siehe Abschnitt 5.8(91).

Diese Definition bedeutet nun, dass die Erkennung dieser Komponente vom Inhalt der Variable `target` abhängt. Diese Variable wird für das `qfs:label` verwendet. Das weitere Merkmal `qfs:label` steht für die Beschriftung der Komponente, z.B. bei MenuItems oder Buttons oder für eine Beschriftung in der Nähe, z.B. bei Textfeldern. Wenn wir uns nun noch einmal der Excel-Datei zuwenden, werden wir sehen, dass in der Spalte `target` genau diese beschreibenden Texte der jeweiligen Komponenten stehen. Bei den anderen Komponenten ist das Vorgehen ähnlich.

Eine zweite Aufälligkeit ist das Package `dialog` unter dem Package `generic`. Grund hierfür ist, dass QF-Test für die Erkennung der Komponenten nicht nur die eigentliche Komponente, sondern auch immer das Fenster oder den Dialog mit in Betracht zieht. Bei Fenstern und Dialogen gibt es zwei Kategorien. Normale Fenster, die die Eingabe auf weiteren Fenstern nicht blockieren und so genannten modale Fenster, die die Eingabe auf weiteren Fenstern blockieren. Es ist in der Regel einfacher diese zwei Fensterkategorien über unterschiedliche Keywords anzusprechen. Man könnte allerdings auch dies noch weiter verallgemeinern. Bei Web-Anwendungen muss diese Unterscheidung nicht getroffen zu werden, weil sich alle Komponenten unterhalb einer Webseite befinden.

In diesem Abschnitt haben wir also gesehen, wie man die Komponentenerkennung mit Variablen versehen kann, um nur noch eine Prozedur pro Aktion und Zielkomponente anzulegen. Dies erlaubt es den gesamten Test in Excel zu definieren. Die notwendigen Prozeduren samt generischer Komponenten müssen noch am Anfang des Projektes angelegt werden. Natürlich kann dieser Ansatz mit aufgezeichneten Prozeduren ergänzt werden. Diese Prozeduren können wie im Abschnitt für fachliche Prozeduren (siehe Abschnitt 31.2.1[416]) angesprochen werden.

31.4 Behavior-Driven Testing (BDT)

Neben dem klassischen Keyword-Driven Testing existiert auch noch das Konzept des Behavior-Driven Testing oder kurz BDT. QF-Test ist gut integrierbar mit den Ansätzen und Tools zum Behaviourdriven Testing wie Cucumber/Gherkin (konkrete Anleitung zur Einrichtung und individuellen Anpassung bitte bei Quality First Software GmbH erfragen). In diesem Konzept werden, im Grunde genommen, Testfälle in Fließtext beschrieben, wobei der Satzanfang einem bestimmten vorgegebenen Vokabular unterliegt. Hierdurch sind Testfälle ohne Vorkenntnisse lesbar und verständlich. Wie auch beim Keyword-Driven Testing kann ein Testfall eher aus technischer Sicht (siehe Abschnitt 31.4.1[425]) oder aus fachlicher Sicht (siehe Abschnitt 31.4.2[428]) beschrieben werden. In den folgenden Abschnitten finden Sie Beispiele für beide Varianten.

31.4.1 Behavior-Driven Testing (BDT) mit technischer Beschreibung

Eine technische Beschreibung eines Testfalles mit Behavior-Driven Testing (BDT) orientiert sich an den grundlegenden Aktionen, die ausgeführt werden müssen. Das dazugehörige Beispiel finden Sie unter `qftest-9.0.0/demo/keywords/behaviordriven/BehaviorDrivenTesting.qft`. Der dazugehörige Testplan liegt unter `qftest-9.0.0/demo/keywords/behaviordriven/createvehicle.xlsx`.

Bitte achten Sie darauf, dass Sie den Ordner in ein projektspezifisches Verzeichnis kopieren und diese dort modifizieren.

Nach BDT mit technischer Sicht wird unser ”Fahrzeug anlegen” Testfall nun wie folgt beschrieben:

1. Gegeben Anwendung läuft
2. Gegeben Dialog Fahrzeuge ist geöffnet
3. Wenn Textfeld Name auf Wert <name> gesetzt
4. Und Textfeld Preis mit Wert <preis> gefüllt
5. Und Button Neu geklickt wurde
6. Und Button OK geklickt wurde
7. Dann ist Tabelle Fahrzeuge eine neue Zeile mit den Werte <name> und <formatierter-preis> erschienen
8. Und Spalte Modell enthält den Wert <name>
9. Und Spalte Preis enthält den Wert <formatierter-preis>

Der BDT Ansatz beschreibt Vorbedingungen und Aktionen mit Hilfe der Schlüsselwörter `Gegeben`, `Wenn`, `Und` und `Dann`, die am Satzanfang stehen müssen. Eine nähere Erläuterung dieses Konzeptes finden Sie in entsprechender Testliteratur.

Für QF-Test bedeutet dies, dass wir wieder Prozeduren bauen müssen, die dieser Beschreibung entsprechen. Hier hat sich bewährt, dass die BDT-Keywords als separate Packages erstellt werden. In der mitgelieferten Testsuite finden Sie hierfür auch die Packages `Given`, `When_And` und `Then`.

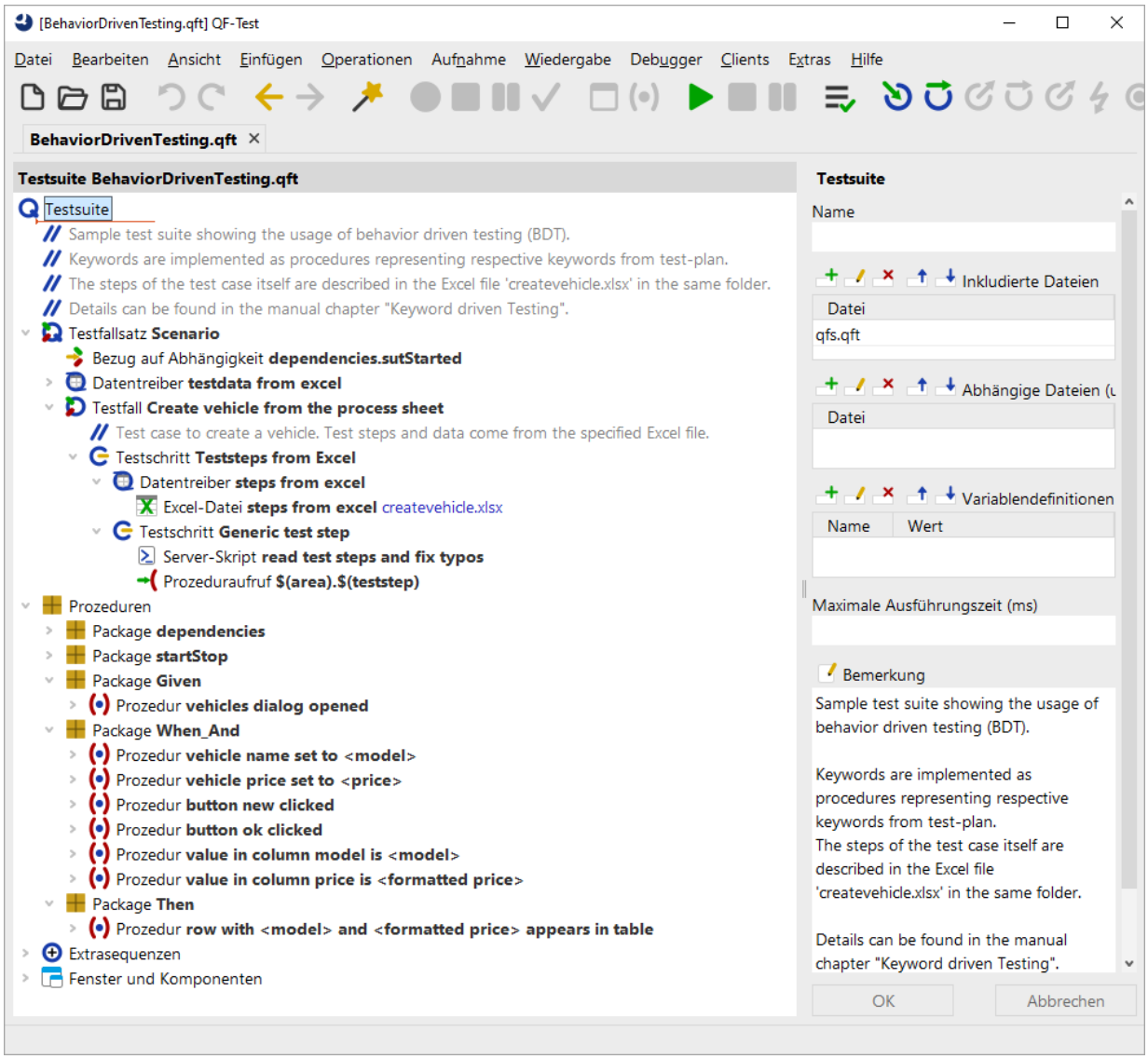


Abbildung 31.6: Testsuite Behavior-Driven Testing technisch

In unserer Beispiel-Testsuite liegen nun die entsprechenden Prozeduren unter den jeweiligen Packages, also z.B. eine Prozedur `vehicles dialog opened` unter dem Package `Given`. Um Flüchtigkeitsfehler bei der Beschreibung zu vermeiden wird vor dem Prozeduraufruf `$(teststep)` ein Server-Skript `read test steps and fix typos` eingeführt, dass die gesamte Beschreibung in Kleinbuchstaben formatiert und mehrfache Leerzeichen durch eines ersetzt.

Damit der selbe Testfall mit unterschiedlichen Testdaten abgespielt werden kann, wurde dieses Beispiel erweitert.

Natürlich könnte man hier auch versuchen, den generischen Erkennungsansatz, wie im

vorigen Abschnitt beschrieben (Abschnitt 31.3(422)), anzuwenden. Hierfür muss entweder die Beschreibung sehr genau sein oder das vorgelagerte Skript muss die Komponenten und deren Namen gut herausfinden können.

31.4.2 Behavior-Driven Testing (BDT) mit fachlicher Beschreibung

Eine fachliche Beschreibung eines Testfalles mit Behavior-Driven Testing (BDT) orientiert sich an aus Benutzersicht notwendigen Aktionen eines Testfalles. Diese Aktionen umfassen daher mehrere Interaktionen wie Mausklicks oder Texteingaben. Das dazugehörige Beispiel finden Sie unter `qftest-9.0.0/demo/keywords/behaviordriven_business/BehaviorDrivenTesting-Business.qft`. Der dazugehörige Testplan liegt unter `qftest-9.0.0/demo/keywords/behaviordriven_business/createvehicle-business.xlsx`. Bitte achten Sie darauf, dass Sie den Ordner in ein projektspezifisches Verzeichnis kopieren und diese dort modifizieren.

Nach BDT mit fachlicher Sicht wird unser ”Fahrzeug anlegen” Testfall nun wie folgt beschrieben:

1. Gegeben Anwendung ist bereit, um neue Fahrzeugdaten einzugeben
2. Wenn ein Fahrzeug mit dem Namen <name> und <preis> angelegt wurde
3. Dann erscheint das Fahrzeug mit den Werten <name> und <formatierter-preis> in der Fahrzeugübersicht

Wie auch im technischen Ansatz werden die Schlüsselwörter `Gegeben`, `Wenn`, `Und` und `Dann` verwendet und daher finden Sie auch hier entsprechende Packages in der mitgelieferten Testsuite. Dort sind diese Packages in englischer Sprache also `Given`, `When_And` und `Then`.

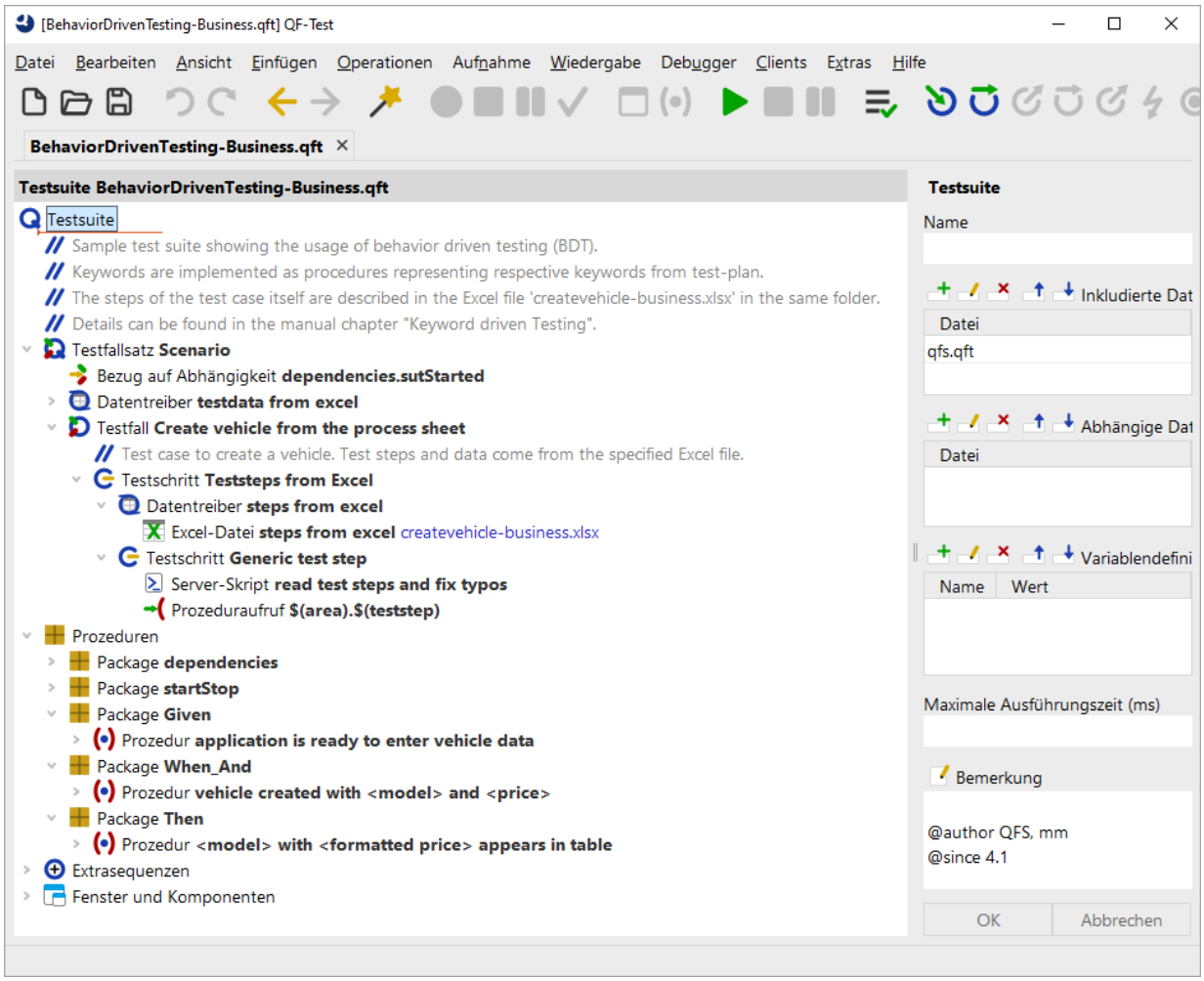


Abbildung 31.7: Testsuite Behavior-Driven Testing fachlich

Um Flüchtigkeitsfehler bei der Beschreibung zu vermeiden wird vor dem Prozeduraufruf `$(teststep)` ein Server-Skript `read test steps and fix typos` eingeführt, dass die gesamte Beschreibung in Kleinbuchstaben formatiert und mehrfache Leerzeichen durch eines ersetzt.

Damit der selbe Testfall mit unterschiedlichen Testdaten abgespielt werden kann, wurde dieses Beispiel erweitert.

31.5 Szenariodateien

Neben der Testfallbeschreibung, ist es natürlich auch möglich das gesamte Testszenario in einer Excel-Datei oder in Ihrem Testmanagementtool zu

beschreiben. In diesem Abschnitt wird der Einfachheit halber wieder eine Excel-Datei genutzt. Das dazugehörige Beispiel finden Sie unter `qftest-9.0.0/demo/keywords/generic_with_scenario/Keywords_With_Generics.qft`. Das dazugehörige Szenario liegt unter `qftest-9.0.0/demo/keywords/generic_with_scenario/ scenario.xlsx`. Die verwendeten Testfälle werden hier in einer separaten Excel-Datei beschrieben, siehe `qftest-9.0.0/demo/keywords/generic_with_scenario/keywords-generic-testcases.xlsx`. Bitte achten Sie darauf, dass Sie den Ordner in ein projektspezifisches Verzeichnis kopieren und diese dort modifizieren.

Das mitgelieferte Szenario besteht aus zwei Testfällen, die mit dem generischen Prozedurenansatz beschrieben sind (Abschnitt 31.3(422)). Natürlich können alle anderen Ansätze auch entsprechend konfiguriert werden.

Schauen wir uns zuerst die Excel-Datei an.

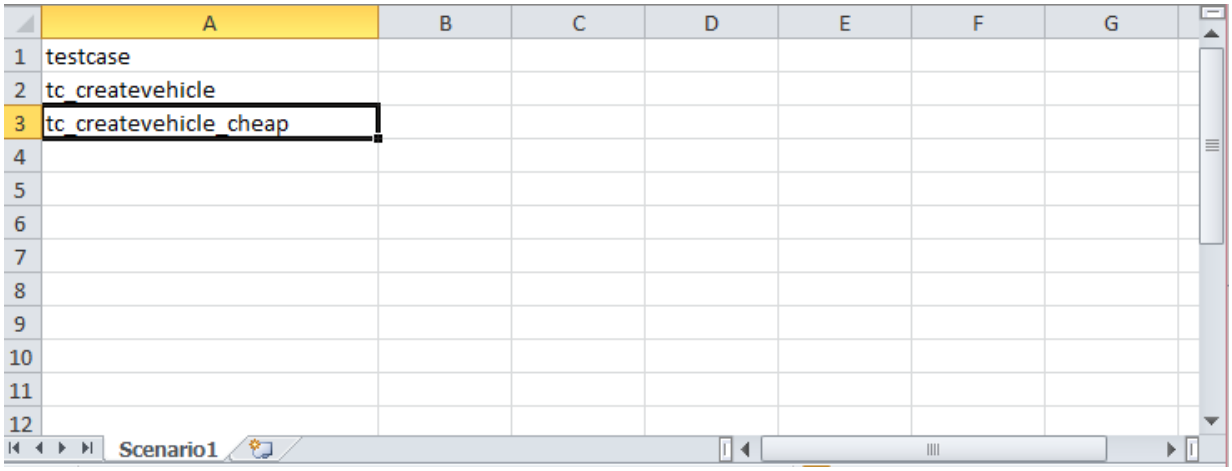


Abbildung 31.8: Excel-Datei als Szenariodatei

Im Tabellenblatt ”Scenario” ist in der ersten Spalte der Wert ”testcase” aufgeführt, welcher später wieder als Variable verwendet werden wird. Jede weitere Zeile steht für einen Testfall. Diese Testfälle entsprechen auch den Namen der Tabellenblätter in der Excel-Datei `keywords-generic-testcases.xlsx`. In diesen Tabellenblättern ”tc_createvehicle” und ”tc_createvehicle_cheap” finden wir auch die entsprechenden Testfallbeschreibungen.

Die dazugehörige Testsuite hat nun folgenden Aufbau:

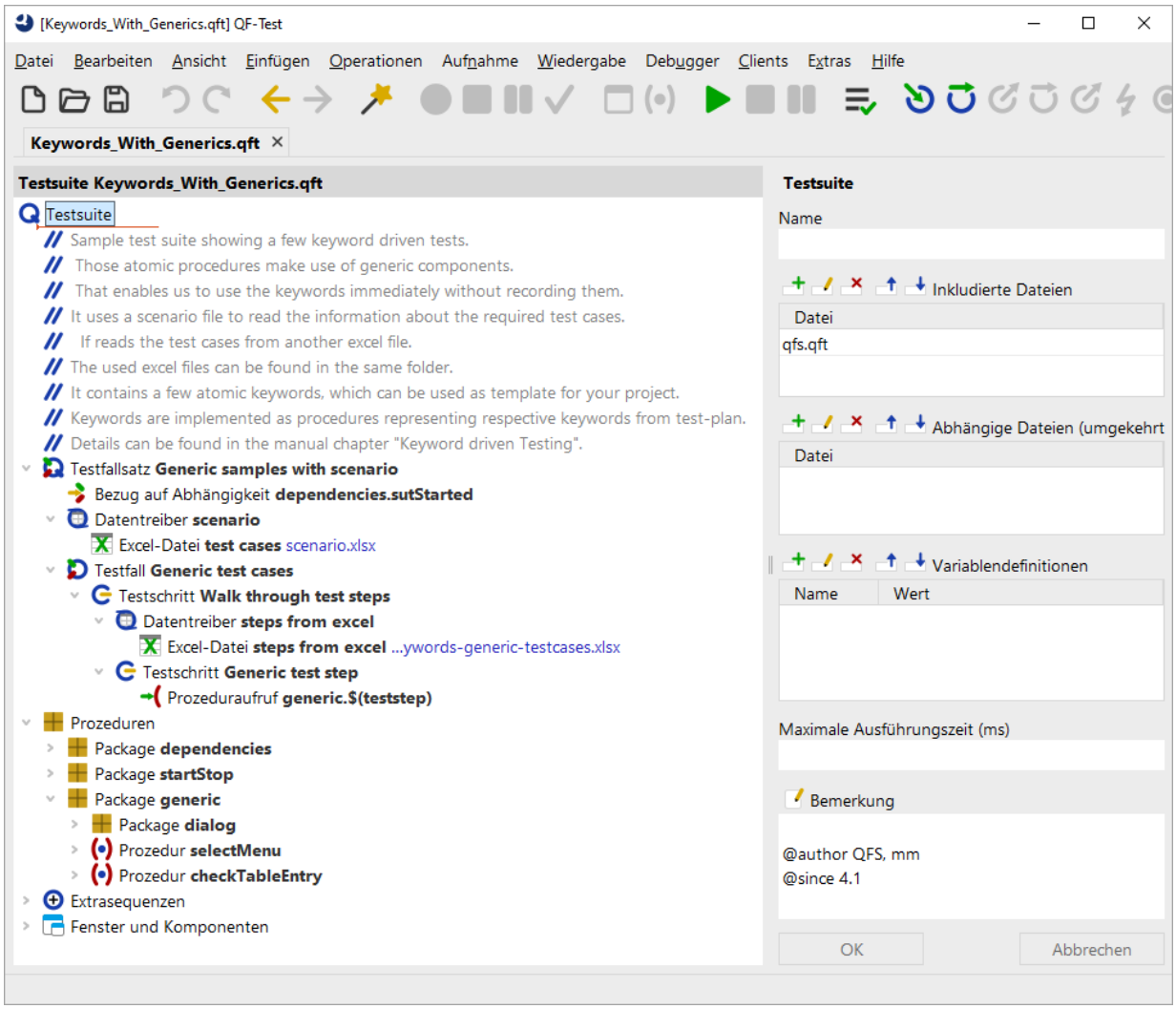


Abbildung 31.9: Testsuite Szenariodateien

Knoten	Zweck
Testfallsatz ”Generic samples with scenario”	Bildet den äußeren Knoten für die Testausführung, kann theoretisch auch weggelassen werden.
Bezug auf Abhängigkeit dependencies.sutStarted	Das Starten und Stoppen der Anwendung sollten Sie auch bei diesem Ansatz einer QF-Test Abhängigkeit überlassen, weil diese nicht nur eine intelligente Verwaltung hierfür bieten, sondern auch über Mittel verfügen, auf unerwartetes Verhalten zu reagieren, siehe Abschnitt 42.3(630).
Datenreiber ”scenario”	Dieser Knoten wird die auszuführenden Testfälle aus der Szenariodatei einlesen.
Excel-Datei ”test cases”	Hier wird auf die Szenario-Excel-Datei verwiesen.
Testfall ”Generic test case”	Dieser Testfall stellt die Implementierung des Testfalles dar.
Testschritt ”Walk through test steps”	Dieser Knoten wird benötigt, um die Excel-Datei für die einzelnen Testschritte mittels Datentreiber Knotens einzulesen.
Datenreiber ”steps from excel”	Dieser Knoten wird die Testschritte zeilenweise auslesen.
Excel-Datei ”steps from excel”	Hier wird auf die Excel-Datei für die Testfälle verwiesen.
Testschritt ”Generic test step”	Dieser Testschritt wird während der Ausführung mit den Namen der Testschritte aus Excel gefüllt, um einen lesbaren Report zu erzeugen.
Prozeduraufruf ”generic.$(teststep)”	Hier wird der entsprechende Testschritt, welche in der Testfall-Excel-Datei definiert ist aufgerufen. Die Variable `teststep` wird hierbei aufgrund des Datentreiber-Mechanismuses auf die Planungsdaten der Excel-Datei gesetzt.

Tabelle 31.6: Aufbau von Keywords_With_Generics.qft

31.6 Eigene Testbeschreibungen

In den vorigen Abschnitten haben wir die Beispiele anhand einer Beschreibung aus Excel-Dateien gesehen. Wie bereits erwähnt, können natürlich sämtliche Dateitypen, z.B. XML oder CSV-Dateien aber auch Rückgabewerte von Webservices ausgewertet werden. Hierfür ist es notwendig Server-Skripte zu implementieren, die die benötigte Information wie Testschrittname, Komponentenname etc. herauslesen und in QF-Test Variablen setzen. Dies können Sie mit der Skriptmethode `rc.setLocal` bzw. `rc.setGlobal` bewerkstelligen.

Neben dem Setzen von Variablen, wird es auch notwendig sein, Testfälle bzw. Pro-

zedurknoten aufzurufen. Hierfür stehen Ihnen die Skriptmethoden `rc.callTest` und `rc.callProcedure` zur Verfügung. Sie finden die vollständige API Beschreibung im Kapitel 11(186).

Einige Beispiele finden Sie in den mitgelieferten Testsuiten des ManualTesters (`qftest-9.0.0/demo/manualtester`) wie auch der imbus TestBench Integration (`qftest-9.0.0/ext/testbench`).

31.7 Anpassung an Ihre Software

Die mitgelieferten Beispiele sind für den QF-Test CarConfigurator erstellt worden. Sie können diese als Basis verwenden, um schlüsselwortgetriebenes Testen für Ihre Anwendung zu implementieren. Aufgrund der Vielfalt von technischen Möglichkeiten zur Erstellung von Anwendungen und unterschiedlichen Testvarianten, können diese Beispiele nur als Vorlage und nicht als vollständige Lösung dienen. Natürlich können Sie Ihre Situation und Wünsche auch unserem Supportteam mitteilen, um einen passenden Lösungsansatz zu finden.

Das vollständige Beispiel finden Sie im Ordner `qftest-9.0.0/demo/keywords/full_sample_for_carconfig`. Dort werden neben den bekannten generischen Prozeduren auch einige erweiterte Tabellenprozeduren verwendet. Die Testfälle werden mittels Szenariodatei, wie im Abschnitt 31.5(429) beschrieben, eingebunden. Bitte achten Sie darauf, dass Sie den Ordner in ein projektspezifisches Verzeichnis kopieren und diese dort modifizieren.

Technologie	Notwendige Anpassungen
JavaFX	1. Bei den Fensterkomponenten muss das GUI-Engine Attribut `awt` durch `fx` ersetzt werden. 2. Ggf. muss die Erkennung der Fenster und Dialoge mit zusätzlichen Variablen versehen werden. 3. Die Prozedur `startStop.startSUT` muss an Ihre Anwendung angepasst werden. Kopieren Sie hierzu die erstellten Schritte aus dem Schnellstart Assistenten. 4. Ggf. müssen einige Resolverskripte erstellt werden, um vernünftige Erkennungsmerkmale zu bekommen.
Java/Swing	1. Ggf. muss die Erkennung der Fenster und Dialoge mit zusätzlichen Variablen versehen werden. 2. Die Prozedur `startStop.startSUT` muss an Ihre Anwendung angepasst werden. Kopieren Sie hierzu die erstellten Schritte aus dem Schnellstart Assistenten. 3. Ggf. müssen einige Resolverskripte erstellt werden, um vernünftige Erkennungsmerkmale zu bekommen.
Java/SWT	1. Bei den Fensterkomponenten muss das GUI-Engine Attribut `awt` durch `swt` ersetzt werden. 2. Ggf. muss die Erkennung der Fenster und Dialoge mit zusätzlichen Variablen versehen werden. 3. Die Prozedur `startStop.startSUT` muss an Ihre Anwendung angepasst werden. Kopieren Sie hierzu die erstellten Schritte aus dem Schnellstart Assistenten. 4. Ggf. müssen einige Resolverskripte erstellt werden, um vernünftige Erkennungsmerkmale zu bekommen.
Web	1. Statt dem Fenster Komponenten benötigen Sie hier einen Knoten `Webseite`. 2. Dialoge in Web-Anwendungen sind Teil des Webseite, hier müssen Sie also eine Komponente innerhalb der Webseite anlegen. 3. Die Prozedur `startStop.startSUT` muss an Ihre Anwendung angepasst werden. Kopieren Sie hierzu die erstellten Schritte aus dem Schnellstart Assistenten. 4. Ggf. müssen erst Klassen und Erkennungsmerkmale mittels CustomWebResolver korrekt konfiguriert werden.

Tabelle 31.7: Notwendige Anpassungen an Ihr SUT

Kapitel 32

Verwendung von QF-Test in Docker Umgebungen

32.1 Was ist Docker?

Docker ist eine freie Virtualisierungssoftware, mit der beliebige Anwendungen sehr einfach auf physischen Computern oder in der Cloud installiert und ausgeführt werden können.

Entwickelt wurde Docker ursprünglich für das Betriebssystem Linux. Inzwischen gibt es Docker für weitere Plattformen, darunter Microsoft Windows und macOS. Auch auf Cloud-Diensten wie Amazon Web Services (AWS) und Microsoft Azure läuft die Virtualisierungssoftware.

Im Unterschied zu virtuellen Maschinen sind Docker-Container wesentlich ressourcensparender, da bei ihnen die Installation eines Gast-Betriebssystems entfällt.

32.2 QF-Test Docker Images

Seit QF-Test Version 6.0.3 gibt es offizielle Docker Images, welche es erlauben QF-Test relativ einfach zu virtualisieren.

Um verschiedene Anwendungsszenarien zu ermöglichen gibt es aktuell 4 verschiedene Docker-Images je QF-Test Version auf Docker Hub. Die genauen Details und Anwendungsmöglichkeiten zu den einzelnen Images sind ebenfalls dort beschrieben.

Es gibt jeweils ein Image mit und ohne vorinstallierten Web-Browser speziell für Web-Tests sowie eine Variante mit oder ohne zusätzlichen VNC Server, der die visuelle Kontrolle der Testausführung sowie evtl. notwendiges Debugging von Tests erlaubt.

Die bereitgestellten Images dienen für Ihre eigenen Anwendungsszenarien nur als Basis und können durch Verwendung dieser als Basis-Image in einem Dockerfile noch entsprechend erweitert werden.

Im Februar 2023 fand ein Spezial-Webinar zum Thema Docker statt, das nach etwas Theorie auch die detaillierten Schritte zur Nutzung der QF-Test Images auf Docker Hub zeigt.

Video

Hier geht es zum

Videomitschnitt des Spezial-Webinars
https://www.qftest.com/de/yt/docker-spezialwebinar.html

auf unserem QF-Test YouTube Kanal.

Kapitel 33

Durchführung von Lasttests mit QF-Test

Video

Video:

Lasttests
https://www.qftest.com/de/yt/lasttests-5.1.html

33.1 Hintergrund und Vergleich mit anderen Techniken

Neben funktionalen und Systemtests eignet sich QF-Test auch zur Durchführung von Lasttests, Stresstests oder Performance-Tests. Diese Art von Tests eignet sich auch um die Stabilität Ihrer gesamten Server/Client Umgebung zu prüfen.

Dabei wird die Performance einer Server-Anwendung getestet, indem eine Anzahl von GUI-Clients gleichzeitig ausgeführt wird. Zusätzlich kann mit QF-Test die, vom Benutzer erfahrene, Zeit an der grafischen Oberfläche, die so genannte End-To-End Zeit, gemessen werden. Im weiteren Verlauf des Kapitels wird der Einfachheit halber nur noch der Begriff Lasttests verwendet.

Es gibt verschiedene Möglichkeiten, Lasttests durchzuführen. Im Gegensatz zu QF-Test arbeiten die meisten davon nicht mit echten GUI-Clients, sondern setzen direkt auf der Protokollschicht zwischen GUI-Client und Server auf, z.B. indem sie HTTP Anfragen absetzen oder mittels RMI oder anderer Middleware Methoden im Server aufrufen.

Protokoll-basierte und GUI-basierte Lasttests haben jeweils verschiedene Vor- und Nachteile:

- Ressourcenverbrauch:
  Protokoll-basierte Tests benötigen nur geringe Ressourcen auf der Clientseite, so dass sie gut bis zum Überlastungspunkt des Servers skalieren, ohne dafür

sehr viel Hardware zu benötigen. GUI-basierte Tests benötigen dagegen für jeden Client entsprechend Rechenzeit und Speicher und gerade bei Rich Clients mit Swing oder JavaFX ist das nicht wenig. Außerdem erzeugt jeder Client ein GUI und braucht daher eine aktive Benutzersession.

- Aufwand Testerstellung:
  Rich Clients haben meistens ein komplexes User Interface, das mit einer gewissen Komplexität der Client-/Server-API korreliert. Die Erstellung von protokollbasierten Tests, die diese API weitgehend abdecken, kann daher sehr aufwändig sein. Dagegen stehen eventuell bereits GUI-basierte Tests zur Verfügung, die für die funktionalen Tests entwickelt wurden und die mit geringen Änderungen für die Lasttests wieder verwendet werden können. Selbst wenn das nicht der Fall ist, ist es wesentlich einfacher, mit QF-Test komplette Use Cases zu automatisieren als auf Protokollebene.

- Messbare Zeiten:
  Mit GUI-basierten Tests können so genannte End-To-End Zeiten ermittelt werden, d.h. die Zeit von einer Aktion des Anwenders bis zur Anzeige des Ergebnisses an der Oberfläche. Protokoll-basierte Tests messen dagegen nur die Zeit für die Anfrage an den Server. Beides kann sinnvoll sein, je nach Situation.

In einigen Fällen kann es durchaus sinnvoll sein, beide Welten zu kombinieren. Sie können z.B. auf einigen System GUI Tests starten, die die End-To-End Zeiten ermitteln und parallel dazu führen Sie protokoll-basierte Tests aus um Last zu erzeugen.

Zusammenfassend lässt sich sagen, dass GUI-basierte Lasttests sehr nützlich und effizient sein können (insbesondere wenn sich funktionale Tests wiederverwenden lassen), vorausgesetzt, es steht ausreichend Hardware zur Verfügung.

Abschließend finden Sie hier noch ein Architekturbild, welches alle involvierten Systeme eines GUI-basierten Lasttests zeigt:

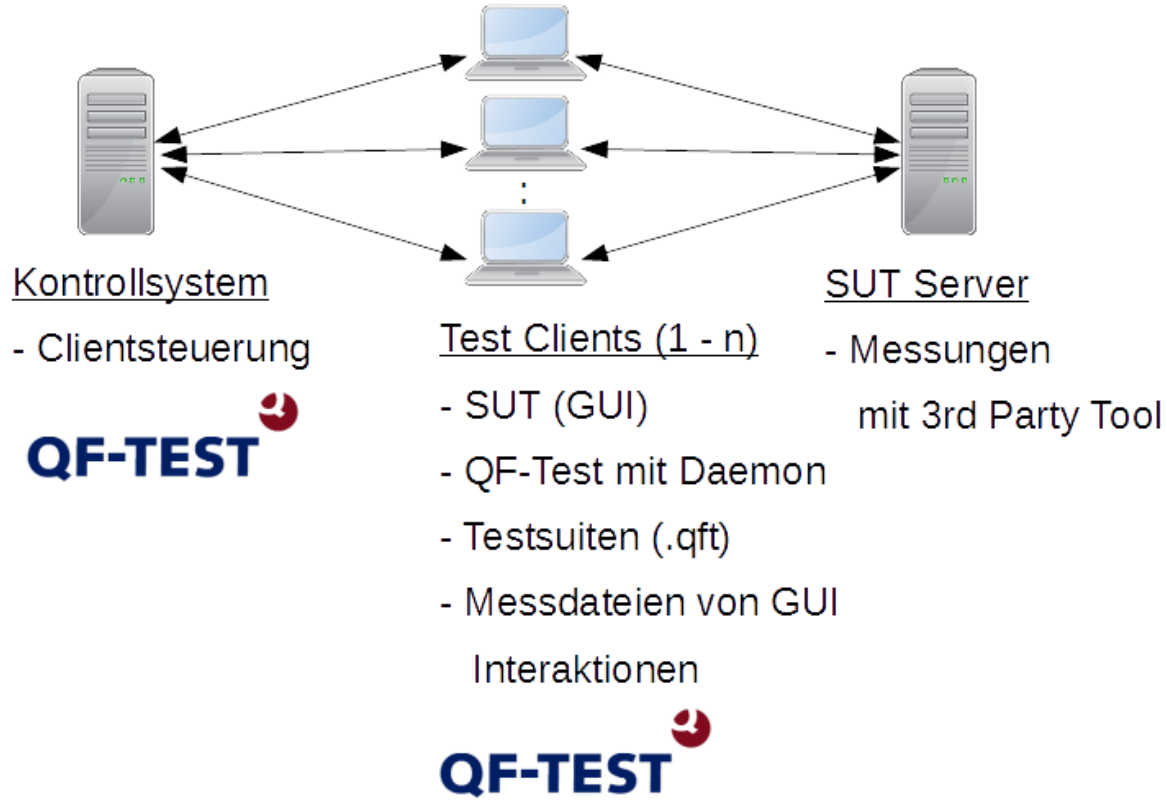


Abbildung 33.1: Lasttest Szenario

33.2 Lasttests mit QF-Test

Dieses Thema ist ein sehr anspruchsvolles. Daher liefert QF-Test eine Demolösung mit, die Sie auch als Ausgangspunkt für Ihr Lasttestprojekt verwenden können. Diese Demolösung finden Sie im Verzeichnis `qftest-9.0.0/demo/loadtesting/`. In diesem Verzeichnis sind folgende Dateien enthalten:

Datei	Zweck
Systems.xlsx	In dieser Excel-Datei können Sie konfigurieren, auf welchen Testsystemen der Testlauf ausgeführt werden soll. Des weiteren können hier auch noch globale Variablen für den Testlauf mitgegeben werden.
carconfig_Loadtesting.qft	Diese Testsuite enthält die GUI Tests, die auf den Testsystemen ausgeführt werden sollen.
daemonController_twoPhases.qft	Diese Testsuite stellt die Controller-Suite für den Testlauf dar. Hiermit starten und koordinieren Sie den Testlauf auf unterschiedlichen Rechnern.
checkForRunningDaemons.qft	Diese Testsuite beinhaltet Testfälle zur Überprüfung laufender Daemon Prozesse auf den einzelnen Testsystemen.

Tabelle 33.1: Inhalt des loadtesting Verzeichnisses

Die oben aufgeführten Testsuiten und Dateien können in einem Lasttest Projekt eingesetzt werden, welches mehrere Testsysteme miteinbezieht. Bitte achten Sie darauf, dass Sie den Ordner in ein projektspezifisches Verzeichnis kopieren und diese dort modifizieren. Die folgende Abbildung zeigt eine exemplarische Aufteilung.

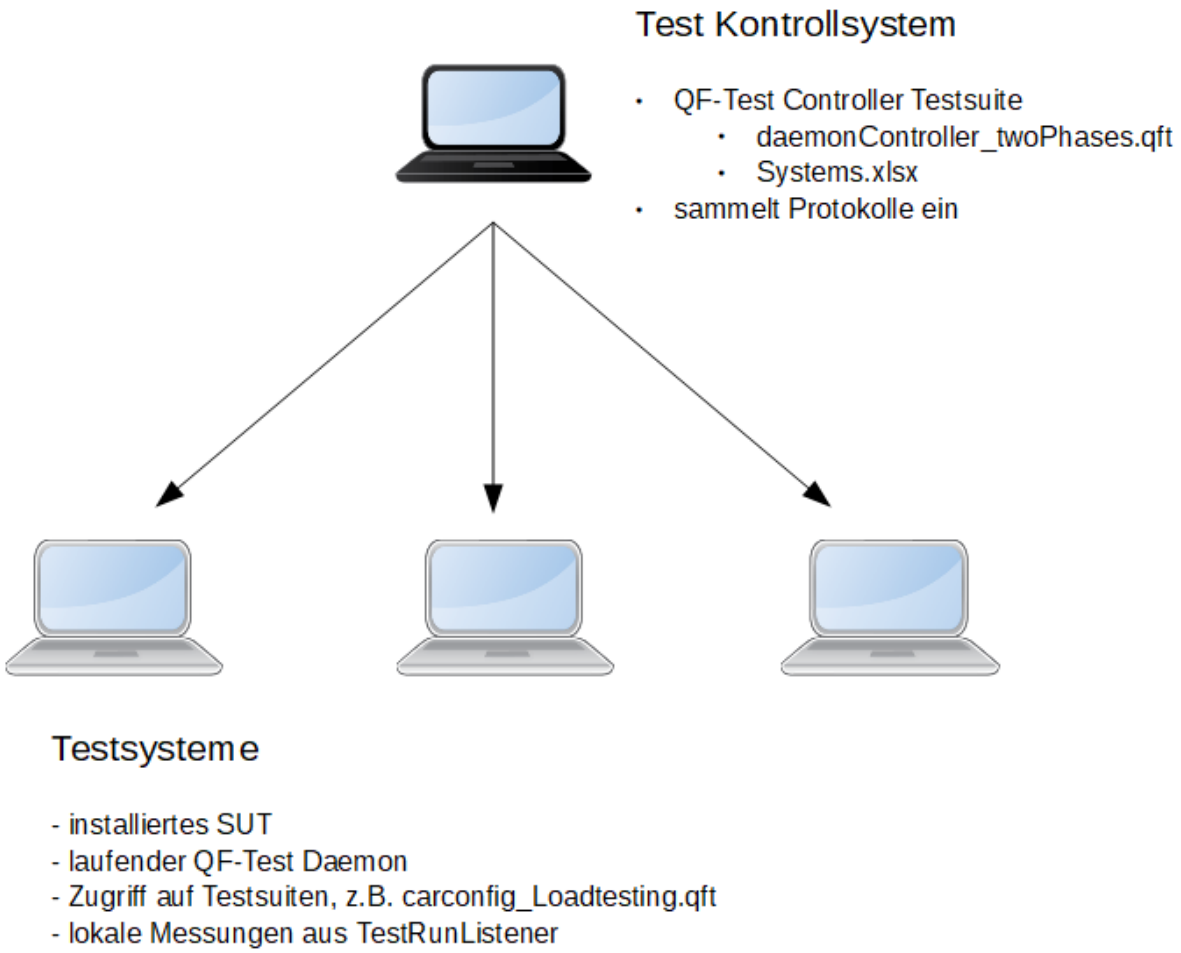


Abbildung 33.2: Übersicht Lasttest Umgebung

Die mitgelieferte Beispieltestsuite für die Steuerung des Testlaufes sieht wie folgt aus:

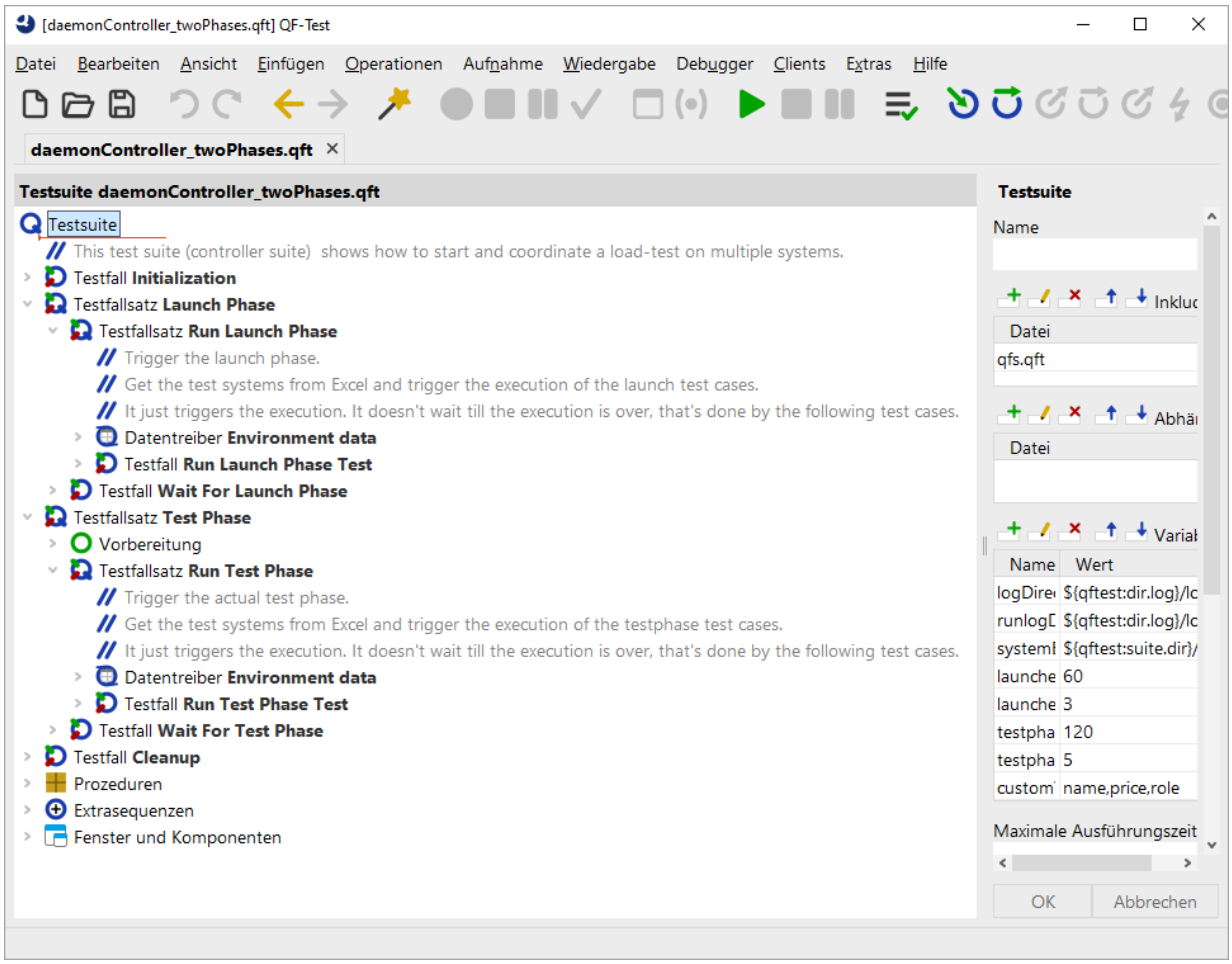


Abbildung 33.3: Die Beispieltestsuite daemonController_twoPhases.qft

Um nun mit QF-Test diese Art von Tests durchzuführen, sollten Sie folgende Punkte beachten:

1. Bereitstellung der Testsysteme
2. Konzeption des Testlaufes
3. Vorbereiten der Testsysteme für den Testlauf
4. Testausführung
5. Testauswertung

In den folgenden Abschnitten finden Sie eine kurze Erläuterung und Denkanstöße für jeden dieser Punkte.

Hinweis Für Tipps zur Durchführung von parallelen Webseiten-Tests mit QF-Test lesen Sie unseren Blogartikel Parallele Webseiten-Tests mit QF-Test .

33.2.1 Bereitstellung der Testsysteme

Mit QF-Test führen Sie Lasttests über das GUI aus. GUI Tests erfordern allerdings eine aktive Benutzersession und sollten nicht parallel auf demselben Desktop laufen. Daher wird empfohlen, dass Sie ein virtuelles oder physisches System für jeden Client bereitstellen. Es ist zwar möglich mehrere GUI Tests parallel auf einem Desktop zu starten, allerdings kann dies zu sehr subtilen Problemen, z.B. Fokusproblemen, führen und wird nur in Ausnahmefällen empfohlen.

Auf jedem System muss vorab QF-Test installiert werden. Die benötigten Testsuiten, sowie die QF-Test Konfiguration und eventuell benötigte Testdatendateien müssen ebenfalls vorhanden sein. Dies können Sie entweder per lokaler Kopien oder durch Einrichtung eines gemeinsames Netzlaufwerk bewerkstelligen. Jedes Testsystem benötigt des weiteren zumindest eine Runtime-Lizenz, um die Tests ausführen zu können. Diese Runtime-Lizenzen können auch für einen begrenzten Zeitraum gemietet werden.

33.2.2 Konzeption des Testlaufes

Im einfachsten Fall laufen auf allen involvierten Testsystemen die selben Tests. Allerdings möchte man in vielen Lasttestprojekten nicht nur einen Satz von GUI Tests über mehrere Clients verteilen, sondern man versucht die Clients in mehrere Gruppen, meistens Rollen genannt, aufzuteilen. Diese Gruppen sollen die Benutzergruppen der Software widerspiegeln. Hier kann es z.B. eine Gruppe geben, die Tests eines normalen Benutzers simuliert und eine zweite Gruppe, die nur zu bestimmten Zeiten administrative Aufgaben erledigt.

Neben der Einteilung in Rollen werden Testläufe für Lasttests auch oft in unterschiedliche Phasen eingeteilt. Eine Phase steht hierbei für einen bestimmen Schwerpunkt, z.B. Zugriff von 100 Benutzern. Eine Beispieleinteilung eines Projektes in vier Phasen könnte wie folgt aussehen. Die erste Phase ist die ”Start” Phase, in der das SUT auf allen Testsystemen gestartet wird und initiale Aktionen, wie das Anmelden des Benutzers ausgeführt werden. In der zweiten Phase führen Sie nun Tests für 50 Clients aus, in der dritten Phase werden dieselben Tests für 100 Clients ausgeführt, in der vierten Phase wieder für 50 Clients. Diese Art der Skalierung kann auch als Ramp-Up Phasen (schrittweises Aufbauen von Last) und Ramp-Down Phasen (schrittweise Entlastung) bezeichnet werden.

Eine derartige Aufteilung in Phasen mit schrittweiser Steigerung der Belastung der Software hat den Vorteil, dass Sie sich an die Belastbarkeit Ihrer Software heran tasten können und nicht sofort von 0 auf 100 einsteigen, sondern immer noch eine Aussage

treffen können, dass z.B. in Phase 1 alles noch in Ordnung war und erst ab der zweiten Phase es Probleme gab.

Diese Aufteilung in unterschiedliche Phasen kann auch bei Tests, die sich auf Rollen fokussieren, Sinn ergeben. In einigen Fällen kann ein gleichzeitiger Start von allen teilnehmenden Testsystemen zu einer Überlast der Umgebung führen und somit würde sich bereits anbieten, zumindest den Start der Anwendung auf jeden Fall in eine separate Phase auszulagern.

Aus Übersichtlichkeitsgründen sollten Sie je Rolle eine Testsuite erstellen, die die entsprechenden Testfälle beinhaltet bzw. aufruft.

Umsetzung in der Beispieltestsuite:
In der mitgelieferten Controller-Suite `daemonController_twoPhases.qft` finden Sie ein Beispiel für die Aufteilung in zwei Phasen. Die erste Phase ist die `Launch Phase` Phase, also das Starten der Anwendung. Die zweite Phase `Test Phase` stellt die Testphase dar. Die erforderlichen Testsuiten können Sie im entsprechenden `Run...Phase` Test-Knoten der jeweiligen Phase konfigurieren.

Das mitgelieferte Beispiel stellt also ein Szenario dar, welches sich auf unterschiedliche Rollen statt auf unterschiedliche Phasen konzentriert. Falls Sie eine dritte Phase einfügen möchten, können Sie den Test-Knoten, der `Test Phase` kopieren, einfügen und entsprechend umbenennen.

33.2.3 Vorbereiten der Testsysteme für den Testlauf

Bevor Sie nun einen Testlauf starten können, müssen Sie auf allen Testsystem den QF-Test Daemon starten. Dieser Daemon benötigt einen freien Netzwerkport. Es empfiehlt sich aus Effizienzgründen, auf allen Rechnern den selben Port zu verwenden, z.B. 5555.

Der Daemon kann nun wie folgt gestartet werden:

```
qftest -batch -daemon -daemonport 5555
```

Beispiel 33.1: Starten des QF-Test Daemon

Bitte beachten Sie, dass der Daemon bereits in einer aktiven Benutzersession gestartet werden muss. Dies können Sie z.B. mittels Aufgabenplaner erreichen. Weitere Details zur Testausführung mittels Daemon finden Sie im Abschnitt 25.2(346). Sie finden im Kapitel Aufsetzen von Testsystemen(474) nützliche Tipps und Tricks für die Einrichtung des Daemon Prozesses. Der technische Hintergrund ist in FAQ 14 beschrieben.

Die Überprüfung, ob die Daemonen auf den Testrechnern laufen, können Sie entweder mit einzelnen Ping-Kommandos des Daemons oder mittels Ausführung der mitgelieferten Testsuite `checkForRunningDaemons.qft` durchführen.

```
qftest -batch -calldaemon -ping -daemonhost localhost -daemonport 5555
```

Beispiel 33.2: Ping des QF-Test Daemon auf localhost

Hinweis Bitte verwenden Sie auf Windows den Befehl `qftestc.exe` statt `qftest.exe` für die obigen Kommandos.

33.2.4 Testausführung

Für die Testausführung gilt es nun zu beachten, dass Sie diverse Skripte oder Software benötigen, die die unterschiedlichen QF-Test Daemone kontaktieren können, um den Testlauf zu koordinieren. Diese Steuerungsskripte können nun die QF-Test Daemon API (siehe Abschnitt 55.2[1277]) ansprechen oder die Kommandozeile (siehe Kapitel 44[971]) verwenden.

Umsetzung in der Beispieltestsuite:
Mit der mitgelieferten Beispieltestsuite `daemonController_twoPhases.qft` sind Sie in der Lage ein Lasttestszenario auszuführen und am Ende die Protokolle wieder einzusammeln. Allerdings müssen Sie vorher noch konfigurieren, welche Systeme beim Testlauf dabei sind. Diese Systeme können Sie in der Excel-Datei `qftest-9.0.0/demo/loadtesting/Systems.xlsx` konfigurieren. Außerdem ist es hier auch möglich die entsprechende Rolle zu konfigurieren, wie im Abschnitt 33.2.2[443] beschrieben.

Sind die Systeme korrekt konfiguriert, können Sie nun den Testlauf starten, indem Sie die gesamte Testsuite ausführen.

Zusätzlich zur reinen Ausführung können Sie mit QF-Test noch weitere Anforderungen abdecken. Folgende Anforderungen werden auch in den mitgelieferten Beispieltestsuiten erläutert:

1. Synchronisierung von mehreren Systemen, siehe Abschnitt 33.3.1[446].
2. Messung von End-to-End Zeiten, siehe Abschnitt 33.3.2[447].

33.2.5 Testauswertung

Die Auswertung von Lasttests kann auf Grund der schieren Masse an Daten eine Herausforderung darstellen. Sie können QF-Test Protokolle mit QF-Test oder als HTML-Report auswerten. Messungen auf Server- bzw. Infrastrukturseite können mit Spezialtools analysiert werden. Natürlich stehen Ihnen noch die Logdateien Ihrer Server zur Verfügung, welche auch mit speziellen Tools ausgewertet werden können.

Es besteht auch die Möglichkeit während der Ausführung mit QF-Test eigene Protokolle für Messungen anzufertigen, mehr hierzu finden Sie im Abschnitt 33.3.2(447).

33.3 Spezielles zur Testausführung

33.3.1 Synchronisierung

Um reproduzierbare Ergebnisse zu erhalten, kann es nötig sein, die Tests auf den verschiedenen Systemen zu koordinieren, entweder um alle Clients möglichst gleichzeitig den Server ansprechen zu lassen, oder um genau das zu verhindern. Des weiteren könnte es eine Rolle (siehe vorigen Abschnitt) erfordern, dass alle Systeme dieser Rolle auf einen bestimmten Zeitpunkt warten, um eine Aktion auszuführen.

Mehrere parallele Testläufe können mit einem Server-Skript(717) Knoten synchronisiert werden. In diesem Skript muss folgender Aufruf enthalten sein:

```
rc.syncThreads("identifier", timeout, remote=3)
```

Der `identifier` ist ein Name für den Synchronisationspunkt, `timeout` ist die maximale Zeit in Millisekunden, nach der alle Systeme diesen Synchronisationspunkt erreicht haben müssen und `remote` gibt an wie viele Rechner aufeinander warten sollen und diesen Synchronisationspunkt auch erreichen müssen.

Wird das Timeout überschritten, ohne dass die gewünschte Anzahl von Systemen den Synchronisationspunkt erreicht, wird eine `TestException`(958) geworfen. Um stattdessen nur einen Fehler auszugeben und den Testlauf fortzusetzen, kann dem Parameter `throw` der Wert 0 mitgegeben werden (Standard ist 1) oder das Server-Skript(717) in einem Try Knoten verpackt werden.

```
rc.syncThreads("case1", 120000, remote=3, throw=0)
```

Eine Beispielsynchronisierung findet auch in der mitgelieferten Testsuite `carconfig_Loadtesting.qft` statt.

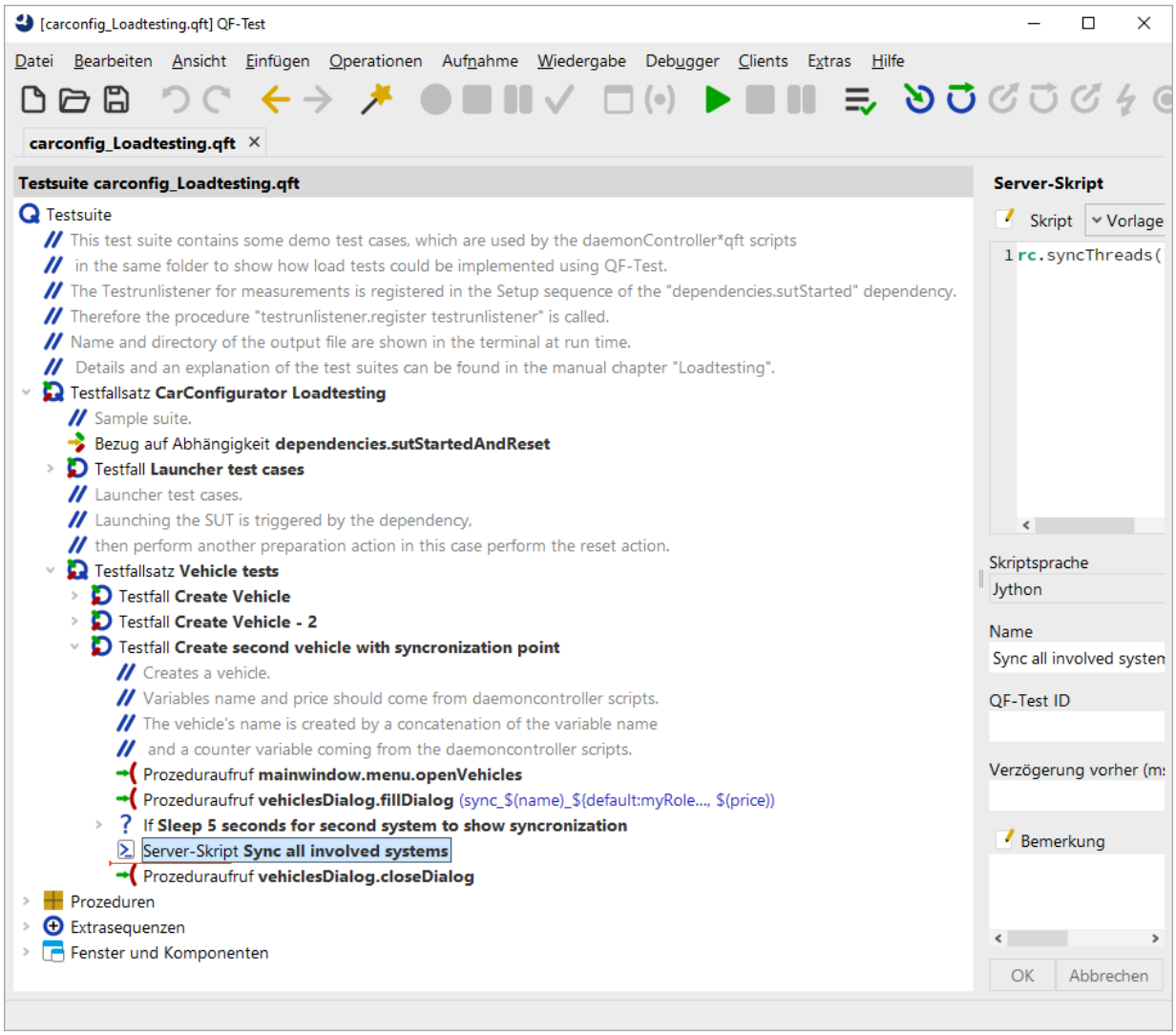


Abbildung 33.4: Der Aufruf von rc.syncThreads in der Beispieltestsuite

33.3.2 Messen von End-to-End Zeiten

Eine häufige Anforderung an GUI Tests ist das Messen von End-to-End Zeiten, sowie deren schnelle Auswertung.

Hierfür schreibt QF-Test diese Zeiten zwar bereits ins Protokoll, allerdings müsste man nun nachträglich diese Formate parsen. Damit Sie diese Zeiten einfacher erhalten, kann auch ein `TestRunListener` genutzt werden, der von Ihnen gekennzeichnete Messpunkte aus der Testsuite in ein separates Messprotokoll schreibt.

Damit nun die richtigen Messungen stattfinden, sollten Sie die Messpunkte mit einem

Schlüsselwort in der Bemerkung des Knotens markieren. Im mitgelieferten Beispiel ist dies `@transaction`. Falls Sie ein anderes Schlüsselwort verwenden wollen, können Sie hierzu den Code des TestRunListeners anpassen.

Die Messungen werden im mitgelieferten Beispiel in eine einfache CSV-Datei geschrieben, damit diese später weiterverarbeitet werden können und der Testlauf nicht zu stark ausgebremst wird. Die Aufbereitung der Daten in einer Excel-Datei oder Datenbank sollte aus Performancegründen erst nach der eigentlichen Testausführung erfolgen.

Details zum TestRunListener finden Sie im Abschnitt 54.6(1223). Sie finden eine Beispielimplementierung in der mitgelieferten Demotestsuite `carconfig_Loadtesting.qft`.

Die erzeugte CSV-Datei sieht wie folgt aus:

```
open vehicles;118;20150921145057;OK
close vehicles;84;20150921145057;OK
```

Beispiel 33.3: CSV-Datei für Zeitmessungen

In der erzeugten CSV-Datei steht der erste Wert für die Namen der Messpunkte, der zweite für die Dauer in Millisekunden, der dritte für den Ausführungszeitpunkt und der vierte, ob der Status überhaupt erfolgreich war.

33.4 Troubleshooting

Durch die Komplexität eines Lasttestprojekts kann es zu unterschiedlichen Problemfeldern kommen.

1. Es werden die falschen Testfälle ausgeführt?

   Hierzu passen Sie Variable `testsuite` in den entsprechenden Testfällen an. Es kann auch ein Testfall direkt angesprochen werden, z.B. `testsuite#testfallsatz.testfall`.

2. Der QF-Test Daemon lässt sich nicht starten.

   Ist der Port den überhaupt frei? Überprüfen Sie dies mit den entsprechenden `netstat`-Kommando. Hier ein Beispiel für den Port 5555.

   Windows
   ```
   netstat -a -p tcp -n | findstr "5555"
   ```

   Linux
   ```
   netstat -a --tcp --numeric-ports | grep 5555
   ```

3. Testsysteme sind nicht ansprechbar, obwohl der QF-Test Daemon läuft.

Überprüfen Sie, ob der QF-Test Daemon läuft und dieser von Ihrem Ausführungssystem aus erreichbar ist, siehe Abschnitt 33.2.1(443). Falls der QF-Test Daemon läuft, dann führen Sie bitte folgende Schritte durch:

(a) Lässt sich der Daemon lokal mit dem Pingkommando ansprechen, siehe Abschnitt 33.2.3(444)?

(b) Stellen Sie sicher, dass der Daemon bzw. der dazugehörige Javaprozess nicht von der Firewall blockiert wird.

(c) Evtl. gibt es Probleme mit der Auflösung Ihres Rechnernamen. Versuchen Sie bitte den Daemon zusätzlich mit dem Parameter `-serverhost localhost` bzw. `-serverhost IP-Adresse` bzw. `-serverhost <Rechnername>` zu starten. Falls Sie mit der IP-Adresse starten, sprechen Sie diesen Testrechner auch mit der IP-Adresse an, sonst mit dem Rechnernamen.

33.5 Web-Lasttests ohne sichtbare Browser-Fenster

Für den Lasttest von Web-Anwendung ist es möglich, Browser im ”headless” Modus zu betreiben. Dies hat den Vorteil, dass ein Headless Browser kein eigenes GUI und damit keine eigene Anwendersitzung benötigt. Der GUI-Test unterliegt aber im Vergleich zu ”normalen” Browsertests einigen Einschränkungen:

- Harte und semi-harte Mausklicks sowie Drag-And-Drop Operationen werden über eine Browser-Schnittstelle simuliert und können daher ein abweichendes Verhalten haben als bei ”normalen” Browsertests.
- Screenshots können erstellt werden, haben aber eventuell nicht hundertprozentig die gleiche Optik wie im normalen Browser, da kein GUI vorhanden ist, das ”abfotografiert” werden kann.
- Außerdem muss es die Anwendung auch selbst erlauben, dass sie in einer Anwendersitzung auf mehreren Browser-Instanzen ausgeführt wird.

Weitere Informationen zu Browsern ohne sichtbares Fenster finden Sie in Abschnitt 14.7(232).

Kapitel 34

Ausführung manueller Tests mit QF-Test

3.0+

34.1 Einführung

QF-Test ist in erster Linie ein Werkzeug, das GUI Tests erstellt und automatisiert auszuführt. Jedoch ist es nicht immer möglich - oder wirtschaftlich - die Tests vollständig zu automatisieren, deshalb gibt es in neben dem automatisierten Testen auch manuelle Testfälle, die auszuführen sind. Eine der größten Herausforderungen in Testprojekten ist die gemeinsame Aufbereitung der automatisierten und manuellen Testergebnisse. Eine solche gemeinsame Aufbereitung ist sehr nützlich, um einen Gesamtüberblick über die Testausführung zu erhalten. QF-Test bietet nun die Möglichkeit Ergebnisse von manuellen und auch von automatisierten Tests gemeinsam anzuzeigen.

Die Schritte eines manuellen Tests müssen hierfür an einer bestimmten Stelle definiert werden. Der `ManualTestRunner` von QF-Test liefert hierfür eine Excel-Datei. Diese Testsuite sowie eine beispielhafte Excel-Datei finden Sie im Verzeichnis `demo/manualtester` im QF-Test Installationsverzeichnis. Der Testdesigner muss nun die einzelnen Schritte des Tests in der Excel-Datei definieren, inklusive des erwarteten Ergebnisses. Nach Ausführung des manuellen Tests von QF-Test aus, liefert QF-Test sowohl das Protokoll als auch HTML-Report und zusätzlich noch eine neue Excel-Datei, die die entsprechenden Ergebnisse beinhaltet. Für eine genauere Beschreibung siehe Abschnitt 34.2(451).

Der Dialog zur Testausführung, der ManualStepDialog, sieht wie folgt aus:

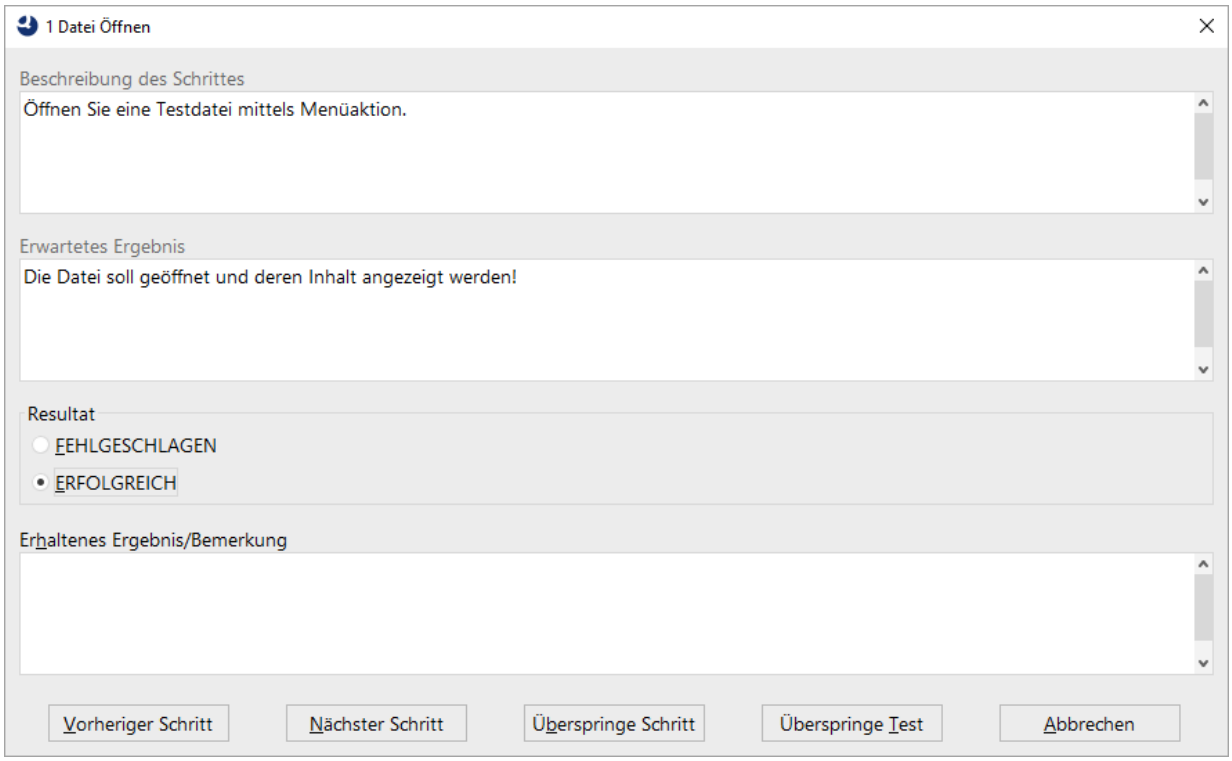


Abbildung 34.1: Beispiel für einen ManualStepDialog

Der Titel des Dialoges beinhaltet den Namen des Testfalles. Die ersten zwei Textfelder enthalten die Beschreibung des manuellen Schrittes sowie das erwartete Ergebnis. Nach Ausführung des Testschrittes muss der Tester nun angeben, ob dieser erfolgreich war oder nicht. Falls der Testschritt nicht erfolgreich gewesen sein sollte, so muss der Tester ein erhaltenes Ergebnis dokumentieren. Sie können diesen Dialog auch für Ihre eigenen Ansätze verwenden, siehe Abschnitt 57.1[(1304)].

34.2 Schritt-für-Schritt Anleitung

Bitte führen Sie folgende Schritte auf dem System aus, wo Sie einen manuellen Test von QF-Test aus starten:

- Kopieren Sie die Excel-Datei von `qftest-9.0.0/demo/manualtester/SampleTestDescription.xlsx` in ein projektspezifisches Verzeichnis. Wie empfehlen auf jedem Testsystem den selben Pfad zu verwenden, vielleicht ein Netzlaufwerk.
- Kopieren Sie auch die Ausführungstestsuite von

`qftest-9.0.0/demo/manualtester/ManualTestRunner.qft` in ein projektspezifisches Verzeichnis. Womöglich sollten Sie diese auch umbenennen.

- Öffnen Sie die Excel-Datei und definieren Sie die Testschritte.
- Nachdem Sie die Änderungen der Excel-Datei abgespeichert haben, müssen Sie die Ausführungstestsuite öffnen und die globale Variable `testFile` anpassen, um die projektspezifische Excel-Datei einzulesen.
- Schalten Sie den QF-Test Debugger aus. Dieser würde Sie nur während der Ausführung der Tests behindern.
- Selektieren Sie nun den Testsuite Knoten und starten Sie die Testsuite mittels des ”Wiedergabe starten” Knopfes.
- QF-Test wird nun die Daten aus der Excel-Datei einlesen und einen Dialog mit dem ersten Testschritt anzeigen.
- Geben Sie das erhaltene Ergebnis ein und fahren Sie mit der Ausführung der nächsten Schritte fort.
- Am Ende der Testausführung erzeugt QF-Test eine neue Excel-Datei mit den erhaltenen Ergebnissen. Sie können von QF-Test aus nun auch das QF-Test Protokoll speichern oder den HTML-Report erzeugen.

Bitte beachten Sie auch die Kommentare in der Ausführungstestsuite und in der Excel-Datei. Sie können dieses Konzept natürlich auch an Ihre eigenen Anforderungen anpassen und sogar nur bestimmte Tests starten.

34.3 Aufbau der Excel-Datei

Die Excel-Datei hat eine bestimmte Struktur, um einen manuellen Testfall so flexibel wie möglich zu beschreiben. Hier folgt eine genauere Beschreibung der einzelnen Spalten:

Spalte	Beschreibung
TestCase	Ein eindeutiger Bezeichner für jeden Testfall. Wird kein Bezeichner definiert, so ist der Schritt Teil des vorherigen Testfalles.
Type of Test	Eine optionale Beschreibung des Types des Testfalles, z.B. funktionaler Test oder Benutzerfreundlichkeits-Test.
Comment	Ein individueller Kommentar zu jedem Testfall. Dieser Kommentar wird als Anmerkung im QF-Test Protokoll angezeigt.
Short Description	Eine Kurzbeschreibung des Testfalles.
Step Description	Eine genauere Beschreibung des manuellen Testschrittes.
Expected Result	Eine Beschreibung des erwarteten Ergebnisses.

Tabelle 34.1: Beschreibung der Excel-Datei für die Testdefinition

Die Excel-Datei, welche dann die Ergebnisse der manuellen Testausführung beinhaltet, hat noch zwei zusätzliche Spalten:

Spalte	Beschreibung
Received Result	Das Ergebnis, welches der Tester während der Testausführung erhalten hat. Dieses Ergebnis muss gesetzt sein, wenn der Testfall fehl schlägt.
State	Der Status des Testfalles: PASSED, FAILED, CANCELED bzw. SKIPPED.

Tabelle 34.2: Beschreibung der Excel-Datei für die Testergebnisse

34.4 Die Ausführungstestsuite

Die Ausführungstestsuite `ManualTestRunner.qft` beinhaltet ein paar globale Variablen, um die Ausführung zu steuern. Variablen, die hier nicht aufgeführt werden, werden von der Testsuite intern genutzt und sollten nicht verändert werden. Hier folgt eine Beschreibung dieser Variablen:

Globale Variable	Beschreibung
testFile	Der Pfad zur Excel-Datei mit den Testschritten.
testSheet	Der Tabellenblatt der Excel-Datei, welche die Testschritte beinhaltet.
resultSheet	Der Name des Tabellenblattes, in welches die Ergebnisse geschrieben werden sollen.
tests	Eine Liste von Tests, die ausgeführt werden sollen. Wenn Sie diese Variable leer lassen, werden alle Tests aus der Excel-Datei ausgeführt. Falls Sie z.B. nur Test 5 und 6 ausführen wollen, dann können Sie diese Variable auf `5,6` oder `5-6` setzen. Es ist sogar möglich, Bereiche zu definieren - wie z.B. `1,3-5,7` um die Tests 1, 3, 4, 5 and 7, zu starten.
defaultState	Der vorbelegte Status, wenn der Dialog erscheint. Er kann entweder auf PASSED oder FAILED gesetzt werden. Jeder andere Status wird auf FAILED konvertiert.
testCaseColumn	Die Überschrift der Spalte für die Testfallnummer.
commentColumn	Die Überschrift der Spalte für den Kommentar.
shortDescColumn	Die Überschrift der Spalte für Kurzbeschreibung des Testschrittes.
stepDescColumn	Die Überschrift der Spalte für die detaillierte Beschreibung des Testschrittes.
expResultColumn	Die Überschrift der Spalte für das erwartete Ergebnis.
recResultColumn	Die Überschrift der Spalte für das erhaltene Ergebnis.
stateColumn	Die Überschrift der Spalte für den Status des Testschrittes.

Tabelle 34.3: Beschreibung der globalen Variablen in der ManualTestRunner Testsuite

34.5 Die möglichen Zustände

Ausgeführte Testschritte können auf folgende Zustände gesetzt werden:

Resultat	Beschreibung
PASSED	Der Testschritt war erfolgreich.
FAILED	Der Testschritt war nicht erfolgreich.
CANCELED	Der Testschritt wurde abgebrochen.
SKIPPED	Der Testschritt wurde übersprungen.

Tabelle 34.4: Mögliche Zustände der manuellen Testausführung

Teil II

Best Practices

Kapitel 35

Einführung

Dieser Teil des Handbuches beschreibt Best Practices basierend auf Erfahrungen aus zahlreichen GUI Testprojekten und Kundenrückmeldungen. Die beschriebenen Konzepte können Sie darin unterstützen, die beste Strategie für den Einsatz von QF-Test in Ihren Projekten zu finden.

Hinweis

QF-Test ist ein sehr generisches Werkzeug. Die hier enthaltenen Tipps und Anregungen können nur als Empfehlungen unsererseits verstanden werden. Wir hoffen dadurch Ihnen dabei zu helfen, effizient und erfolgreich mit QF-Test zu arbeiten. Natürlich führen mehrere Wege nach Rom und deshalb müssen Sie Ihren eigenen Weg finden, welcher für Sie und Ihre Anforderungen am besten geeignet ist.

Kapitel 36

Wie beginnt man in einem Testprojekt?

Dieses Kapitel beschreibt die wichtigsten Aspekte, welche berücksichtigt werden sollten, **bevor** Sie QF-Test großflächig in einem Testprojekt einsetzen. Es wirft Fragen auf und versucht allgemeine Antworten mit Referenzen auf detaillierte Beschreibungen zu geben.

Ziel des Kapitels ist es Ihnen einige Hinweise zu geben, worauf Sie achten sollten, um Ihre GUI-Tests verlässlich, stabil, einfach wiederholbar und vor allem wartbar zu gestalten.

36.1 Infrastruktur und Testumgebung

Bevor Sie beginnen automatisierte Tests zu erstellen und auszuführen, sollten Sie sich ein paar Gedanken über die zu verwendende Testumgebung machen. Damit Sie Tests verlässlich und wiederholbar abspielen können, müssen Sie bedenken, dass Sie hierfür das SUT in einen definierten Zustand bringen sollten, inklusive dem Backend, z.B. einen Server und/oder eine Datenbank. Wenn Sie sich dessen nicht bewusst sind, kann es unter Umständen ziemlich schwierig werden, Tests wiederholbar abzuspielen oder einfach nur die Testresultate zu analysieren. Außerdem kann die Wartung der Tests zu einem Albtraum werden.

Bitte bedenken Sie zusätzlich noch folgende Themen:

1. Wie sieht der Initialzustand des SUT aus?
    - Welcher Benutzer wird verwendet um die Tests in der SUT abzuspielen? Die meisten Projekte arbeiten mit dedizierten Testbenutzern. Ein anderer Ansatz ist, dass jeder Tester mit seinen eigenen Testbenutzer arbeitet.
    - Welche Spracheinstellung des SUT wird hauptsächlich verwendet? Ist es wirklich notwendig eine volle Sprachabdeckung über alle unterstützten Spra-

chen zu erreichen? Oder genügt es, die gesamten Testfälle auf einer speziellen Sprache auszuführen und nur ein paar ausgesuchte für die Lokalisierungstests zu verwenden? In den meisten Fällen deckt das Wiederholen der Testfälle in unterschiedlichen Sprachen dieselbe Funktionalität ab und Sie erlangen dadurch keine neuen Informationen über das SUT. Wenn Sie sich trotzdem entschließen mit mehreren Spracheinstellungen zu testen, könnte dies die Wiedererkennung der grafischen Komponenten beeinflussen, siehe Abschnitt 5.3(54).

2. Wie sieht der Initialzustand der Datenbank aus?

   - Können Sie mit einer separaten Testdatenbank arbeiten oder benutzen Sie eine Produktionsdatenbank? Testdatenbanken können geplante Testdaten beinhalten, wohingegen Produktionsdatenbanken wirkliche Daten aus der realen Umgebung beinhalten. Sind diese Daten vorhersagbar und verlässlich? Was passiert, wenn Ihre Daten durcheinander kommen oder gar zerstört werden? Sie sollten in jedem Fall keine automatisierten Tests auf der Produktionsumgebung ausführen.
   - Können Sie Ihre Testumgebung nach einem Testlauf wieder auf den Ausgangszustand zurücksetzen, um den Testlauf zu wiederholen? Ist es möglich, Änderungen in der Datenbank zurückzunehmen oder benötigt jeder neue Testlauf neue Testdaten?
   - Wie können Testdaten gelesen bzw. geschrieben werden? Können Sie hierfür Standard SQL-Skripte verwenden oder gibt es Bibliotheken von der Entwicklung? Einige Projekte setzen sogar die Datenbank vor jedem Testlauf komplett neu auf, weil sie weder Testdaten wiederverwenden noch die Datenbank korrekt säubern können.

3. Wollen Sie QF-Test mit anderen Werkzeugen, z.B. Build- oder Testverwaltungswerkzeuge, integrieren?

   - Wie integriert man QF-Test in eine Testverwaltung? Wenn Sie bereits geplante Testschritte wiederverwenden können, dann vermeiden Sie redundante Arbeit in der Testplanungsphase. Einige Standardintegrationen sind unter Kapitel 28(373) beschrieben.
   - Sollen Tests von einem Buildtool gestartet werden? Nachdem Sie Tests erstellt haben, können Sie diese unbeaufsichtigt ausführen und diesen Testlauf von einem Buildtool, z.B. Ant oder CruiseControl, anstoßen. Für Details über Testausführung, siehe Kapitel 25(340).
   - Sollen Testresultate in ein Reporting- oder Testverwaltungssystem hochgeladen werden oder reicht es den erzeugten HTML-Report und die Protokolle auf einen zentralen HTTP-Server zu legen?

4. Wer wird mit QF-Test arbeiten?

    - Arbeiten nur ein oder zwei Tester mit QF-Test oder werden auch mehrere Entwickler und Fachtester in die Testentwicklung miteinbezogen? Im Abschnitt Abschnitt 37.5(467) finden Sie einige Tipps für das Arbeiten im Team.
    - Wie sind die Kenntnisse der Tester? Sie sollten mindestens eine dedizierte Person mit guten QF-Test Kenntnissen im Team haben, welche auch in der Lage ist, Skripte zu implementieren und Softwareentwicklungsprinzipien versteht.

Natürlich werden in Ihrem Projekt noch weitere Punkte auftauchen. Denken Sie einmal darüber nach.

36.2 Speicherorte

Sie sollten sich überlegen, wohin Sie bestimmte Dateien legen bzw. installieren wollen:

1. Wo soll QF-Test installiert werden? QF-Test kann lokal auf jedem Rechner installiert werden. Dieses Vorgehen zwingt Sie allerdings wiederum, neue Versionen auf jedem Rechner einzeln zu installieren. Alternativ können Sie QF-Test auf einem Netzlaufwerk installieren, wenn Ihr Netzwerk verlässlich funktionieren sollte. Details hierzu finden Sie unter Abschnitt 36.2.1(460).

2. Wo soll die Konfigurationsdatei `qftest.cfg` abgelegt werden? Diese Datei beinhaltet u.a. wie QF-Test die grafischen Komponenten wiedererkennt oder was im Protokoll gespeichert werden soll. Diese Einstellungen müssen für jeden QF-Test Benutzer dieselben sein, sonst können keine Tests im Team geteilt werden. Damit Sie dieselbe Datei verwenden, können Sie QF-Test entweder zentral auf einem Netzlaufwerk installieren oder Sie geben den Pfad zur Konfigurationsdatei als Kommandozeilenparameter beim Start von QF-Test mit. Wenn Sie mit einer Datei auf einem Netzlaufwerk arbeiten, dann stellen Sie sicher, dass diese Datei schreibgeschützt ist und nur bestimmte Benutzer diese ändern können. Details finden Sie unter Abschnitt 1.6(12).

3. Wo soll die Lizenzdatei `license` liegen? Sie sollten die Lizenzdatei an einer zentralen Stelle ablegen, damit Sie die Lizenz nur einmal aktualisieren müssen, wenn Sie ein Update erhalten. Hier können Sie ebenfalls entweder an eine zentrale Netzwerkinstallation von QF-Test denken oder einen Kommandozeilenparameter beim Start von QF-Test verwenden. Details finden Sie unter Abschnitt 1.5(11).

4. Wo sollen Testsuiten gespeichert werden? Der beste Speicherort für Testsuiten ist ein Versionsmanagementsystem, womit Sie Änderungen verfolgen und auf jede

Version der Datei zugreifen können. Wenn dies nicht möglich sein sollte, dann sollten die Dateien auf einem zentralen Netzlaufwerk installiert werden.

5. Wo sollen die Testdatendateien gespeichert werden? Testdatendateien gehören immer zu Testsuiten und sollten daher nahe an diesen abgelegt werden, d.h. im selben Versionsmanagementsystem oder Netzlaufwerk.

6. Wo sollen die HTML-Reports und Protokolle abgelegt werden? Diese Dateien sollten an einer zentralen Stelle liegen, auf die jeder Tester zugreifen und die Testresultate auswerten kann. Die meisten Kunden legen diese auf einen zentralen HTTP Server oder einen Netzlaufwerk ab.

36.2.1 Netzwerkinstallation

Wenn Sie planen QF-Test auf einem Netzlaufwerk zu installieren, müssen Sie einige Dinge beachten.

Der kritische Punkt ist die Konfigurationsdatei `qftest.cfg`. Es ist empfehlenswert (und notwendig), dass alle Tester dieselben Einstellungen verwenden, besonders für die Wiedererkennung. Dies erreichen Sie durch Sharen dieser Datei. Jedoch sollten Sie darauf achten, dass sie schreibgeschützt ist, damit ein Benutzer nicht unabsichtlich die Einstellungen verändert. Da QF-Test vorgenommene Änderungen so nicht beim Beenden speichern kann. Jede gewollte Änderung sollte durchgeführt werden, in dem Sie die Datei explizit mit Schreibrechten versehen, dann QF-Test beenden, und danach die Schreibrechte wieder entziehen. Eine andere Möglichkeit ist, dass jeder Benutzer, seine eigene Konfigurationsdatei benutzt, z.B. mittels dem Kommandozeilenparameter `-systemcfg <Datei>`[(992)], aber das ist nicht empfehlenswert.

Die laufenden QF-Test Instanzen teilen sich auch das Logverzeichnis (internes Logging, das kein Problem darstellt) und den Jython Package Cache, welcher gelegentlich Probleme verursacht. In diesem Fall kann QF-Test den Jythoninterpreter nicht mehr initialisieren. Das passiert sehr selten und kann mit Säuberung (nicht kompletten Löschens) des Jython Cache Verzeichnisses behoben werden.

Auf Windows sollte jeder Benutzer die Datei `setup.exe` ausführen, damit die aktuelle QF-Test Version, die sich im installierten qftest-x.y.z. Verzeichnis befindet, mit den entsprechenden Windowseinstellungen in der Registry verknüpft wird.

In sehr seltenen Fällen kann das Überschreiben von Jar-Dateien von QF-Test beim Einspielen eines QF-Test Patches, bereits laufende Instanzen auf Windows zum Absturz bringen.

36.3 Wiedererkennung von Komponenten

Der wichtigste Aspekt eines GUI Testtools ist eine stabile und verlässliche Wiedererkennung der grafischen Komponenten. In diesem Bereich ist QF-Test sehr flexibel und bietet unterschiedliche Konfigurationsmöglichkeiten. In den meisten Fällen reicht die Standardkonfiguration der Komponentenerkennung aus, allerdings müssen Sie diese manchmal anpassen.

Wenn Sie die Komponentenerkennungseinstellungen nach Erstellen mehrerer Testfälle ändern, laufen Sie Gefahr, dass diese Testfälle nicht mehr lauffähig sind. Deshalb sollten Sie so früh wie möglich versuchen eine angemessene Erkennungsstrategie für Ihr Testprojekt zu finden. Sie sollten sich diese Zeit in jedem Fall nehmen und diesen Bereich wirklich genau ansehen, bevor Sie viele Testfälle kreieren. Im schlimmsten Fall müssen Sie alle bereits erstellten Testfälle neu erstellen, nachdem Sie die Erkennungseinstellungen verändert haben.

Am besten erstellen Sie einige Demotestfälle und versuchen herauszufinden, wie QF-Test die Komponenten Ihres SUT erkennt. Der Wiedererkennungsmechanismus ist in den Kapiteln Kapitel 5(47) und Abschnitt 5.9(92) beschrieben. Wenn Sie die Demotestfälle, im Idealfall mit einer anderen SUT Version, ausführen und auf Wiedererkennungsprobleme stoßen, versuchen Sie folgende Punkte zu klären:

1. Gibt es genügend Synchronisationspunkte, wie Warten auf Komponente oder Check Knoten mit Wartezeiten, um die Testschritte nur dann auszuführen, wenn das SUT wirklich bereit dazu ist?
   (a) Beispiel 1: Nach Öffnen eines Fenster können Sie Aktionen nur dann darin ausführen, wenn es wirklich schon vorhanden ist -> Verwenden Sie einen Warten auf Komponente Knoten.
   (b) Beispiel 2: Nachdem Sie auf einen ”Suchen” Knopf geklickt haben, können Sie erst dann fortfahren, wenn die Suche wirklich beendet ist -> Verwenden Sie einen Check Knoten mit Wartezeit.

Ein weiterer Punkt neben Synchronisationspunkten ist die richtige Einstellung für die Komponentenerkennung. Sie sollten folgende Punkte vorab klären, um herauszufinden, welche Einstellung für Sie die geeignetste ist:

1. Werden eindeutige und stabile Namen für die Komponenten von den Entwicklern vergeben? Siehe Abschnitt 42.13(918) für Details.
2. Vielleicht reicht es aus, einen regulären Ausdruck im Merkmal Attribut der Komponente des Hauptfensters unter des Fenster und Komponenten Knotens zu setzen? Siehe Abschnitt 5.4.3(71) für Details.

3. Wenn die Entwicklung keine benutzbaren oder sogar dynamische Namen vergeben haben, dann könnten Sie dies mit einem NameResolver lösen. Siehe Abschnitt 5.3(54).

4. Müssen Sie vielleicht die QF-Test Wiedererkennungsoptionen ändern? Diese sind im Kapitel Abschnitt 5.3(54) beschrieben.

5. Ist es denkbar, generische Komponente einzusetzen? Siehe Abschnitt 5.8(91).

In einigen Fällen reicht es aus, einfach die Standardkonfiguration zu ändern. Angenommen die Entwicklung hat eindeutige und stabile Namen für die Zielkomponenten vergeben, d.h. für Buttons, Textfelder, Checkboxen etc., dann können Sie einfach die 'Gewichtung von Namen' Option von QF-Test auf 'Name übertrifft alles' setzen. Diese Einstellung teilt QF-Test mit, Änderungen in der Komponentenhierarchie zu ignorieren und nur mit der eigentlichen Zielkomponente und deren Fenster zu arbeiten.

Hinweis Sie müssen diese Option an zwei Stellen ändern. Einmal unter 'Aufnahme' -> 'Komponenten' -> 'Gewichtung von Namen' und einmal unter 'Wiedergabe' -> ' Wiedererkennung' -> 'Gewichtung von Namen'. Siehe Abschnitt 5.3(54) für mehr Informationen.

Kapitel 37

Organisation von Testsuiten

Eine der herausforderndsten Aufgaben eines Testprojektes ist, die Testsuiten über einen langen Zeitraum wartbar zu gestalten. Besonders dann, wenn sich das eine oder andere Fenster oder sogar der Bedienungsablauf signifikant ändert, sollte der Wartungsaufwand so minimal wie möglich sein.

Sie sollten auch darüber nachdenken wie man den Erstellungsaufwand von Tests minimiert; insbesondere, wenn diese eine große Menge an ähnlichen oder sogar gleichen Schritten beinhalten. Ein typischer Anwendungsfall ist das Starten des SUT oder ein Login-Prozess oder ein sehr wichtiger Basisworkflow wie das Navigieren zu einem bestimmten Punkt des SUTs.

Ein weiterer Aspekt ist, wie Testsuiten effizient organisiert werden, wenn unterschiedliche Leute im Projekt involviert sind.

3.5+ Auf jeden Fall sollten Sie Ihre Testsuiten in einem QF-Test Projekt, wie unter Kapitel 9(180) beschrieben erstellen, um einen besseren Überblick über Ihre Testsuiten und Verzeichnisse zu bekommen.

Die folgenden Abschnitte zeigen nun einige Best Practices auf, wie Sie Tests wartbar, erweiterbar und gut organisiert gestalten können.

37.1 Organisation von Tests

In Abschnitt 8.2(153) werden die Konzepte der Testfallsatz und Testfall Knoten beschrieben. Ein Testfall Knoten steht für einen dedizierten Testfall und seine Testdaten. Ein typischer Testfall kann von Use Cases, von Anforderungen oder von einer Defektbeschreibung in Ihrer Umgebung abgeleitet werden, z.B. 'Kalkulation des Preises von Fahrzeug xyz mit 10% Rabatt' in der CarConfigurator Applikation.

Testfallsatz Knoten sind Sammlungen von Testfallsätze und Testfälle. Diese Knoten kön-

nen Sie für die Strukturierung von Testfällen einsetzen, z.B. 'Tests für Preiskalkulation'.

Testschritt Knoten repräsentieren individuelle Testschritte eines Testfall Knotens, wie 'Öffne das Fenster' oder 'Prüfe Berechnung'.

Falls Sie eine externe Beschreibung des Testfalls oder andere relevante Informationen mit dem Testfall verknüpfen wollen, dann sollten Sie einen HTML-Link in das Bemerkung Attribut des Testfall Knotens aufnehmen. Sie werden diesen Link auch später im Report sehen. Des Weiteren ist es möglich, eine separate Testdokumentation mittels der Menüaktion Datei→Testdoc erstellen zu erzeugen. Weitere Details über die Dokumentation finden Sie unter Kapitel 24(330).

3.1+ Der Report und die Testdokumentation können auch Testschritt Knoten, welche in Testfall Knoten benutzt werden, beinhalten.

Wenn ein Testfall aus vielen Prozeduraufrufen oder Sequenzen besteht, dann sollten Sie die einzelnen Testschritte in Testschritt Knoten organisieren. Diese Testschritt Knoten haben den Vorteil, dass Sie jeden relevanten Testschritt in QF-Test auf einen Blick sehen und diese auch im Report nach der Ausführung aufgeführt werden.

Wenn Sie mehrere Knoten in einem Testschritt zusammenfassen wollen, dann können Sie diese Knoten in einen Testschritt packen. Hierzu selektieren Sie die betroffenen Knoten, führen einen Rechtsklick aus und wählen Knoten einpacken→Testschritt im Kontextmenü aus.

37.2 Modularisierung

Eines der wichtigsten Konzepte einer effektiven Testautomatisierung ist die Modularisierung. Modularisierung in diesem Bereich steht für das Ablegen von wiederverwendbaren Sequenzen an einer bestimmten Stelle, damit diese von überall her aufgerufen werden können, wenn möglich. Dieses Konzept ermöglicht es Ihnen, eine Sequenz nur einmal zu erstellen, aber diese so oft wie benötigt zu benutzen ohne die Sequenz erneut aufzuzeichnen. Falls sich das SUT in einer dieser wiederverwendbaren Sequenzen ändern sollte, z.B. eine Änderung eines Basisworkflows, dann müssen Sie diese Änderung nur an einer Stelle, nämlich der wiederverwendbaren Sequenz, anpassen und nicht in mehreren Testfällen in unterschiedlichen Testsuiten.

Modularisierung wird in QF-Test mittels Prozedur Knoten realisiert. Prozeduren sind im Abschnitt 8.5(157) beschrieben.

Wenn Sie eine große Menge an Testfällen haben, sollte fast jeder Testschritt eine Prozedur sein und Sie könnten diese Prozeduren vorab erstellen, wenn möglich. Danach können Sie die Testfälle mit diesen Prozeduren füllen und das nur mittels Hinzufügen des jeweiligen Prozeduraufruf Knotens.

In größeren Projekten kann es ebenfalls nützlich sein, Prozeduren in unterschiedlichen

Schichten zu erstellen, z.B. komponentenspezifische Prozeduren wie 'Klick auf OK' und workflowspezifische Prozeduren wie 'Ein Fahrzeug anlegen'.

37.3 Parametrisierung

Das Konzept der Modularisierung ermöglicht es, Testschritte an einer zentralen Stelle zu pflegen. Aber was ist mit den Testdaten für verschiedene Tests?

Wenn Sie eine Prozedur erstellt haben, welche mit unterschiedlichen Testdaten aufgerufen werden kann, z.B. ein typischer 'Login' Ablauf mit Benutzernamen und Passwort oder die 'Zubehör auswählen' Prozedur des CarConfigurators, dann können Sie Variablen in den QF-Test Knoten verwenden. Diese Variablen sollten für einen Texteingabe Knoten, für die Auswahl von Elementen in einer Liste oder Tabelle oder Auswahl eines Baumknotens verwendet werden.

Wenn eine Prozedur Variablen benötigt, dann sollten Sie diese in der Liste Variablendefinitionen(673) definieren. Dies dient dazu, eine Liste aller benötigten Parameter bei jedem Einfügen eines Prozeduraufruf Knotens der Prozedur zu bekommen. Einige Kunden benutzen für diese Standardwerte sogar Dummywerte, damit man sofort merkt, dass ein Parameter nicht vom Test initialisiert wurde.

Der nächste Schritt ist nun, die Variablen vom Prozeduraufruf entweder in die Variablendefinitionen(604) Liste des Testfall Knotens oder in einen Datentreiber mit einer Datentabelle oder sonstigen externen Datenquelle zu verschieben.

Variablen und Parameter sind im Abschnitt 8.5(157) beschrieben. Parameter können auch automatisch erstellt werden, siehe Abschnitt 8.5.4(160). Sie finden weitere Details über das Datentreiber Konzept für das Laden von Testdaten aus Datenquellen in Kapitel 23(319).

37.4 Arbeiten in mehreren Testsuiten

Bis jetzt haben Sie die Modularisierung und Parametrisierung Konzepte kennen gelernt, mit denen Sie redundante und damit nicht notwendige Arbeiten während der Testerstellung vermeiden können. Sie sollten gemerkt haben, dass sich mit diesen Konzepten natürlich auch der Wartungsaufwand der Testsuiten im Falle von Änderungen reduzieren lässt, da Sie Änderungen nur einmal statt öfters durchführen. Aber wie können wir die Arbeit für mehrere Testentwickler oder für ein sehr großes Projekt mit vielen GUI-Komponenten aufteilen?

Die Antwort für eine effektive Arbeitsorganisation kommt wiederum aus der Softwareentwicklung. Wir sollten unterschiedliche 'Bibliotheken' für unterschiedliche Bereiche und

unterschiedliche Zuständigkeiten erstellen.

Verbinden von Testsuiten untereinander ermöglicht es Ihnen, eine Kapselung der Testsuiten durchzuführen. Eine typische Struktur in Projekten könnte in etwa so aussehen:

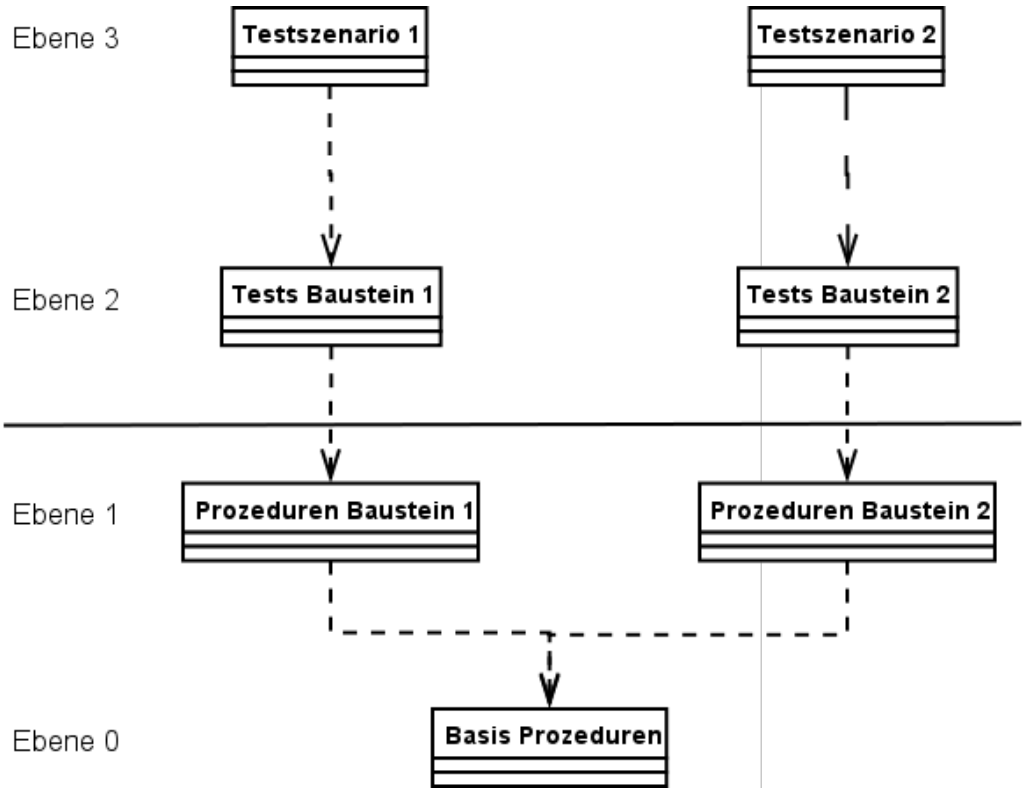


Abbildung 37.1: Struktur mit mehreren Testsuiten

Ebene 0 beinhaltet Testschritte (d.h. Prozeduren), welche für fast alle Testfälle im Projekt von Interesse sind. Solche Testschritte sind beispielsweise 'SUT starten' oder 'Login ausführen'.

Ebene 1 enthält Testschritte für bestimmte Bereiche des SUT. Für den CarConfigurator könnte man sich beispielsweise die Testsuite 'Fahrzeuge', welche Prozeduren wie 'Fahrzeug erstellen' oder 'Fahrzeug löschen' beinhaltet, und eine weitere Testsuite 'Zubehör', welche Prozeduren wie 'Zubehör anlegen' oder 'Zubehör löschen' beinhaltet, vorstellen.

Ebene 2 ist die Testfallebene. Diese beinhaltet Testfälle und Testfallsätze für einen bestimmten Bereich des SUT, z.B. 'Tests für Anlage von Fahrzeugen' oder 'Tests für Löschen von Zubehör'. Es ist auch möglich sich Testsuiten, wie 'Integrationstests', welche Testschritte aus unterschiedlichen Bereichen der Ebene 1 und Ebene 0 aufrufen, auszudenken.

Ebene 3 ist die so genannte Szenarioebene. Diese Testsuiten beinhalten normalerweise

nur Testaufrufe aus Ebene 2 und stehen für unterschiedliche Szenarien, z.B. 'Nächtliches Testszenario', 'Fehlerverifikationsszenario' oder 'Schnelle Buildverifikation'.

Hinweis

Die Struktur, die in diesem Dokument beschrieben wird, ist natürlich nur als Vorschlag und eine mögliche Lösung zu sehen, wie man Testsuiten organisiert. Das ist keine strenge Regel, die unbedingt befolgt werden muss. Sie könnten auch Ebene 1 in eine GUI-Komponenten-Ebene und eine Workflow-Ebene aufteilen oder Ebene 2 und Ebene 3 zusammenlegen. Welche Struktur Sie schlussendlich verfolgen, hängt auch von den Erfahrungen und Fähigkeiten Ihrer im Projekt involvierten Testentwickler ab.

Der 'Include'-Bereich von Ebene-1-Testsuiten sieht wie folgt aus:

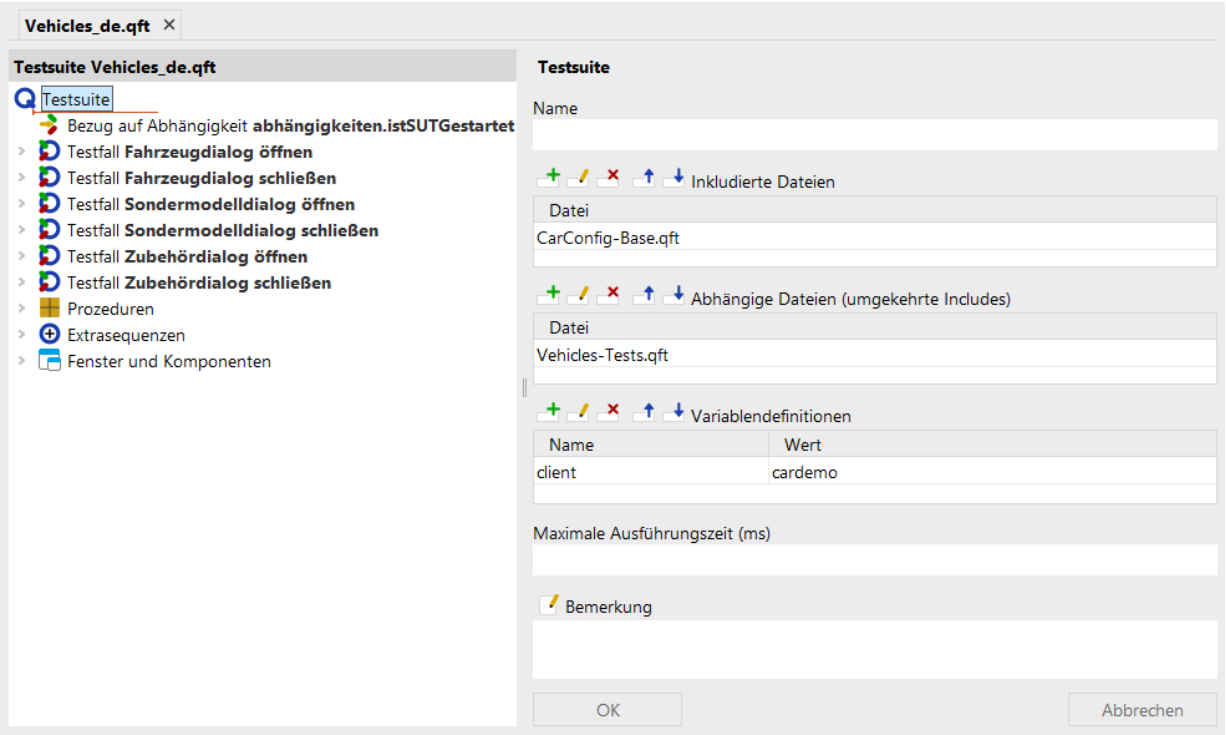


Abbildung 37.2: 'Include'-Bereich von Ebene-1-Testsuiten

Sie finden genauere Informationen über das Inkludieren von Testsuiten unter Abschnitt 26.1(359) und Abschnitt 49.6(1026).

Im Abschnitt 38.5(472) finden Sie eine Schritt-für-Schritt Beschreibung, wie Sie bereits erstellte Testsuiten erweitern können. Im Abschnitt 38.4(471) finden Sie Strategien für die Verwaltung von Komponenten.

37.5 Rollen und Zuständigkeiten

Wenn Sie einen näheren Blick auf den vorigen Abschnitt 37.4(465) werfen, können Sie feststellen, dass man Testsuiten auch anhand von unterschiedlichen Fähigkeiten der Tester strukturieren kann.

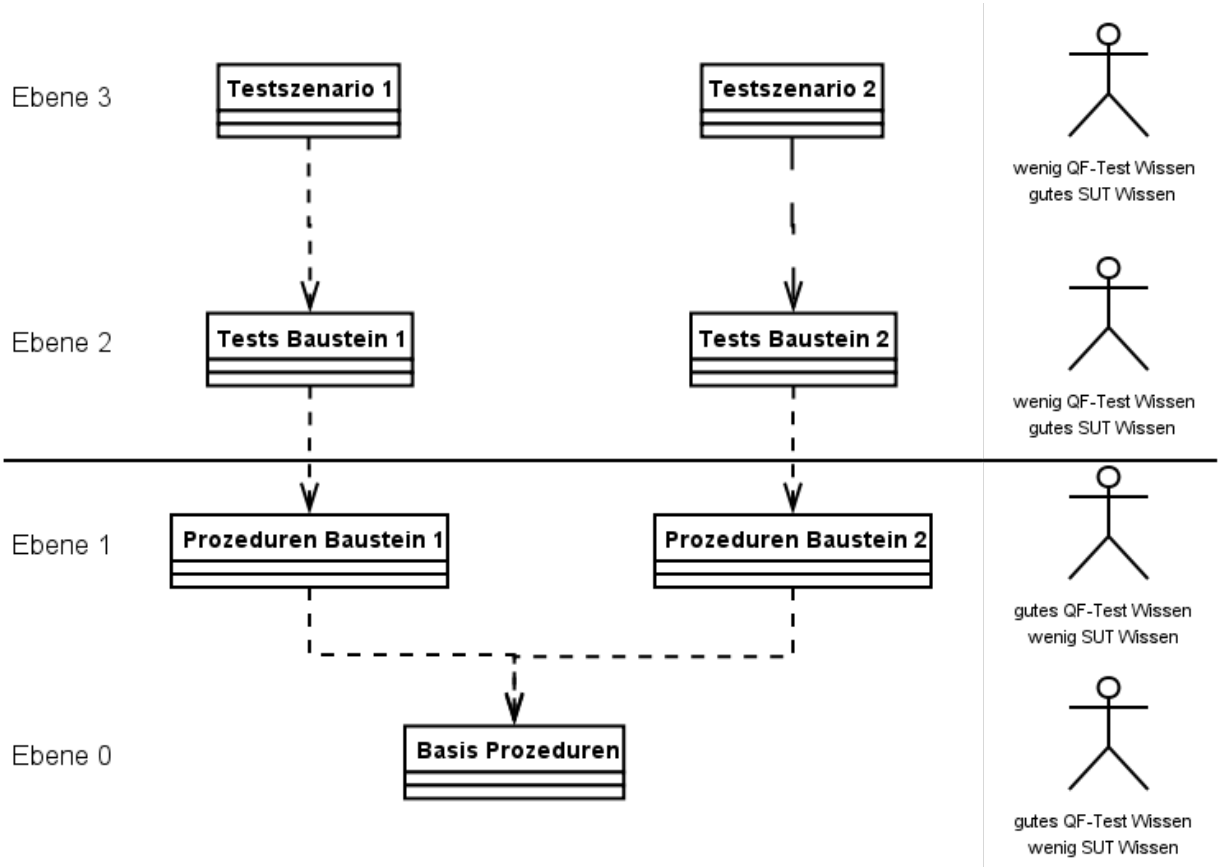


Abbildung 37.3: Struktur von Testsuiten mit Rollen

Ebene 0 und Ebene 1 erfordern gute Kenntnisse in QF-Test, aber nicht unbedingt ein tiefes Wissen über das SUT. Andererseits erfordern Ebene 2 und Ebene 3 sehr gute Kenntnisse des SUT und der geplanten Testfälle, allerdings kein großes Wissen über QF-Test, so lange diese nur Prozeduren von Ebene 0 und Ebene 1 verwenden.

Tester, die in Ebene 0 und Ebene 1 arbeiten, sollten in der Lage sein, Skripte oder Kontrollstrukturen (wie das Try/Catch Konzept), welche die Erstellung einer mächtigen Testbibliothek ermöglichen, umzusetzen. Mindestens einer dieser Tester sollte auch über ein sehr gutes Wissen verfügen, wie die QF-Test Komponentenerkennung funktioniert. Diese ist beschrieben in den Kapiteln Kapitel 5(47), Abschnitt 5.9(92) und Abschnitt 5.3(54).

Hinweis Falls Sie alleine in einem Projekt arbeiten, sollten Sie zumindest Tests und Prozeduren

voneinander trennen, um die Wartbarkeit zu erleichtern. Arbeiten in mehreren Testsuiten ist einfacher als alles in einer großen Testsuite zu halten.

37.6 Komponenten in unterschiedlichen Ebenen verwalten

Wenn Sie den Ansatz des vorigen Abschnittes (Abschnitt 37.4(465)) befolgen, dann müssen Sie noch klären, wo die Komponenten abgelegt werden sollen. Hier gibt es zwei Möglichkeiten:

1. Speichern Sie alle Komponenten in Ebene 0.
2. Aufteilen der Komponenten analog zu den Prozeduren in unterschiedliche Bereiche.

Es ist die einfachste Lösung, alle Komponenten in Ebene 0 zu speichern. Allerdings führt das zu sehr häufigen Änderungen von Ebene 0, nur weil sich einige Komponenten geändert haben. Sie müssen auch Zuständigkeiten definieren und eine saubere Struktur beibehalten.

In großen Projekten können Sie sich überlegen, allgemeine Komponenten wie die Loginmaske und die Navigationselemente - also Komponenten, die für jeden wichtig sind - in Ebene 0 zu speichern. Komponenten, welche nur für einen bestimmten Bereich interessant sind, sind allerdings in der jeweiligen Testsuite der Ebene 1 zu speichern, z.B. ein Fahrzeugdialog für den CarConfigurator sollte nur in der Fahrzeuge.qft Testsuite gespeichert werden.

Der Arbeitsablauf wie man Komponenten zwischen unterschiedlichen Testsuiten verschiebt ist im Abschnitt 38.4(471) beschrieben. Eine Beschreibung, wie man bereits bestehende Testsuiten erweitert, finden Sie in Abschnitt 38.5(472).

37.7 Umgekehrte Includes

3.5+ Für Testsuiten die zu einem QF-Test Projekt gehören, wie unter Kapitel 9(180) beschrieben, brauchen Sie sich nicht um Abhängige Dateien(597) zu kümmern. QF-Test übernimmt die Anpassungen der Testsuiten automatisch für Sie.

Wenn Sie mit unterschiedlichen Testsuiten arbeiten, dann kommt es manchmal vor, dass Sie eine Prozedur oder einen Testfall umbenennen oder verschieben. Falls Sie das tun, werden die Referenzen zu dieser Prozedur oder zum Testfall in anderen Testsuiten nicht angepasst. Wenn Sie diese Änderungen auch zu anderen Testsuiten propagieren

wollen, müssen Sie diese Testsuiten im Attribut Abhängige Dateien des Wurzelknotens der Bibliothek pflegen.

Wenn Sie den Ansatz aus Abschnitt 37.4[465] folgen, sollten Sie sicherstellen, dass Ebene 0 'Umgekehrte Includes' auf alle Suiten von Ebene 1, Ebene 1 auf Ebene 2 und Ebene 2 auf Ebene 3 enthält. Ein Beispiel aus den Demotestsuiten finden Sie in Abbildung 37.2[467].

Kapitel 38

Effiziente Arbeitstechniken

Dieses Kapitel beschreibt hilfreiche Arbeitstechniken, um unnötigen Mehraufwand beim Arbeiten mit QF-Test zu vermeiden.

38.1 Arbeiten mit QF-Test Projekten

3.5+

Im vorigen Kapitel wurde die Erstellung von mehreren Testsuiten beschrieben. Wie Sie sich sicher vorstellen können, werden sich während der Entwicklung einige Testsuiten ansammeln. Um hier einen besseren Überblick über die bestehenden Suiten zu bekommen, sollten Sie ein QF-Test Projekt erstellen.

QF-Test Projekte ermöglichen Ihnen eine sehr gute Übersicht über die involvierten Testsuiten. Des Weiteren sind Projekte auch dafür zuständig, Änderungen an Testsuiten bei Bedarf an aufrufende Testsuiten automatisch weiterzugeben. Nähere Informationen hierzu finden Sie unter Kapitel 9(180).

38.2 Erstellung von Testsuiten

Im vorigen Kapitel wurde die Erstellung von wartbaren Testsuiten mittels Prozeduren und Variablen in QF-Test beschrieben. Normalerweise beginnen Anwender damit, sehr lange Sequenzen aufzuzeichnen und diese dann in kleinere Bereiche und später dann in Prozeduren aufzuteilen. Allerdings ist dieses Aufteilen manchmal ziemlich schwierig und zeitaufwendig, weil Sie die gesamte Sequenz durchgehen und entsprechende Grenzen finden müssen. Ein weiterer Nachteil ist, dass Sie nicht sofort erkennen, welche Schritte Sie bereits in bestehenden Testfällen oder Prozeduren erstellt haben.

Stattdessen empfehlen wir, Tests und deren Schritte im Voraus zu planen. Diese Planung sollte auch die verwendeten und benötigten Prozeduren umfassen. Damit können

Sie dann Prozedur für Prozedur einzeln aufzeichnen. Wir kamen zum Schluss, dass eine vorausschauende Aufzeichnung, besonders für größere Teams, sehr hilfreich ist. Ein typischer Workflow für die Testerstellung könnte in etwa so aussehen:

1. Zuerst planen Sie die benötigten Prozeduren.
2. Planen Sie auch die benötigte Packagestruktur.
3. Zeichnen Sie jede Prozedur als separate Sequenz auf.
4. Benennen Sie die aufgezeichnete Sequenz so um, wie die Prozedur heißen würde.
5. Konvertieren Sie diese Sequenz nun in eine Prozedur, mittels der Knoten konvertieren Aktion aus dem Kontextmenü.
6. Schieben Sie die Prozedur nun an die entsprechende Stelle unter Prozeduren.
7. Ersetzen Sie Testdaten durch Variablen. Dies können Sie entweder manuell oder automatisch (siehe Abschnitt 8.5.4(160) erledigen.
8. Legen Sie die benötigten Variablen im Variablendefinitionen Attribut des Prozedur Knotens an. Eventuell können Sie auch Standardwerte vergeben.
9. Beschreiben Sie die Prozedur im Bemerkung Attribut, siehe Abschnitt 8.7(179).

Ein alternativer Ansatz zur Erstellung von Prozeduren kann deren automatische Erstellung sein. Dieses Konzept ist im Kapitel 27(368) beschrieben.

38.3 Die Standardbibliothek qfs.qft

QF-Test liefert eine Standardbibliothek `qfs.qft` aus, die in jeder Testsuite inkludiert ist.

Diese Suite enthält viele hilfreiche Prozeduren, um auf Komponenten, Datenbanken oder das Dateisystem zuzugreifen. Bitte werfen Sie immer einen Blick in diese Testsuite, bevor Sie etwas selber versuchen zu lösen, was eventuell schon einmal von uns gelöst wurde.

38.4 Ablage von Komponenten

QF-Test zeichnet jede neue Komponente in der Testsuite auf, in der die Aufzeichnung gestoppt wurde, deshalb könnte es passieren, dass Komponenten in einer falschen Testsuite abgelegt werden.

Wenn Sie nun diese Komponenten in eine andere Testsuite verschieben wollen, sollten Sie dies immer mit der Menüaktion Datei→Importieren der Zieltestsuite ausführen. Stellen Sie sicher, dass beide Testsuiten zum selben Projekt gehören, oder dass Sie hierfür ein korrektes 'Inkludierte Dateien'/'Abhängige Dateien' Verhältnis zwischen den beiden Testsuiten erstellt haben. Der dazugehörige Arbeitsablauf ist detailliert im Abschnitt 26.2(360) beschrieben.

3.1+ Um Ihre Komponentenstruktur sauber zu halten, können Sie zuerst die Testsuite in sich selbst importieren. Danach können Sie den Fenster und Komponenten Knoten selektieren und nach einem Rechtsklick die Menüaktion Ungenutzte Komponenten markieren... auswählen. Sie bekommen nun eine Liste aller Komponenten, die nicht im Projekt verwendet werden. Wenn Sie sich sicher sind, dass diese Komponenten gelöscht werden können, selektieren Sie im Kontextmenü des Fenster und Komponenten Knoten den Eintrag Ungenutzte Komponenten entfernen.

Hinweis Sobald eine Komponente Informationen im Kommentarfeld enthält wird diese als "genutzt" betrachtet, selbst wenn es keine direkten Referenzen auf die Komponenten gibt.

38.5 Erweitern von Testsuiten

Sie können bestehende Testsuiten auf unterschiedliche Arten erweitern:

1. Sie zeichnen einfach die neuen Schritte in der Testsuite direkt auf.
2. Sie arbeiten mit einer Erweiterungssuite wie im Kapitel 26(358) beschrieben.

Wenn Sie Erweiterungen direkt in einer Testsuite vornehmen, indem Sie in der entsprechenden Testsuite auf 'Aufnahme stoppen' klicken, dann müssen Sie darauf achten, dass die Komponenten eventuell in einer anderen Hierarchie aufgezeichnet werden könnten. Das kann passieren, wenn Sie die aufgezeichnete Komponentenhierarchie unter Fenster und Komponenten verändert haben. Des Weiteren kann das Verschieben von einzelnen Komponenten schwierig werden.

Wenn Sie mit einer Erweiterungssuite arbeiten und darin neue Testschritte erstellt haben, können Sie die aufgezeichneten Komponenten und erstellten Prozeduren und Tests auf einmal in die eigentliche Testsuite importieren.

Für die Erweiterung von Testsuiten auf Ebene 1 (wie beschrieben im Abschnitt 37.4(465)) könnte daher wie folgt aussehen:

1. Erstellen Sie eine neue Testsuite.
2. Fügen Sie die zu erweiternde Testsuite zum Inkludierte Dateien Bereich der neuen Testsuite hinzu.

3. Speichern Sie die neue Testsuite.
4. Stellen Sie sicher, dass beide Testsuiten zum selben Projekt gehören oder fügen Sie die neue Testsuite zum Abhängige Dateien Bereich der zu erweiternden Testsuite hinzu.
5. Zeichnen Sie neue Testschritte in der Entwicklungssuite auf. Erstellen Sie auch sofort die Prozeduren, sofern benötigt.
6. Jetzt importieren Sie die Komponenten, Prozeduren und Testfälle in die Zieltestsuite wie im Abschnitt 38.4(471) und Abschnitt 26.3(361) beschrieben.

Eine detailliertere Beschreibung, wie man mit mehreren Testsuiten arbeitet, finden Sie im Kapitel 26(358).

38.6 Arbeiten mit dem Skripteditor

Der Skripteditor von QF-Test enthält einige nette Möglichkeiten, um Ihnen Tipparbeit zu ersparen.

Wenn Sie Methoden des Runcontexts `rc` aufrufen wollen, dann tippen Sie einfach `rc.` und drücken danach Strg-Leertaste. Jetzt bekommen Sie eine Liste aller unterstützten Methoden.

Diese automatische Vervollständigung funktioniert für folgende Variablen:

Variablen	Methoden
doc	Methoden von DocumentNode.
frame	Methoden von FrameNode.
iw	Methoden von ImageWrapper.
node	Methoden von DOMNode.
Options	Die Namen und Werte der QF-Test Optionen.
qf	Methoden des qf Modules.
rc	Methoden des Runcontexts.
resolvers	Methoden des Resolvers Moduls.
Wenn Sie nur Strg-Leertaste klicken ohne etwas einzugeben	Eine Liste aller Variablen, für die Vervollständigung möglich ist.

Tabelle 38.1: Liste der Variablen mit Vervollständigung.

Kapitel 39

Aufsetzen von Testsystemen

Dieses Kapitel gibt Ihnen Hinweise, wie Sie Ihre Testsysteme und Prozesse aufsetzen können, um die Grundlagen für eine stabile Testausführung legen zu können.

39.1 Einrichten von Prozessen und Services via Aufgabenplaner

Um Tests oder andere Prozesse regelmäßig auszuführen, werden auf Windows oft Services eingesetzt. Diese Services haben aber den Nachteil, dass sie nicht unter einer verwendbaren Benutzersession laufen. Demnach sollten Prozesse, wie ein Buildserverprozess oder Skripte, die den QF-Test Daemon starten, nicht als Service gestartet werden, weil es sonst zu Problemen bei der GUI Testausführung kommen kann. Technische Details hierzu finden Sie unter FAQ 14.

Wir empfehlen in der Regel statt Services eine Aufgabe über den Aufgabenplaner zu definieren. Diese können Sie direkt über die grafische Oberfläche des Aufgabenplaners einrichten. Die folgende Vorgehensweise funktioniert bei Windows 7, Windows 8, Windows 8.1 und Windows 10, wobei sich die Dialoge nur leicht unterscheiden:

1. Starten Sie dazu zuerst den Aufgabenplaner über die Systemsteuerung -> Verwaltung -> Aufgabenplanung.
2. Klicken Sie auf ”Aufgabe erstellen” rechts oben.
3. Auf dem ”Allgemein” Reiter geben Sie einen Namen an, z.B. ”QF-Test”.
4. Nun klicken Sie auf ”Benutzer oder Gruppen ändern” und wählen den Benutzer aus, in dessen Session der Test ausgeführt werden soll. Wählen Sie hier unbedingt einen echten Benutzer und keinen Pseudo-User und nicht die Service-Session.

5. Klicken Sie auf OK, um den Dialog zu schließen.
6. Wählen Sie dann ”Nur ausführen, wenn der Benutzer angemeldet ist”.
7. Wählen Sie _nicht_ ”Mit höchsten Privilegien ausführen”.
8. Sie können nun noch die korrekte Windows-Version für die Aufgabe anwählen.
9. Auf dem ”Trigger” Reiter klicken Sie auf ”Neu...” und definieren Sie den Trigger auf dem neuen Dialog.
10. Schließen Sie danach den Dialog zur Zeitplanung.
11. Auf dem ”Aktionen” Tab klicken Sie nun ”Neu...” und wählen Sie ”Programm starten” und ”Durchsuchen” Sie den Rechner nach dem .cmd oder .bat Script, welches entweder den Batch-Aufruf für QF-Test enthält oder den Prozess Ihres Buildsystems startet.
12. Mit OK schließen Sie wieder den Dialog.
13. Auf dem ”Bedingungen” Reiter und ”Einstellungen” Tab können Sie nun noch die gewünschten Einstellungen vornehmen.
14. Danach können Sie die Konfiguration abschließen.

Wichtig ist nun, dass der User, in dessen Session dieser Prozess ausgeführt wird, zuvor eingeloggt wird. Entweder manuell oder automatisch (siehe Abschnitt 39.3(476)). Empfehlenswert sind daher für GUI Tests immer virtuelle Maschinen. Hier muss nur auf dem Gastsystem der User eingeloggt sein, der Host kann gesperrt werden.

39.2 Fernzugriff auf Windowsrechner

Beim Fernzugriff auf Windowsrechner unterliegt die Verwendung von RDP einigen Einschränkungen bzw. erfordert eine explizite Konfiguration des Systems, um uneingeschränkt verwendet werden zu können. Dies liegt daran, dass die bei den Windows Desktop Varianten implementierte Variante des Zugriffs über RDP immer nur einen aktiven Anwender erlaubt. Wenn Sie sich also per RDP mit einem Testrechner verbinden, wird dessen (virtueller) Bildschirm gesperrt, während Sie auf diesem Rechner arbeiten können. Schließen Sie Ihr RDP-Fenster, bleibt jedoch der Bildschirm des Testrechners gesperrt. Und auf einem gesperrten Bildschirm kann man unter üblichen Umständen keine Benutzeroberfläche bedienen, also auch nicht testen.

Hinweis Ab Windows 10 bzw. Windows Server 2016 können Sie unter gewissen Voraussetzungen RDP verwenden, wenn Sie folgende Änderung in der Registry

vornehmen. Unter `HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client` oder `HKEY_LOCAL_MACHINE\Software\Microsoft\Terminal Server Client` fügen Sie bitte einen neuen `DWORD` namens `RemoteDesktop_SuppressWhenMinimized` ein. Dort setzen Sie den Wert `2`. Das erlaubt dann das Minimieren des Fensters einer RDP-Verbindung, aber leider immer noch nicht das Schließen oder Trennen.

Natürlich können Sie auch Alternativen zu RDP verwenden. Hier können Sie sich z.B. der Möglichkeiten bedienen, die Ihnen der Server für die virtuellen Maschinen zur Verfügung stellt. Beim einem VMware Server bietet sich also der vSphere Client an, bei VirtualBox kann man sich mit RDP mit VirtualBox (nicht mit dem Windows RDP des Clients) verbinden. Letzterer hat dann selbstverständlich nicht die oben genannten Effekte mit der Bildschirm-Sperrung.

39.3 Automatische Anmeldung auf Windowsrechnern

Eine Möglichkeit sicherzustellen, dass ein Testrechner in den meisten Fällen eine aktive Session hat, ist, dass sich auf diesem Rechner bei jedem Start automatisch ein Testbenutzer anmeldet. Wie man ein Windowssystem entsprechend konfigurieren kann, ist in diesem Abschnitt beschrieben.

Hinweis

Eine automatische Anmeldung an Windows ist immer ein Sicherheitsrisiko. Daher sollte selbstverständlich sichergestellt sein, dass die betroffenen Rechner außerhalb der Testumgebung nicht zugänglich sind.

Diese Anleitung ist zwar grundsätzlich gültig, wird aber üblicherweise auf virtuelle Rechner angewendet werden, auf die dann von Fall zu Fall remote zugegriffen werden wird. Was bei diesem Remote-Zugriff zu beachten ist, finden Sie im Abschnitt 39.2(475).

Die folgende Vorgehensweise funktioniert bei Windows 7, Windows 8, Windows 8.1 und Windows 10, wobei sich die Dialoge nur leicht unterscheiden:

1. Starten Sie eine Eingabeaufforderung mit Administrator-Rechten.
2. Geben Sie `control userpasswords2` ein.
3. Es öffnet sich daraufhin der Dialog ”Benutzerkonten”.
4. Bitte entfernen Sie hier das Häkchen bei ”Benutzer müssen Benutzernamen und Kennwort eingeben”, nachdem Sie den Benutzer markiert haben, der sich später automatisch anmelden soll.
5. In dem nun erscheinenden Dialog ”Automatische Anmeldung” tragen Sie bitte das Benutzer-Kennwort zwei Mal ein.

6. Nach dem anschließenden Klick auf die ”OK” Schaltfläche ist die automatische Anmeldung fertig eingerichtet.

Es gibt auch noch andere Wege, dies zu erreichen. Zum Einen könnten Sie den entsprechenden Registry-Eintrag direkt ändern. Zum Anderen aber auch das Tool ”Autologon” von Microsoft unter https://technet.microsoft.com/de-de/sysinternals/bb963905 herunter laden. Alle diese Varianten erreichen aber im Endeffekt exakt dasselbe (nämlich den entsprechenden Registry-Eintrag). Wir empfehlen jedoch die hier vorgestellte Variante, da kein Download nötig ist und Tippfehler beim Registry-Eintrag ausgeschlossen sind. Übrigens funktioniert die automatische Anmeldung nie bei Benutzern, die sich an einer Domäne anmelden. Aber das wäre bei Testrechnern ja auch ungeschickt, wird aber die meisten Administrator:innen sehr beruhigen.

39.4 Testausführung unter Linux

Auf Linux-Systemen können virtuelle Displays mittels Tools wie VNC-Server einfach eingerichtet werden. Ein guter Windowmanager hierfür ist, z.B. xfce.

Kapitel 40

Testausführung

Dieses Kapitel gibt Ihnen Hinweise, wie Sie Ihre Tests implementieren können um eine stabile und verlässliche Testausführung zu erreichen.

40.1 Abhängigkeiten

Das 'Abhängigkeiten' Konzept von QF-Test ermöglicht es, dass alle Vorbedingungen eines Testfalles vor der eigentlichen Ausführung des Testfalles sichergestellt werden. Es bietet auch Möglichkeiten, auf ein unerwartetes Verhalten zu reagieren, z.B. Schließen eines unerwarteten Fehlerdialogs.

Dieses Konzept ist im Abschnitt 42.3(630) beschrieben. Im Tutorial können Sie einen Anwendungsfall im Kapitel 'Abhängigkeiten' finden.

Sie sollten zumindest eine Abhängigkeit implementieren, die verantwortlich ist das SUT zu starten. Diese Abhängigkeit sollte eine Vorbereitung Sequenz, die das SUT startet, eine Aufräumen Sequenz für das normale Beenden des SUTs und darüber hinaus noch einen Catch Knoten, um auf unerwartetes Verhalten zu reagieren, beinhalten.

Hinweis

Wenn Sie eine Aufräumen Sequenz implementieren, versuchen Sie zuerst das SUT normal zu beenden. Nur wenn das nicht klappt, sollten Sie versuchen das SUT mit einem Programm beenden Knoten abzuschießen.

Für SWING und SWT Applikationen, verwenden Sie bitte die Prozeduren `qfs.cleanup.swing.closeAllModalDialogs` und `qfs.cleanup.swt.closeAllModalDialogsAndShells` aus der Standardbibliothek `qfs.qft` um unerwartete Fehlerdialoge zu schließen.

40.2 Wartezeiten und Verzögerungen

Zur Optimierung der Ausführungszeit sollten Sie anstatt der Attribute 'Verzögerung vorher' und 'Verzögerung nachher' Synchronisationsknoten von QF-Test verwenden.

Die erste Art von Synchronisationsknoten sind die 'Warte'-Knoten wie Warten auf Komponente, Warten auf Client, Warten auf Laden des Dokuments und Warten auf Programmende. Hier können Sie das Attribut Wartezeit setzen, um auf eine Komponente, einen Prozess oder ein Dokument zu warten. Der Warten auf Komponente Knoten bietet sogar die Möglichkeit, auf das Verschwinden von Komponenten zu warten.

Die zweite Art sind 'Check'-Knoten, bei denen Sie auch wieder das Attribut Wartezeit setzen können. Diese Knoten können verwendet werden, um einen Test erst fortsetzen, wenn eine GUI-Komponente einen bestimmten Status erreicht hat.

40.3 Was soll man tun, wenn das Protokoll einen Fehler enthält?

Wenn der Testreport eine Fehlermeldung oder eine Exception enthält, dann sollten Sie folgende Schritte ausführen, um die Fehlerquelle schnell zu finden:

1. Analysieren Sie das Protokoll, insbesondere die Bildschirmabbilder und Meldungen.
2. Wenn Sie die Fehlerquelle nicht sofort sehen können, dann springen Sie zur fehlerhaften Stelle in der Testsuite mit der Tastenkombination Strg-T.
3. Setzen Sie einen Breakpoint vor dem fehlerhaften Schritt.
4. Versichern Sie sich, dass der Debugger von QF-Test eingeschaltet ist.
5. Starten Sie den fehlerhaften Test.
6. Wenn QF-Test den Breakpoint erreicht und die Ausführung stoppt, öffnen Sie das Debugger-Fenster und kontrollieren die aktuellen Variablenbelegungen, vielleicht beinhaltet diese ja falsche Werte.
7. Vielleicht sehen Sie jetzt auch sofort den Fehler im SUT.
8. Wenn Sie immer noch keine Fehlerursache erkennen können, dann führen Sie den fehlerhaften Schritt aus.
9. Wenn Sie nun immer noch einen Fehler bekommen, dann müssen Sie eventuell einige Schritte vor dem eigentlichen Fehler debuggen. Verwenden Sie hierzu

wenn möglich die Aktion 'Ausführung hier fortsetzen' aus dem Kontextmenü, um zu einem beliebigen Knoten in der Testsuite zu springen anstatt eines Neustarts des kompletten Tests.

3.1+ Seit QF-Test Version 3.1 ist es möglich Knoten mittels Kontextmenü Marke setzen zu markieren oder ein Lesezeichen mittels der Kontextmenüaktion Lesezeichen hinzufügen zu setzen. Diese Features ermöglichen es Ihnen schnell die wichtigsten Knoten wiederzufinden.

Wenn Sie Probleme mit der Komponentenerkennung haben, dann schauen Sie bitte in die Kapitel Abschnitt 5.10(101) und Abschnitt 5.3(54).

Teil III

Referenzteil

Kapitel 41

Einstellungen

Es gibt diverse Möglichkeiten, das Verhalten von QF-Test zu beeinflussen. Insbesondere für das Aufnehmen und Abspielen von Tests gibt es viele Einstellmöglichkeiten.

QF-Test unterscheidet zwischen *User* und *System* Optionen. Useroptionen sind Einstellungen, die das Verhalten der Oberfläche von QF-Test beeinflussen. Sie haben im Gegensatz zu den Systemoptionen keinen Einfluss auf die Aufnahme und das Abspielen von Tests. Useroption werden für jeden Anwender getrennt gespeichert, während Systemoptionen in einer gemeinsamen Systemdatei abgelegt werden. Näheres hierzu finden Sie in Abschnitt 1.6(12).

3.1+ Bei vielen Option kann deren Wert zur Laufzeit, wie in Abschnitt 50.5(1030) beschrieben, per Skript mittels `rc.setOption` verändert werden. Abhängig davon, ob die jeweilige Option in QF-Test selbst oder im SUT ausgewertet wird, ist bei der Beschreibung der Option ein "Server-Skript Name" oder ein "SUT-Skript Name" angegeben, welcher der Konstante aus der `Options` Klasse entspricht. Natürlich muss die Option in einem passenden Server-Skript(717) oder SUT-Skript(720) Knoten gesetzt werden. Falls der Wert der Option aus einer Liste ausgewählt werden kann, führt die Beschreibung auch die Konstanten für die möglichen Werte auf.

Auch wenn die Anzahl der Optionen entmutigend erscheinen mag, lassen Sie sich davon nicht abhalten. Alle Optionen haben vernünftige Standardwerte, sodass QF-Test in den meisten Fällen problemlos funktioniert. Wenn Sie dennoch feststellen, dass Sie etwas ändern müssen oder wenn Sie einfach den Bereich von QF-Test's Fähigkeiten kennenlernen wollen, so ist dieses Kapitel etwas für Sie.

Sie können die Optionen mit Hilfe des Dialogs, den Sie über Bearbeiten→Optionen... erreichen, ansehen und manipulieren. Die Einstellungen werden (mit kleinen Ausnahmen, s.u.) beim Beenden von QF-Test gespeichert (siehe Abschnitt 1.6(12)).

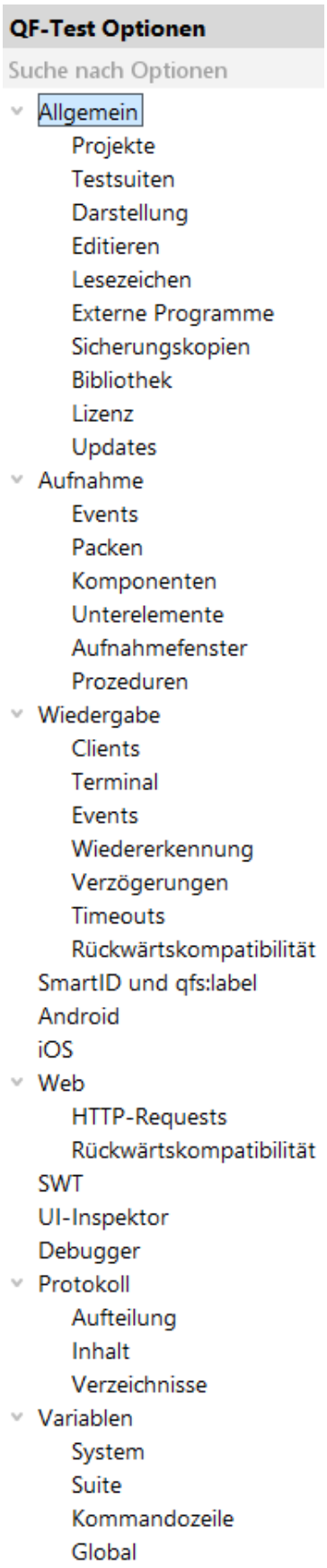


Abbildung 41.1: Optionen Baumstruktur

Um zu den gewünschten Optionen zu gelangen, selektieren Sie zunächst im Baum den entsprechenden Ast. Im rechten Teil können Sie dann die Optionen einstellen. Beim Wechsel von einer Gruppe zur anderen werden die Optionen zwar auf Gültigkeit überprüft, aber noch nicht übernommen. Dies geschieht erst bei Bestätigung mit dem OK Button.

41.1 Allgemeine Optionen

Dieser Ast ist für allgemeine QF-Test Einstellungen zuständig.

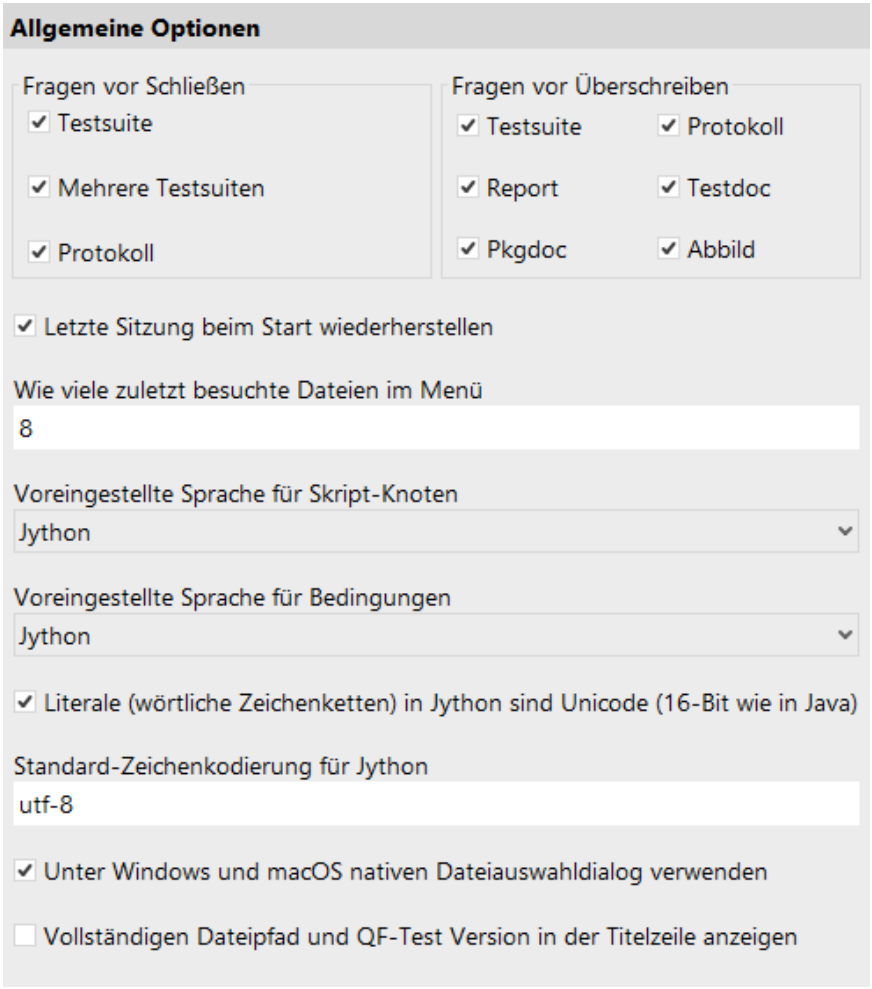


Abbildung 41.2: Allgemeine Optionen

Fragen vor Schließen (User)

Wurde eine Testsuite oder ein Protokoll modifiziert, wird vor dem Schließen eines Fensters oder dem Beenden von QF-Test normalerweise gefragt, ob die Daten gespeichert werden sollen. Diese Abfrage kann durch Ausschalten dieser Optionen, getrennt für Testsuiten und Protokolle, verhindert werden, was gefährlich ist, da die Änderungen dann einfach verworfen werden.

Fragen vor Überschreiben (User)

Wenn Sie eine Testsuite oder ein Protokoll abspeichern oder einen Report, Testdoc oder Pkgdoc-Dokumentation generieren oder ein Abbild eines Check Abbild(828) abspeichern und die Zieldatei bzw. das Verzeichnis bereits existiert, wird normalerweise rückgefragt, ob Sie die Datei überschreiben wollen. Durch ausschalten der entsprechenden Optionen können Sie diese Frage für jeden Dateityp getrennt unterbinden.

Letzte Sitzung beim Start wiederherstellen (User)

Ist diese Option gesetzt und wird QF-Test in der Workbench-Ansicht geöffnet, wird die letzte Sitzung wieder hergestellt indem die zuletzt geöffneten Testsuiten geladen werden und der zuletzt selektierte Knoten in der jeweiligen Suite selektiert wird. Sind ein oder mehrere Testsuiten auf der Kommandozeile angegeben, werden diese zusätzlich geladen und erhalten nach dem Start den initialen Fokus.

Wie viele zuletzt besuchte Dateien im Menü (User)

Im Datei Menü können Sie schnell auf die Testsuiten und Protokolle zugreifen, die Sie zuletzt bearbeitet haben. Diese Option legt die Zahl derartiger Menüeinträge fest.

Voreingestellte Sprache für Skript-Knoten (User)

Diese Option kann die Werte ”Jython”, ”Groovy” oder ”JavaScript” annehmen und legt die Standardeinstellung für das Attribut Skriptsprache(719) in neu erstellten Server-Skript(717) oder SUT-Skript(720) Knoten fest.

Voreingestellte Sprache für Bedingungen (User)

Diese Option kann die Werte ”Jython”, ”Groovy” oder ”JavaScript” annehmen und legt die Standardeinstellung für das Attribut Skriptsprache(695) in neu erstellten If(693), Elseif(697), While(688), Testfallsatz(606) oder Testschritt(621) Knoten fest.

Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java) (System)

5.3+

Server (automatisch weiter an SUT) Skript Name: OPT_JYTHON_UNICODE_LITERALS
Diese Option legt fest, wie Literale (wörtlich definierte String-Konstanten wie "abc") in Jython-Skripen in Server-Skript[717]- und SUT-Skript[720]-Knoten, dem Bedingung[694]-Attribut in If[693] und anderen Knoten sowie den interaktiven Jython-Konsolen für QF-Test und das SUT behandelt werden.

Ist die Option gesetzt, werden Jython-Literale als 16-Bit Unicode-Strings implementiert, genau wie in Java selbst. Andernfalls werden Literale zu den 8-Bit Byte-Strings von Python 2, die schlecht mit Java und somit QF-Test zusammenspielen. Detaillierte Informationen und Beispiele finden Sie in Abschnitt 11.4.5[201].

Falls QF-Test eine bereits bestehende ältere Systemkonfiguration vorfindet, ist diese Option standardmäßig ausgeschaltet, also auf 8-Bit Literale eingestellt. Für neue Installationen ist sie standardmäßig aktiviert.

Standard-Zeichenkodierung für Jython (System)

Server (automatisch weiter an SUT) Skript Name: OPT_JYTHON_DEFAULT_ENCODING
Diese Option legt die Standard-Zeichenkodierung zur Konvertierung zwischen Jython 16-Bit Unicode-Strings und 8-Bit Byte-Strings fest. Diese findet Anwendung bei expliziter Konvertierung wie `str(...)` und bei impliziter Konvertierung. Ist die vorhergehende Option Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java)[485] nicht gesetzt, findet für alle vorkommenden Jython Literale (wörtlich definierte String-Konstanten wie "abc") implizite Konvertierung statt. Detaillierte Informationen und Beispiele finden Sie in Abschnitt 11.4.5[201].

5.3+ Ab QF-Test 5.3 ist der Standardwert für diese Option "utf-8" (vorher war er "latin-1"). Bereits bestehende Systemeinstellungen sind von dieser Änderung nicht betroffen.

Unter Windows und macOS nativen Dateiauswahldialog verwenden (User)

3.5+

Server Skript Name: OPT_USE_NATIVE_FILECHOOSER
Unter Windows und macOS ist der native Dateiauswahldialog der Swing Variante deutlich überlegen und wird daher von QF-Test normalerweise verwendet. Deaktivieren Sie diese Option wenn Sie den Auswahldialog von Swing vorziehen.

Vollständigen Dateipfad und QF-Test Version in der Titelzeile anzeigen (User)

4.1.3+

QF-Test zeigt den kompletten Pfad der aktuellen Testsuite und die QF-Test Version in der Titelzeile des Hauptfensters an, wenn diese Option aktiviert ist.

41.1.1 Einstellungen für Projekte

3.5+

Es gibt verschiedene Optionen, welche die Darstellung und Handhabung von Projekten in QF-Test beeinflussen.

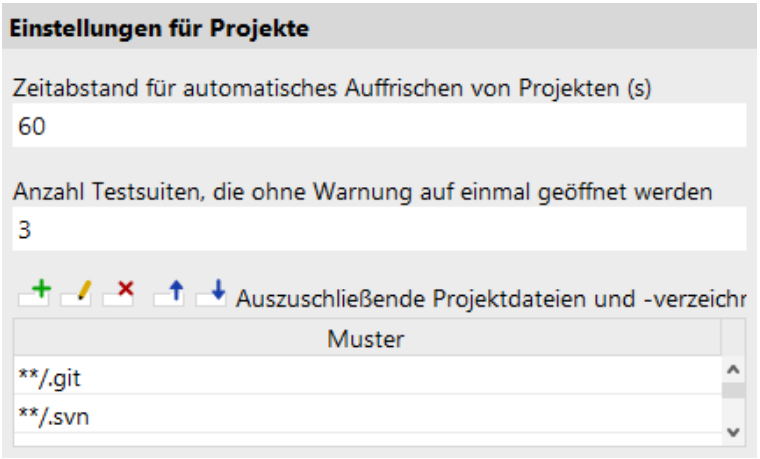


Abbildung 41.3: Einstellungen für Projekte

Zeitabstand für automatisches Auffrischen von Projekten (s) (User)

Projekte werden automatisch im hier definierten Zeitabstand komplett aktualisiert. Sie können ein Verzeichnis jederzeit manuell über das Kontextmenü oder durch drücken von F5 aktualisieren. Um die gesamte Hierarchie unterhalb eines Verzeichnisses aufzufrischen, drücken Sie Shift-F5.

Anzahl Testsuiten, die ohne Warnung auf einmal geöffnet werden (User)

Vom Projektbaum aus können Sie alle Testsuiten, die in einer Verzeichnishierarchie enthalten sind, in einem Rutsch öffnen. Falls Sie dabei versehentlich zu viele Suiten selektieren, zeigt QF-Test zunächst die Anzahl in einer Warnung an, so dass Sie die Aktion noch abbrechen können. Diese Option legt die Schwelle für diese Warnung fest.

Auszuschließende Projektdateien und -verzeichnisse (System)

Es gibt Verzeichnisse und Dateien, die nicht wirklich zu einem Projekt gehören, insbesondere die Unterverzeichnisse, die von verschiedenen Versionierungs-Systemen wie Git, Subversion oder CVS erstellt werden. Über diese Option können Sie Muster für Datei- und Verzeichnisnamen festlegen, die generell aus Projekten ausgeschlossen werden sollen.

Die hier verwendeten Muster sind keine regulären Ausdrücke, sondern eine einfachere Form, die oft in Entwicklungswerkzeugen eingesetzt wird. Aus Kompatibilitätsgründen wird ausschließlich '/' als Trennzeichen verwendet, nicht der unter Windows gebräuchliche Rückstrich '\'. Ein '*' steht für 0 oder mehr Zeichen exklusive '/', '**' für 0 oder mehr beliebige Zeichen inklusive '/'. Jedes Muster ist relativ zum Wurzelverzeichnis des Projekts. Einige Beispiele:

`**/.svn`
: Alle Verzeichnisse namens `.svn` in beliebiger Tiefe.

`**/.*`
: Alle Verzeichnisse beginnend mit einem '.' in beliebiger Tiefe.

`deprecated`
: Ein Verzeichnis namens `deprecated` direkt unter dem Wurzelverzeichnis des Projekts.

41.1.2 Speichern von Testsuiten

7.0+

Die folgenden Optionen legen das Format für das Speichern von Testsuiten als XML fest. Das Format für das Speichern von Protokollen wird durch die Option Protokoll in aktuellem XML-Format mit UTF-8-Kodierung speichern[582] bestimmt.

Hinweis

Unabhängig vom gewählten XML-Format können Testsuiten und Protokolle auch mit QF-Test Versionen älter als 7.0 geöffnet werden (wobei später eingeführte Knotentypen natürlich nicht erkannt werden). Beim Speichern mit einer alten QF-Test Version wird immer auf das alte XML-Format zurückgewechselt.

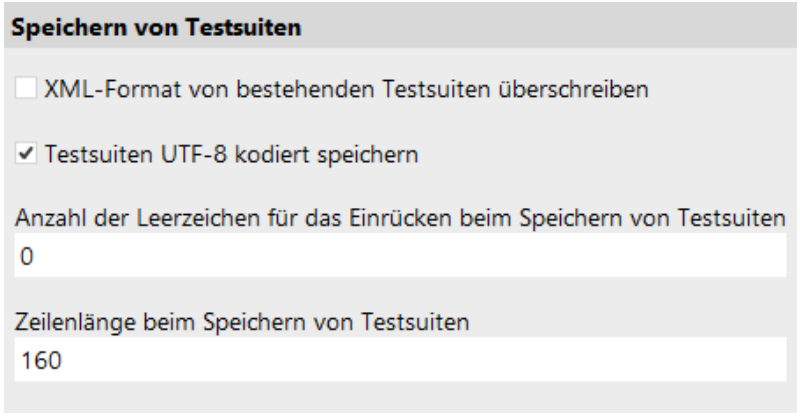


Abbildung 41.4: Speichern von Testsuiten

XML-Format von bestehenden Testsuiten überschreiben (System)

7.0+

Eine Änderung des XML-Dateiformats führt beim nächsten Speichern einer Testsuite zu einer großen Zahl von rein syntaktischen Änderungen. Diese werden in Versionskontrollsystemen sichtbar und überlagern dabei eventuell die eigentlichen inhaltlichen Änderungen. Um dies zu vermeiden, wird das mit den nachfolgenden Optionen definierte XML-Format nur für neue Testsuiten angewandt. Bereits existierende Testsuiten behalten ihr Format bei, sofern diese Option nicht aktiviert wird.

Hinweis

Das Ändern des XML-Formats sollte eine projektweite Entscheidung sein und idealerweise so in einem Rutsch durchgeführt werden, dass alle Dateien gemeinsam und ohne andere Änderungen in einem Commit in das Versionskontrollsystem eingecheckt werden. Die Konvertierung selbst kann sehr einfach mit QF-Test im Batchmodus mit dem Kommandozeilenargument `-convertxml`(979) durchgeführt werden. Die genaue Syntax hierfür finden Sie in Abschnitt 44.1(971).

Testsuiten UTF-8 kodiert speichern (System)

7.0+

Ist diese Option aktiv (der Standard), werden Testsuiten mit UTF-8-Kodierung gespeichert, andernfalls mit ISO-8859-1.

QF-Test Versionen vor 7.0 speichern immer mit ISO-8859-1 Kodierung.

Anzahl der Leerzeichen für das Einrücken beim Speichern von Testsuiten (System)

7.0+

XML-Dateien mit Einrückung sind für Menschen einfacher zu lesen. Testsuiten werden aber primär von QF-Test verarbeitet, ein typischer QF-Test Anwender

bekommt das XML nur im Fall von Merge-Konflikten zu Gesicht. Letztere werden durch eine Einrückung von 0, dem neuen Standardwert, reduziert, da ansonsten im XML alle Zeilen von ein- oder ausgepackten Knoten geändert werden.

QF-Test Versionen vor 7.0 verwenden immer eine Einrückung von 2 Zeichen.

Zeilenlänge beim Speichern von Testsuiten (System)

7.0+

Die einzigen Zeilen, die in XML-Dateien für Testsuiten problemlos umgebrochen werden können, sind die mit den Attributen der Knoten. Der aus Skripten oder Bemerkungen stammende Textinhalt würde durch zusätzliche Umbrüche beschädigt, daher stellt diese Option keine harte Beschränkung der Zeilenlänge dar.

Leider gibt es keinen idealen Standardwert für diese Option. Der aktuelle Standard von 160 Zeichen ist ein Kompromiss zwischen den folgenden beiden Extremen:

Ein negativer oder extrem großer Wert führt zu einer praktisch unbegrenzten Zeilenlänge, genug um immer alle Attribute eines XML-Knotens auf einer einzelnen Zeile zu halten. Dies ist kompakt und führt zu guten Ergebnissen beim Mergen, da die geänderten Attribute mit der - meist eindeutigen - ID des Knotens in derselben Zeile liegen.

Der Wert 0 führt zu einem speziellen neuen Format, bei dem jedes Attribut und sogar das schließende > Zeichen in jeweils einer eigenen Zeile stehen. Dadurch können zeilenbasierte Kommandos wie `git blame` die letzte Änderung jedes einzelnen Attributs anzeigen. Bei langen Zeilen erhält man damit nur die letzte Änderung des gesamten Knotens. Auch lassen sich mit diesem Format die Änderungen in Diffs zwischen zwei Versionen einer XML-Datei einfacher interpretieren. Der Nachteil dieses Formats ist die leicht erhöhte Gefahr von falschen Ergebnissen beim Merge, da der Kontext einer Änderung - üblicherweise 3 Zeilen - eventuell das ID Attribut nicht enthält.

QF-Test Versionen vor 7.0 verwenden immer eine Zeilenlänge von 78 Zeichen.

41.1.3 Darstellung

Diese Einstellungen betreffen die Darstellung des Testsuite-Baums und seiner Knoten.

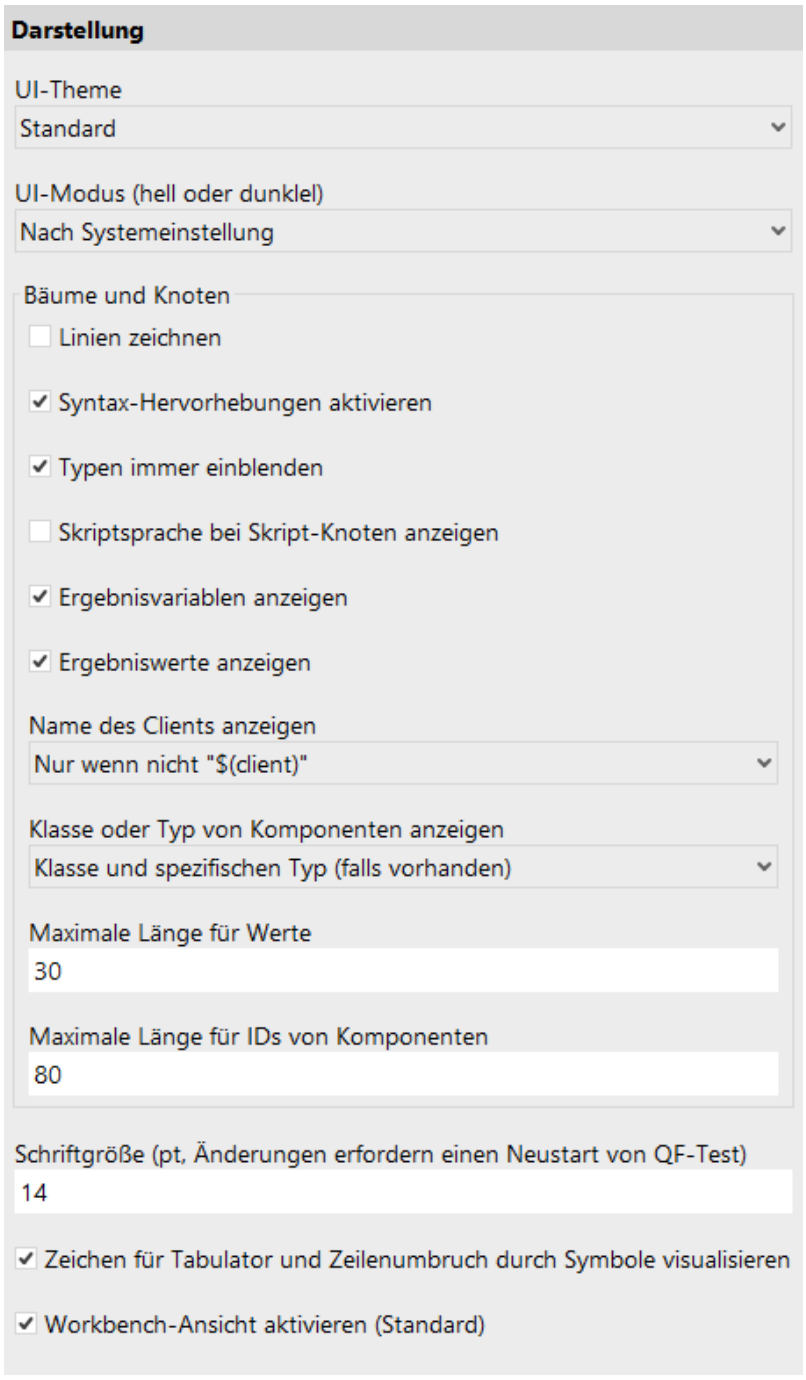


Abbildung 41.5: Darstellung

UI-Theme (User)

Diese Option legt das allgemeine Design der Benutzeroberfläche von QF-Test fest. Ein schneller Zugriff auf diese Option ist über das Menü Ansicht→UI-Theme möglich.

UI-Modus - Hell oder dunkel (User)

Diese Option legt fest, ob das aktuelle QF-Test Theme im hellen oder im dunklen Modus angezeigt wird. Die Standardeinstellung ist das Übernehmen der Vorgaben vom Betriebssystem. Ein schneller Zugriff auf diese Option ist über das Menü Ansicht→UI-Theme möglich.

Linien zeichnen (User)

Diese Option steuert, ob senkrechte Linien zwischen Baumknoten mit gleicher Einrückungstiefe angezeigt werden.

Syntax-Hervorhebungen für Baumknoten (User)

4.0+

Diese Option steuert die Aktivierung von Syntax-Hervorhebungen für Baumknoten in Testsuiten und Protokollen. Wenn sie aktiviert ist, werden spezifische Teile der Texte in den Baumknoten durch unterschiedliche Farben oder Stile hervorgehoben, z.B. Knotenname, Parameter, Client etc., was die Lesbarkeit deutlich verbessert.

Typen bei Baumknoten mit Namen einblenden (User)

7.0+

Wird diese Option deaktiviert, werden in Baumknoten von Testsuiten und Protokollen Beschriftungen wie "Testfall" ausgeblendet, sofern der jeweilige Knoten mit einem Namen versehen und sein Icon eindeutig ist.

Skriptsprache bei Skript-Knoten anzeigen (User)

7.1+

Diese Option steuert, ob die Skriptsprache eines Server-Skript(717) oder SUT-Skript(720) Knotens im Text des Testsuite-Baums angezeigt wird.

Ergebnisvariablen im Baum anzeigen (User)

7.1+

Über diese Option wird gesteuert, ob im Baumknoten der Name der Variablen, in die das Ergebnis gespeichert wird, angezeigt wird.

Ergebniswerte im Baum anzeigen (User)

7.1+

Über diese Option wird gesteuert, ob im Protokoll im Baumknoten der Wert, der

einer Ergebnisvariablen zugewiesen wurde, angezeigt wird. Die Option Maximale Länge für Werte im Baum[493] steuert die maximale Länge des angezeigten Werts.

Name des Clients im Baum anzeigen (User)

7.1+

Mit dieser Option können Sie beeinflussen, ob der Name des Clients, auf den sich der Knoten bezieht, im Testsuite-Baum angezeigt wird. Sie können "Immer" oder "Nie" auswählen, oder dass der Wert des Client Attributs nur dann angezeigt wird, wenn es nicht der Standardwert `$(client)` ist.

Klasse oder Typ von Komponenten anzeigen (User)

9.0+

Die Option legt fest, ob im Baum für einen Komponente[930]-Knoten nur die primäre Klasse angezeigt wird, der spezifische Typ oder beides. Konkret wäre das z.B. nur 'Panel', nur 'TitledPanel' oder kombiniert 'Panel:TitledPanel'.

Maximale Länge für Werte im Baum (User)

7.1+

Die Option ist nur relevant, wenn Ergebniswerte im Baum anzeigen[492] aktiviert wurde. In diesem Fall steuert sie die maximale Länge des im Baumknoten angezeigten Ergebniswerts.

Maximale Länge für IDs von Komponenten im Baum (User)

7.1+

Die Option steuert die maximale Länge der im Baumknoten angezeigten QF-Test ID der Komponente.

Schriftgröße (pt) (User)

Mit dieser Option wird die Schriftgröße definiert (gemessen in Punkten), die für die Darstellung der UI-Elemente in QF-Test verwendet wird. Eine geänderte Einstellung wird erst nach dem Neustart von QF-Test wirksam.

Zeichen für Tabulator und Zeilenumbruch durch Symbole visualisieren (User)

3.5+

Wenn diese Option gesetzt ist, werden spezielle Symbole für Tabulatoren und Zeilenumbrüche in Tabellenzeilen und in relevanten Textbereichen angezeigt.

41.1.4 Editieren

Diese Einstellungen betreffen das Editieren von Knoten im Baum und in der Detailansicht.

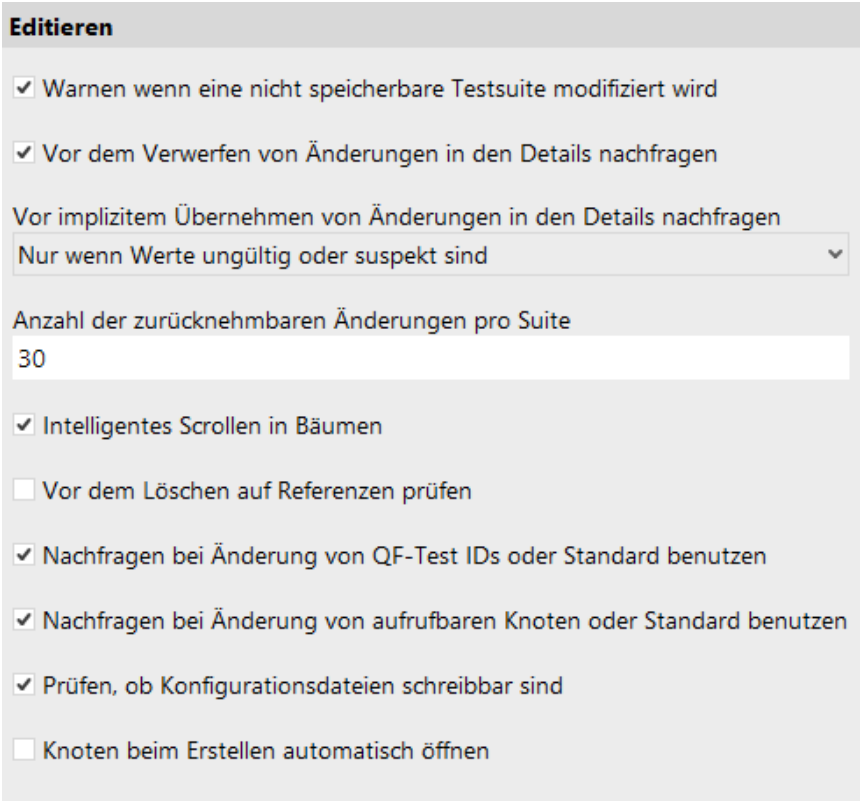


Abbildung 41.6: Editieren

Warnen wenn eine nicht speicherbare Testsuite modifiziert wird (User)

Falls das Speichern von Testsuiten nicht erlaubt ist, z.B. wenn Sie ohne Lizenz arbeiten, gibt QF-Test nach der ersten Änderung an einer Suite die Warnung aus, dass Sie Ihre Änderungen nicht speichern können. Durch Ausschalten dieser Option können Sie diese Warnung unterdrücken.

Vor dem Verwerfen von Änderungen in den Details nachfragen (User)

4.0+

Wenn Sie angefangen haben, Änderungen an einem bestehenden oder neu einzufügenden Knoten vorzunehmen und diese durch Drücken von Escape oder einen Klick auf den ”Abbrechen” Button verwerfen, fragt QF-Test nach, ob Sie die Bearbeitung wirklich abbrechen wollen. Die Rückfrage können Sie durch Deaktivieren dieser Option unterdrücken. Dabei sollten Sie sich allerdings bewusst sein, dass - insbesondere bei Skripten - im Fall eines Versehens viel Arbeit verloren gehen kann.

Vor implizitem Übernehmen von Änderungen in den Details nachfragen (User)

Wenn Sie Änderungen an den Attributen eines Knotens in der Detailansicht des Editors vornehmen und vergessen, diese mit OK zu bestätigen, bevor Sie in der Baumansicht zu einem anderen Knoten wechseln, kann QF-Test die Änderungen wahlweise automatisch übernehmen oder zunächst einen Dialog mit der Detailansicht zur Bestätigung öffnen. Folgende Optionen stehen zur Auswahl:

Immer
Werte nicht implizit übernehmen sondern immer bestätigen lassen.

Nur wenn Werte ungültig oder suspekt sind
Werte implizit ohne Bestätigung übernehmen sofern sie gültig und nicht suspekt sind. Aktuell ist ”suspekt” mit Leerzeichen am Beginn oder Ende eines Wertes gleichzusetzen, was zu subtilen Problemen führen kann, deren Ursache schwer zu erkennen ist.

Niemals
Alle gültigen Werte implizit ohne weitere Bestätigung übernehmen.

Das ausdrückliche Verwerfen von Änderungen mit Hilfe des entsprechenden Buttons oder durch drücken von Escape wird hiervon nicht beeinflusst.

Anzahl der zurücknehmbaren Änderungen pro Suite (User)

Hiermit legen Sie fest, wieviele Bearbeitungsschritte Sie in einer Testsuite oder einem Protokoll rückgängig machen können.

Intelligentes Scrollen in Bäumen (User)

Die Standardmethoden von Swing für die Interaktion mit Bäumen lassen einiges zu wünschen übrig. So führt zum Beispiel das Bewegen der Selektion zu unnötigem horizontalen Scrolling. Zusätzlich hat Swing die Tendenz, den selektierten Knoten so zu positionieren, dass nur wenig Kontext darum herum zu sehen ist.

Da die Navigation in Bäumen für QF-Test von zentraler Bedeutung ist, gibt es eine alternative Implementierung einiger dieser Methoden, die eine natürlichere Bedienung ermöglichen und sicherstellen, dass immer genug Kontextinformation um den selektierten Knoten herum zu sehen ist. Da derartige Dinge Geschmackssache sind, können Sie durch Deaktivieren dieser Option wieder zurück auf das Standardverhalten von Swing schalten.

Vor dem Löschen auf Referenzen prüfen (User)

3.4+

Wenn diese Option gesetzt ist, dann wird beim Löschen eines Knoten geprüft, ob es Referenzen auf diesen Knoten gibt. Falls es Referenzen gibt, wird eine Liste der Referenzen geöffnet.

Nachfragen bei Änderung von QF-Test IDs oder Standard benutzen (User)

3.5.3+

Wenn diese Option gesetzt ist, fragt QF-Test bei Änderungen von QF-Test IDs von Komponenten nach, ob Referenzen angepasst werden sollen. Ist diese Option nicht gesetzt, dann werden alle QF-Test IDs von Komponenten angepasst, wenn diese eindeutig sind.

Nachfragen bei Änderung von aufrufbaren Knoten oder Standard benutzen (User)

3.5.3+

Wenn diese Option gesetzt ist, fragt QF-Test bei Änderungen von aufrufbare Knoten (d.h. Prozeduren, Packages, Testfällen und Abhängigkeiten) nach, ob Referenzen angepasst werden sollen. Ist diese Option nicht gesetzt, dann werden alle Referenzen angepasst, wenn diese eindeutig sind.

Prüfen, ob Konfigurationsdateien schreibbar sind (User)

4.1.2+

Wenn diese Option gesetzt ist, prüft QF-Test beim Öffnen des Optionendialoges, ob die Konfigurationsdateien Schreibrechte besitzen. Falls dies nicht der Fall ist, wird ein Hinweisdialog angezeigt.

Knoten beim Erstellen automatisch öffnen (User)

6.1.0+

Wenn diese Option gesetzt ist werden Knoten bei der Erstellung automatisch geöffnet.

Workbench-Ansicht aktivieren (User)

8.0+

Wenn diese Option gesetzt ist, werden alle geöffneten Testsuiten in einem gemeinsamen Fenster - der Workbench - angezeigt. Die Nutzung ohne Workbench, also mit jeder Testsuiten in einem eigenen Fenster, wurde mit QF-Test Version 8.0 abgekündigt.

41.1.5 Lesezeichen

Hier können Sie Ihre Lesezeichen bearbeiten, eine Liste von Dateien und Knoten, auf die schnell über das Menü Datei→Lesezeichen zugegriffen werden kann.

4.0+ Sie können statt einer Datei auch ein Verzeichnis angeben. Bei Auswahl des entspre-

chenden Lesezeichens wird dann der Dateiauswahldialog direkt für dieses Verzeichnis geöffnet. Die QF-Test ID für den Knoten wird in diesem Fall ignoriert.

Abbildung 41.7: Lesezeichen

Sie können neue Lesezeichen zwar auch manuell erstellen, einfacher geht es aber über den Menüeintrag Datei→Lesezeichen→Aktuelle Datei hinzuzufügen, um ein Lesezeichen für eine Testsuite oder ein Protokoll zu erstellen, oder durch Auswahl des Eintrags Zu Lesezeichen hinzuzufügen im Kontextmenü eines Knotens in einer Testsuite, um ein Lesezeichen für diesen speziellen Knoten zu erstellen.

41.1.6 Externe Programme

Die folgenden Optionen legen fest, welche externe Programme QF-Test für verschiedene Zwecke aufruft.

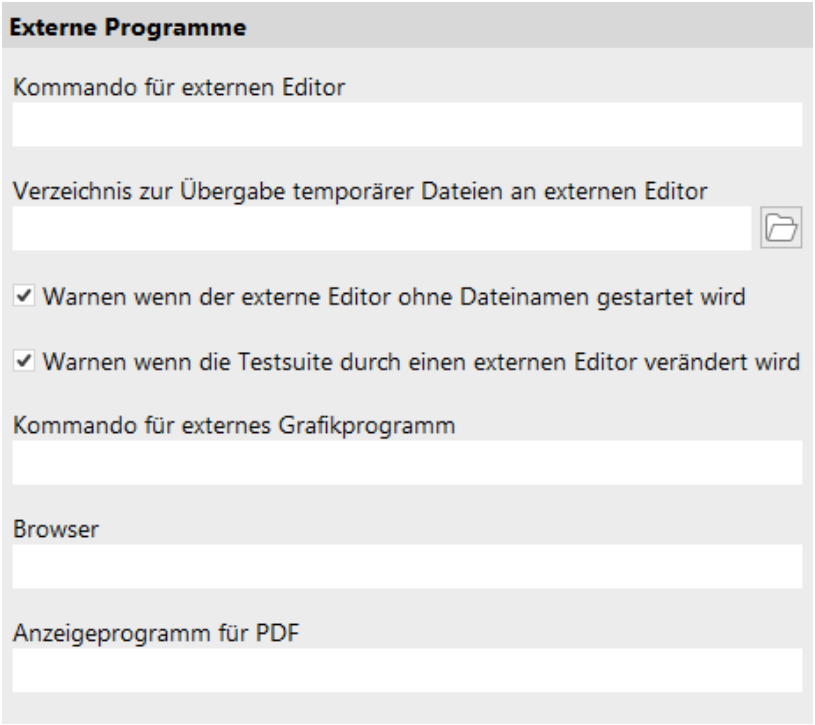


Abbildung 41.8: Optionen für Externe Programme

Kommando für externen Editor (User)

Skripte können durch Drücken von Alt-Eingabe oder Klicken des  Buttons oberhalb des Textfeldes in einem externen Editor bearbeitet werden. Dazu wird der Inhalt des Textfeldes in einer temporären Datei gespeichert und der externe Editor wird aufgerufen, um diese Datei zu bearbeiten. Es wird empfohlen, dem Skript vorher einen Namen zu geben (siehe Warnen wenn der externe Editor ohne Dateinamen gestartet wird[499]), andernfalls wird eine zufällig gewählte Zahl als Dateiname verwendet, was die Arbeit mit mehreren gleichzeitig in einem externen Editor geöffneten Skripten erschwert.

Änderungen am Skriptcode über den externen Editor werden automatisch von QF-Test übernommen. Je nach gewählten Einstellungen wird eine Warnung angezeigt, sobald das passiert (Warnen wenn die Testsuite durch einen externen Editor verändert wird[499]). Sollte der Skriptcode parallel zum externen Editor auch in QF-Test bearbeitet werden: Diese Änderungen werden ebenfalls in der temporären Datei gespeichert. Texteditoren wie jEdit sind ihrerseits in der Lage, diese zu bemerken und laden die Datei automatisch neu.

Diese Option legt das Kommando zum Aufruf des externen Editors fest. Es gibt

hierzu zwei Varianten: Die einfache Angabe einer ausführbaren Datei oder einen komplexen Befehl einschließlich Optionen. Letztere zeichnet sich dadurch aus, dass der Name der externen Datei durch den Platzhalter $(file) angegeben werden muss. Zusätzlich kann dabei über $(line) auch die aktuelle Zeile angegeben werden.

Hinweis

Die Syntax $(file)/$(line) wird ausschließlich verwendet, um nicht wieder eine neue Konvention für variable Attribute einzuführen. Es findet keine standard QF-Test Variablenexpansion für $(...) Ausdrücke statt.

Einfache Kommandos müssen nicht durch Anführungsstriche geschützt werden, z.B.:

- emacsclient
- notepad
- C:\Program Files\Crimson Editor\cedt.exe

Komplexe Kommandos benötigen eventuell Anführungsstriche, insbesondere unter Windows. Um die Anführungsstriche für das $(file) Argument kümmert sich QF-Test selbst:

- ”C:\Program Files\eclipse-3.6\eclipse.exe” –launcher.openFile $(file)
- javaw.exe -jar C:\Programme\jEdit4.2\jedit.jar -reuseview $(file)
- ”C:\Program Files\Crimson Editor\cedt.exe” $(file)
- xterm -e vi +$(line) $(file)

Ist diese Option leer, wird der Wert der Umgebungsvariablen EDITOR verwendet, sofern diese beim Start von QF-Test definiert ist.

Verzeichnis zur Übergabe temporärer Dateien an externen Editor (User)

4.1+

Über diese Option kann das Verzeichnis festgelegt werden, in das QF-Test temporäre Dateien zur Bearbeitung im externen Editor (siehe Kommando für externen Editor(498)) bereitstellt. Fall leer, werden die Dateien im benutzerspezifischen Konfigurationsverzeichnis(12) gespeichert.

Warnen wenn die Testsuite durch einen externen Editor verändert wird (User)

Bei Änderung eines Skripts durch einen externen Editor wird eine Warnung angezeigt (siehe Kommando für externen Editor(498)).

Warnen wenn der externe Editor ohne Dateinamen gestartet wird (User)

Es wird gewarnt, wenn ein namenloses Skript im externen Editor geöffnet werden soll (siehe Kommando für externen Editor(498)).

Kommando für externes Grafikprogramm (User)

Das Abbild[830] eines Check Abbild[828] Knotens kann in einem externen Grafikprogramm bearbeitet werden. Dazu wird die Grafik im PNG Format in einer temporären Datei gespeichert und das externe Grafikprogramm wird aufgerufen, um diese Datei zu bearbeiten. Nach dem Speichern der Datei und Beenden des Programms, lädt QF-Test die Daten aus der Datei zurück in das Abbild.

Diese Option legt das Kommando zum Aufruf des externen Grafikprogramms fest. Es gibt hierzu zwei Varianten: Die einfache Angabe einer ausführbaren Datei oder einen komplexen Befehl einschließlich Optionen. Letztere zeichnet sich dadurch aus, dass der Name der externen Datei durch den Platzhalter $(file) angegeben werden muss.

Hinweis

Die Syntax $(file)/$(file) wird ausschließlich verwendet, um nicht wieder eine neue Konvention für variable Attribute einzuführen. Es findet keine standard QF-Test Variablenexpansion für $(...) Ausdrücke statt.

Einfache Kommandos müssen nicht durch Anführungsstriche geschützt werden, z.B.:

- gimp
- mspaint
- C:\Windows\System32\mspaint.exe

Komplexe Kommandos benötigen eventuell Anführungsstriche, insbesondere unter Windows. Um die Anführungsstriche für das $(file) Argument kümmert sich QF-Test selbst:

- gimp –no-splash $(file)
- ”C:\Windows\System32\mspaint.exe” $(file)

HTML Browser (User)

Diese Option legt den HTML Browser fest, der für das Öffnen von HTML-Dateien (z.B. Reports oder die die kontextsensitive Hilfe) verwendet wird. Sie können ein komplexes Kommando angeben, mit ’$url’ als Platzhalter für die anzuzeigende URL, z.B.

```
netscape -remote openURL($url)
```

oder ein einfaches Kommando wie

```
firefox
```

dem dann die URL als letztes Argument übergeben wird. Ist der Eintrag leer, so wird der System-Browser verwendet.

41.1.7 Sicherungskopien

Beim Speichern einer Testsuite oder eines Protokolls ist es möglich, automatisch Sicherungskopien von bereits vorhandenen Dateien zu erstellen. Mit Hilfe der folgenden Optionen legen Sie fest, unter welchen Bedingungen Sicherungskopien angelegt werden und wie deren Name gebildet wird.

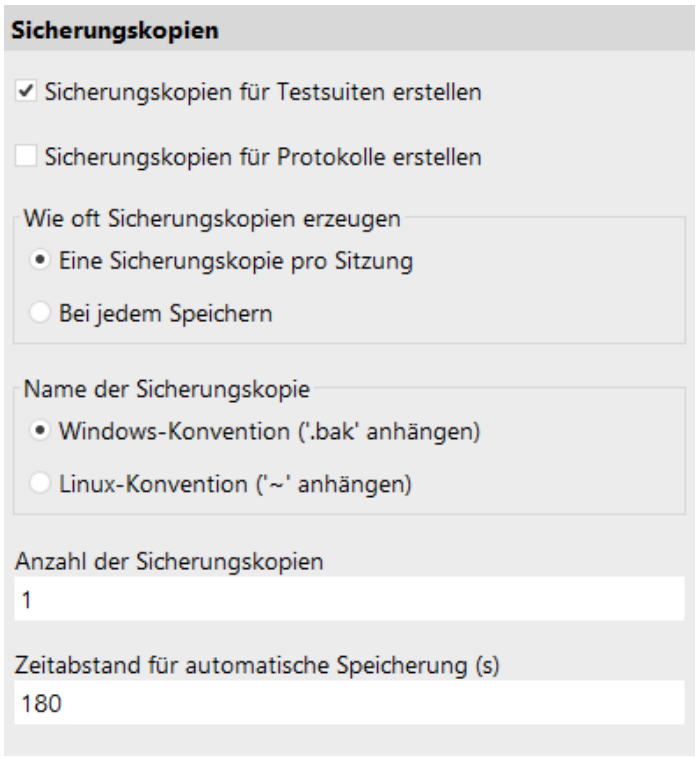


Abbildung 41.9: Optionen für Sicherungskopien

Sicherungskopien für Testsuiten erstellen (User)

Nur wenn diese Option aktiviert ist, werden Sicherungskopien von Testsuiten erstellt. Bedenken Sie bitte, wieviel Arbeit in einer guten Testsuite steckt und wie leicht die Daten zerstört werden könnten, wenn Sie keine Kopie haben. Deaktivieren Sie diese Option daher nur, wenn Sie anderweitig für eine Sicherung gesorgt haben, z.B. durch den Einsatz eines Versionskontrollsystems.

Sicherungskopien für Protokolle erstellen (User)

Ein Protokoll ist im Allgemeinen weit weniger ”wertvoll” als eine Testsuite, daher

können Sie hiermit separat festlegen, ob Sie auch beim Speichern von Protokollen Sicherungskopien erstellen wollen.

Wie oft Sicherungskopien erzeugen (User)

Es gibt zwei Varianten für die Häufigkeit, mit der Sicherungen ihrer Dateien angelegt werden:

Eine Sicherungskopie pro Sitzung bedeutet, dass nur beim ersten Speichern einer Testsuite, der Stand der letzten Sitzung gesichert wird. Bei jedem weiteren Speichern wird die neue Version überschrieben, die Kopie des alten Standes bleibt erhalten. Erst wenn Sie eine neue Testsuite laden, wird beim nächsten Speichern wieder kopiert. Diese Einstellung ist sinnvoll, wenn Sie nur eine Sicherungskopie pro Testsuite vorhalten.

Wenn Sie dagegen mehrere Sicherungen für eine Testsuite erstellen, empfiehlt es sich, bei jedem Speichern eine Kopie anzulegen.

Name der Sicherungskopie (User)

Wie vieles andere unterscheiden sich auch die Konventionen für die Namensgebung von Sicherungskopien in Linux und Windows Umgebungen. Unter Windows wird vorrangig die Endung `.bak` an den Dateinamen angehängt, während es unter Linux verschiedene Varianten gibt. Sehr häufig ist jedoch das Anhängen einer Tilde `'~'` anzutreffen.

Anzahl der Sicherungskopien (User)

Mit dieser Option legen Sie fest, wie viele Sicherungskopien Sie für jede Datei vorhalten wollen. Wenn Sie nur eine Datei wählen, wird deren Name wahlweise durch Anhängen von `.bak` oder einer Tilde `'~'` gebildet. Jedes mal, wenn eine weitere Sicherungskopie erstellt wird, wird die alte Sicherungskopie überschrieben.

Wenn Sie dagegen mehrere Sicherungskopien wählen, erhält der Name zusätzlich eine Nummer nach folgendem Schema: `bak1`, `bak2`... für die Windows Konvention und `~1~`, `~2~`... andernfalls. Die aktuellste Sicherungskopie hat immer die Nummer 1. Beim Erstellen der nächsten Kopie, wird diese zur 2 und die neue Kopie erhält die 1. Ist die Maximalzahl erreicht, werden jeweils die ältesten Sicherungskopien gelöscht.

Zeitabstand für automatische Speicherung (s) (User)

Legt den Zeitabstand fest, nach dem eine modifizierte Testsuite automatisch gesichert wird. Ein Wert von 0 schaltet die automatische Sicherung aus, andere

Werte unter ca. 20 Sekunden sind nicht sinnvoll. Protokolle werden grundsätzlich nicht automatisch gesichert. Autosave-Dateien werden im selben Verzeichnis wie die Testsuite abgelegt, oder - im Fall von neuen Testsuiten, die noch nie gespeichert wurden - im benutzerspezifischen Konfigurationsverzeichnis(12).

41.1.8 Bibliothek

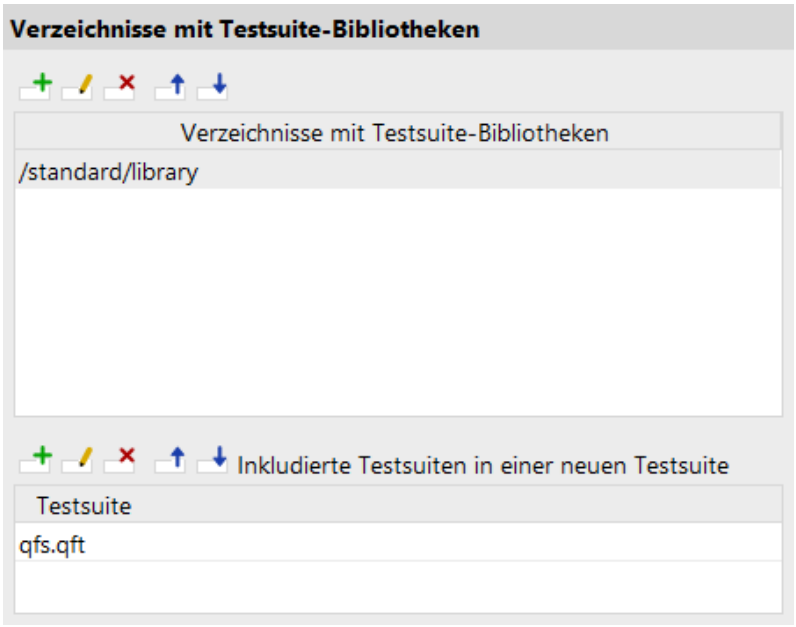


Abbildung 41.10: Bibliotheksoptionen

Verzeichnisse mit Testsuite-Bibliotheken (System)

Hierbei handelt es sich um eine Liste von Verzeichnissen, die durchsucht werden, wenn eine Referenz auf eine Testsuite als relative Datei angegeben wird und nicht relativ zur aktuellen Suite aufgelöst werden kann. Das gilt für das Name der Prozedur(676) Attribut eines Prozeduraufruf(675) Knotens oder die Referenz der QF-Test ID(931) einer Komponente ebenso, wie für Testsuiten, die über das Attribut Inkludierte Dateien(596) des Testsuite(595) Knotens eingebunden werden.

9.0+ In der Verzeichnisangabe können Umgebungsvariablen oder System-Properties referenziert werden. Die Syntax dafür lautet `${env:...}` beziehungsweise `${system:...}`. Sie können das Verzeichnis sogar während der Laufzeit ändern, indem Sie den in der Umgebungsvariablen oder System-Property hinterlegten Wert per Skript auf den neuen Wert setzen. Verwenden Sie hierzu `rc.setProperty`, das in Abschnitt 50.5(1030) beschrieben ist.

Hinweis

Die obige Syntax entspricht zwar dem QF-Test Standard für Gruppen-Variablen oder Properties. Dies ist aber ein Sonderfall bei dem nur die Gruppen `env` bzw. `system` verwendet werden können.

Das zur aktuellen Version von QF-Test gehörende `include` Verzeichnis wird immer automatisch (und unsichtbar) an das Ende des Bibliothekspfads gestellt. Dadurch ist sichergestellt, dass die Bibliothek `qfs.qft` eingebunden werden kann, ohne ihren exakten Ort zu kennen, und dass ihre Version der von QF-Test entspricht.

Hinweis

Ist das Kommandozeilenargument `-libpath <Pfad>`(983) angeben, hat es Vorrang vor dieser Option. Im interaktiven Modus wird der Wert des Kommandozeilenarguments hier angezeigt. Er wird aber nicht in der Systemkonfiguration gespeichert, es sei denn, der Wert wird manuell verändert.

Inkludierte Testsuiten in einer neuen Testsuite (System)

9.0+

Diese Option legt fest, welche Testsuiten bei einer neuen Testsuite in der Liste Inkludierte Dateien(596) eingetragen werden. Wenn nur der Testsuitename ohne Pfad angegeben wird, muss die Testsuite im gleichen Verzeichnis der inkludierenden Testsuite liegen oder in einem der Verzeichnisse mit Testsuite-Bibliotheken(503). Wenn keine Testsuite angegeben ist, wird in einer neuen Testsuite nur die Standard-Bibliothek `qfs.qft` eingebunden.

41.1.9 Lizenz

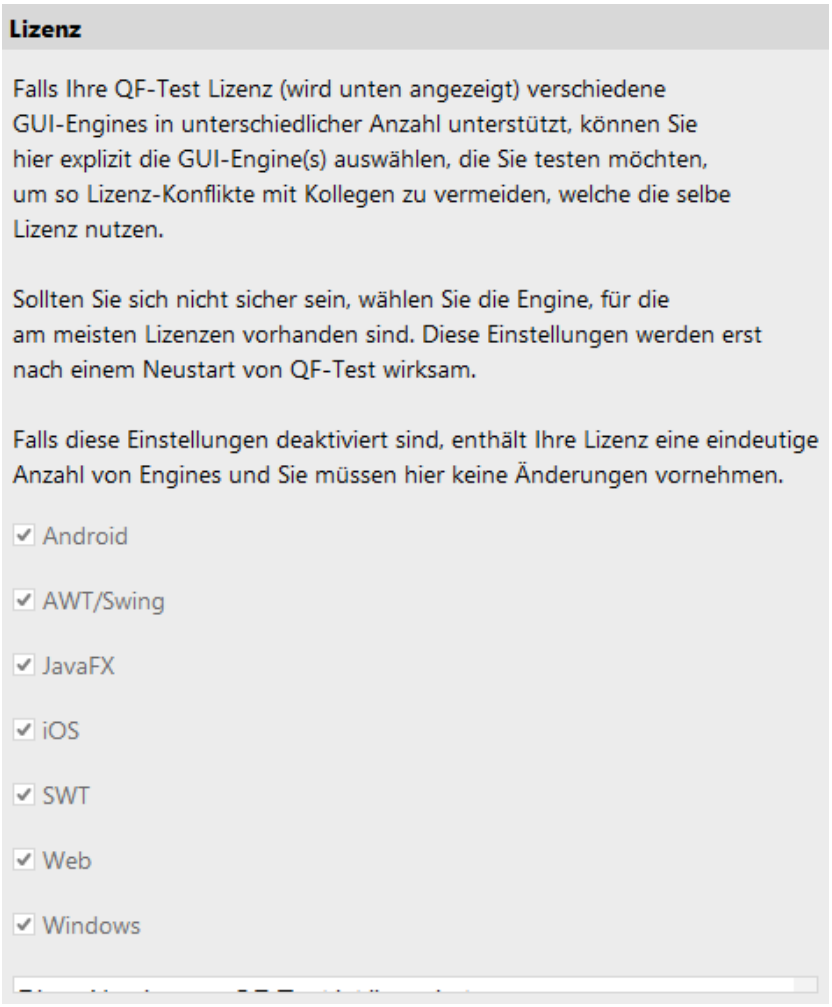


Abbildung 41.11: Lizenz Optionen

Normalerweise beinhalten QF-Test Lizenzen eine homogene Mischung von GUI-Engines. Ein Bündel von QF-Test/swing Lizenzen unterstützt z.B. nur die AWT/Swing Engine, QF-Test/suite Lizenzen beinhalten sowohl AWT/Swing als auch SWT für alle Instanzen. Für derartige Lizenzen spielen diese Lizenz-Einstellungen keine Rolle.

Ein kleines Problem entsteht im Fall von gemischten Engine-Lizenzen, bei denen eine GUI Engine nur von einem Teil der Lizenzen unterstützt wird. Ein Beispiel für eine solche Lizenz ist ein Lizenzbündel, das früher für qftestJUI angeschafft wurde, mit QF-Test 2.0 auf QF-Test/suite aktualisiert und später um weitere QF-Test/swing Lizenzen ergänzt wurde, sagen wir zwei Lizenzen für QF-Test/suite und zwei für QF-Test/swing. Eine

solche Lizenz erlaubt den Start von vier QF-Test Instanzen, von denen aber nur zwei SWT unterstützen. Der Versuch mehr als zwei Instanzen mit Nutzung der SWT Engine zu starten führt zu einem Lizenzkonflikt.

Wenn QF-Test eine solche gemischte Lizenz zum ersten mal erkennt, fragt es Sie, welche GUI Engines Sie benötigen. Die dort getroffene Entscheidung kann hier jederzeit korrigiert werden. Außerdem können Sie QF-Test mit dem Kommandozeilenargument `-engine <Engine>`[981] starten um für diese Ausführung die GUI-Engines explizit festzulegen.

41.1.10 Updates

Um die neusten Features und Fehlerbehebungen zu erhalten sucht QF-Test automatisch nach Updates. Die folgenden Optionen legen fest, ob QF-Test nach Updates suchen soll und wann über Updates informiert werden soll. Zusätzlich kann mit Hilfe des Kommandozeilenparameters -noupdatecheck 44.2.3[984] die Suche nach Updates deaktiviert werden.

Abbildung 41.12: Optionen für Updates

Automatisch nach Updates suchen (User)

Beim Start von QF-Test wird automatisch nach Updates gesucht. Wenn Sie dies nicht möchten, deaktivieren Sie diese Option.

Nach Update fragen (User)

Wenn eine neue Version verfügbar ist, zeigt QF-Test einen Hinweis mit Verweisen

auf die Release Notes und die Download-Seite an. Diese Option kann die Hinweise auf bestimmte Arten von Updates begrenzen:

- Minor Updates enthalten vorwiegend Fehlerbehebungen und kleine Verbesserungen.
- Medium Upgrades erscheinen, um neue Features einzuführen.
- Major Upgrades beinhalten große neue Features und können das Verhalten von QF-Test ändern.

41.2 Aufnahme

Über die Aufnahme-Optionen lässt sich festlegen, welche Events von QF-Test aufgezeichnet werden. Außerdem sind etliche Sonderbehandlungen von Events oder ganzen Sequenzen möglich.

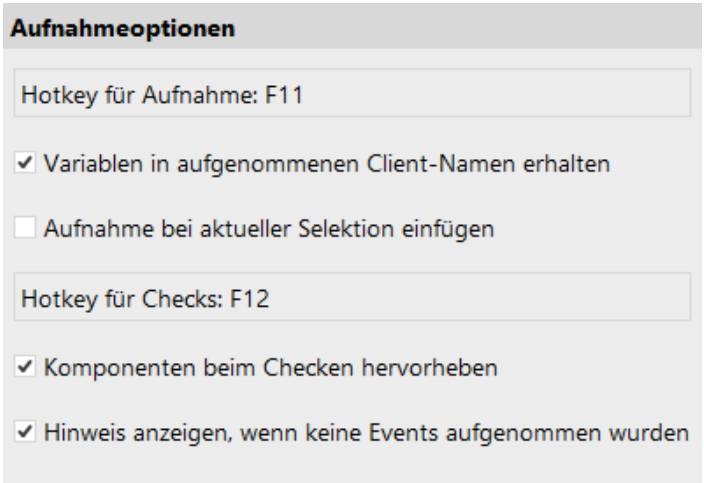


Abbildung 41.13: Aufnahmeoptionen

Initiale Schnellstarthilfe für Aufnahmeknopf zeigen (User)

Steuert die Anzeige eines initialen Fragezeichens auf dem Aufnahmeknopf, um neuen Benutzern direkt den Weg zum Schnellstart-Assistenten zu weisen.

Hotkey für Aufnahme (User)

SUT Skript Name: OPT_RECORD_HOTKEY

Der Aufnahmemodus kann mittels einer Taste direkt im SUT gestartet und gestoppt werden. Mit dieser Option legen Sie die Taste für diese Funktion fest in dem Sie in das Feld klicken und die gewünschte Tastenkombination drücken. Die Standardtaste ist F11.

Variablen in aufgenommenen Client-Namen erhalten (System)

Ein sehr nützliches Feature für fortgeschrittene Anwender: Wenn Ihr SUT aus mehreren Clients besteht oder Sie einfach nur wiederverwendbare Tests erstellen wollen, ist es sehr sinnvoll, den Clientnamen in den verschiedenen Events, Checks etc. nicht fest einzugeben, sondern über eine Variable festzulegen (z.B. `$(client)`).

Damit müssten Sie aber jedes mal, wenn Sie eine Aufnahme gemacht haben, den Namen des Clients in allen neuen Events durch die variable Angabe ersetzen, was zwar Dank Bearbeiten→Suchen und ersetzen in einem Rutsch geht, aber immer noch lästig ist. Wenn Sie diese Option aktivieren und beim Java-SUT-Client starten(724) ebenfalls die variable Syntax verwenden, wird diese bei den Aufnahmen automatisch an Stelle des expandierten Namens eingesetzt.

Aufnahme bei aktueller Selektion einfügen (User)

Ein sehr nützliches Feature: Je nachdem woran Sie gerade arbeiten, kann es sinnvoll sein, eine Aufnahme direkt an der Einfügemarkierung einzufügen - oder auch nicht. Ist diese Option gesetzt, werden alle neuen Events direkt eingefügt, andernfalls wird eine neue Sequenz mit der Aufzeichnung in den Extrasequenzen(629) angelegt.

Hotkey für Checks (User)

SUT Skript Name: OPT_RECORD_CHECK_HOTKEY
Wenn sich QF-Test im Aufnahmemodus befindet, können Sie im SUT mittels einer Taste zwischen der normalen Aufzeichnung und der Aufzeichnung von Checks hin und her schalten. Mit dieser Option legen Sie die Taste für diese Funktion fest in dem Sie in das Feld klicken und die gewünschte Tastenkombination drücken. Die Standardtaste ist F12.

Komponenten beim Checken hervorheben (User)

SUT Skript Name: OPT_RECORD_CHECK_HIGHLIGHT
Wenn Sie einen Check aufzeichnen, kann QF-Test eine kleine Hilfestellung geben, indem es die Komponente hervorhebt, über der sich der Mauszeiger gerade befindet. Dies geschieht durch Vertauschen von Vorder- und

Hintergrundfarbe der Komponente, was vereinzelt zu unerwünschten visuellen Effekten führen kann. Daher können Sie diese Funktionalität hiermit abschalten.

Hinweis anzeigen, wenn keine Events aufgenommen wurden (User)

Server Skript Name: OPT_SHOW_EMPTY_RECORDING_MESSAGE
Wenn eine Aufnahme gestartet und wieder beendet, ohne zwischendurch mit dem SUT zu interagieren, wird ein Hinweisdialog angezeigt, dass keine Events aufgenommen wurden. Durch Deaktivieren dieser Option kann dieser Dialog unterdrückt werden.

41.2.1 Folgende Events aufnehmen

Über diese Schalter lassen sich gezielt Events ein- oder ausschalten. Im Prinzip gilt hier: Finger weg, die Defaultwerte sind OK. In besonderen Fällen kann es aber doch sinnvoll sein, mit den Schaltern zu experimentieren.

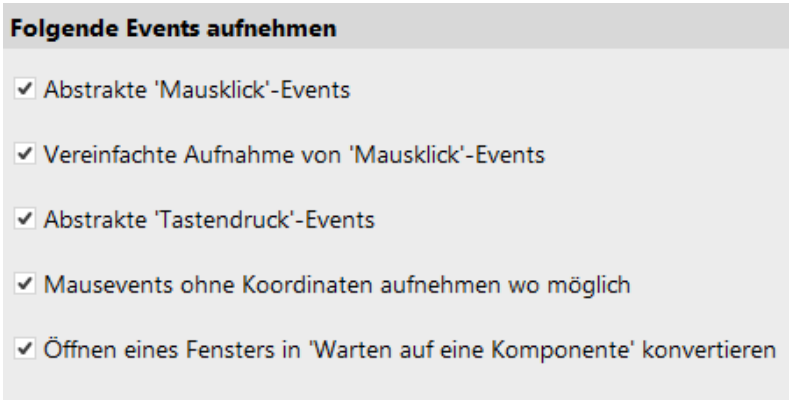


Abbildung 41.14: Optionen für die Aufnahme von Events

Abstrakte ’Mausklick’-Events (System)

Ist diese Option aktiviert, wird eine Abfolge von `MOUSE_MOVED`, `MOUSE_PRESSED`, `MOUSE_RELEASED` und `MOUSE_CLICKED` Events als ’Mausklick’ Pseudoevent aufgenommen (vgl. Abschnitt 42.8.1(775)).

Vereinfachte Aufnahme von ’Mausklick’-Events (System)

4.1+

Werden ’Mausklick’-Events aufgenommen, sollte diese Option ebenfalls aktiviert

sein. Außer für Drag&Drop und einige spezielle MOUSE_MOVED Events basiert die Aufnahme dann primär auf der Umwandlung von MOUSE_PRESSED Events in Mausklicks. Dies liefert in den meisten Fällen die besten Ergebnisse, selbst wenn QF-Test vom SUT zu wenige oder zu viele Events empfängt. Ist diese Option deaktiviert, kommt der Algorithmus von QF-Test 4.0 und älter zum Einsatz. Dieser ist einen Versuch wert, wenn eine aufgenommene Sequenz einmal nicht direkt wiedergegeben werden kann.

Abstrakte 'Tastendruck'-Events (System)

Hiermit können Sie eine Abfolge von KEY_PRESSED, KEY_TYPED und KEY_RELEASED Events (bzw. nur KEY_PRESSED und KEY_RELEASED für Funktions- und Sondertasten) automatisch als 'Tastendruck' Pseudoevent aufnehmen (vgl. Abschnitt 42.8.2(780)).

Mausevents ohne Koordinaten aufnehmen wo möglich (System)

SUT Skript Name: OPT_RECORD_REPOSITION_MOUSE_EVENTS
Für viele Arten von Komponenten und Unterelementen kommt es nicht darauf an, wo genau ein Mausevent registriert wird. Allerdings können große Werte für die X und Y Koordinaten von Mausevents(775) zu Problemen führen, wenn die Zielkomponente ein wenig kleiner wird, z.B. weil sich der Font geändert hat oder ein Fenster verkleinert wurde. Zu große Koordinaten sind auch eine häufige Fehlerursache, wenn eine aufgenommene Sequenz in eine Prozedur(672) mit variabler Zielkomponente konvertiert wird.

Ist diese Option aktiviert, ignoriert QF-Test die Koordinaten von Mausevents bei der Aufnahme, wenn es für die Zielkomponente keinen Unterschied macht, also z.B. für alle Arten von Buttons, Menüs, Zellen von Tabellen, Listeneinträgen und Baumknoten. Bei letzteren unterscheidet QF-Test zwischen Klicks auf den Knoten selbst und Klicks auf den Schalter zum Ein- und Ausklappen. Bei der Wiedergabe von Mausevents ohne Koordinaten steuert QF-Test die Mitte der Zielkomponente bzw. des Unterelements an, wobei für Unterelemente die X-Koordinate auf 5 begrenzt ist, da die Maße von Unterelementen nicht immer zuverlässig ermittelt werden können.

Öffnen eines Fensters in Warten auf Komponente(876) konvertieren (System)

Wenn während einer Aufnahme im SUT ein neues Fenster geöffnet wird, ist es oft sinnvoll, beim Abspielen der Sequenz an dieser Stelle erst zu warten, bis das Fenster erschienen ist, bevor Events an Komponenten des Fensters geschickt werden. Ist diese Option eingeschaltet, wird ein WINDOW_OPENED Event automatisch in einen Warten auf Komponente Knoten konvertiert.

Diese Option hat seit der Einführung des impliziten Wartens(555) stark an Bedeutung verloren.

Web

Für Web-Clients wird bei Aktivierung dieser Option ein Warten auf Laden des Dokuments(880) Knoten jeweils dann eingefügt, wenn das Laden eines Dokuments abgeschlossen ist. Dies ist für die zuverlässige Synchronisation beim Navigieren zu einer anderen Seite sehr wichtig.

41.2.2 Eventsequenzen packen

Machen Sie einmal ein Experiment und nehmen Sie eine kleine Sequenz auf, nachdem Sie die folgenden Optionen ausgeschaltet haben. Sie werden überrascht sein, wie viele Events generiert werden, nur um ein paar kleine Aktionen durchzuführen. Damit diese Flut beherrschbar bleibt gibt es in QF-Test verschiedene Möglichkeiten, Events zu filtern oder Eventsequenzen zusammenzupacken.

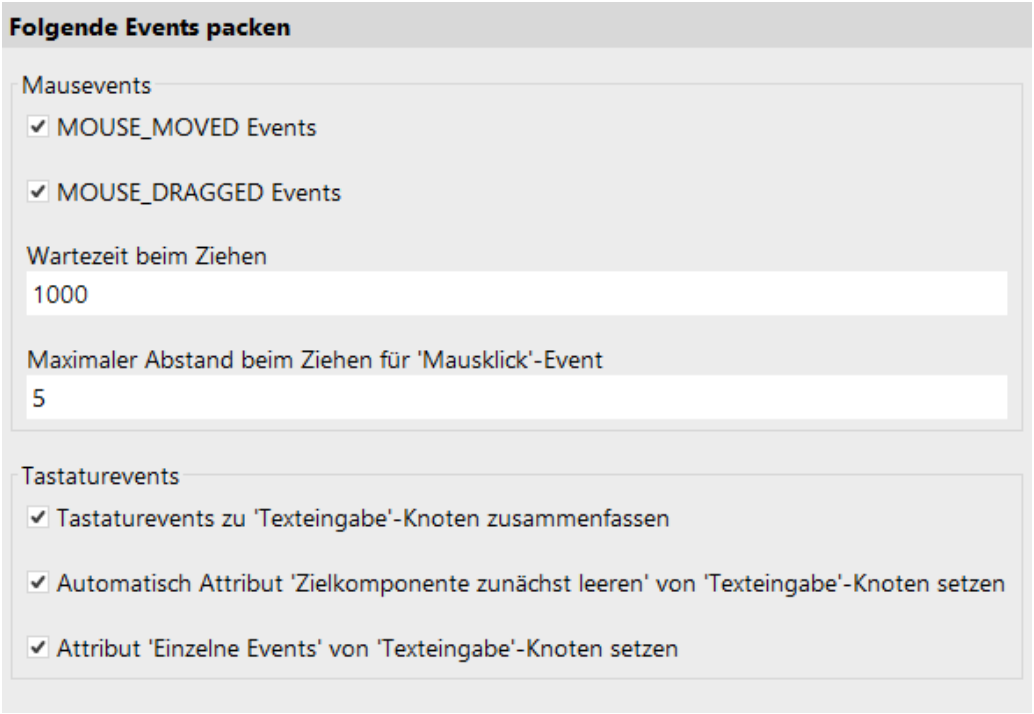


Abbildung 41.15: Optionen für das Packen von Events

`MOUSE_MOVED` Events (System)

SUT Skript Name: OPT_RECORD_PACK_MOUSE_MOVED

Gerade MOUSE_MOVED Events werden in besonders großer Zahl generiert, wenn Sie Aktionen mit der Maus durchführen. Im Normalfall ist von einer ununterbrochenen Folge von diesen Events nur der letzte interessant (eine Ausnahme wäre z.B. das Freihand-Zeichnen in einem Grafikprogramm). Ist diese Option eingeschaltet, werden die überflüssigen MOUSE_MOVED Events herausgefiltert.

Hinweis

Da auch die Events MOUSE_PRESSED, MOUSE_RELEASED und MOUSE_CLICKED Koordinaten enthalten, könnte man annehmen, dass MOUSE_MOVED Events gänzlich überflüssig sind. Dies ist nicht der Fall. Manche Java Komponenten reagieren auf einen Mausklick nur, wenn vorher die Maus tatsächlich dorthin bewegt wurde.

MOUSE_DRAGGED Events (System)

SUT Skript Name: OPT_RECORD_PACK_MOUSE_DRAGGED
Bei den MOUSE_DRAGGED Events verhält es sich ähnlich wie bei den MOUSE_MOVED Events: Nur der letzte in einer aufeinanderfolgenden Sequenz ist normalerweise von Bedeutung. Die Filterung ist daher analog zu den MOUSE_MOVED Events.

Wartezeit beim Ziehen (System)

SUT Skript Name: OPT_RECORD_MOUSE_DRAGGED_HOVER
Es gibt Situationen, in denen nicht alle MOUSE_DRAGGED Events gefiltert werden dürfen. Ein typisches Beispiel ist das Öffnen eines Untermenüs.

Hinweis

Das folgende Beispiel stimmt so nicht mehr, da QF-Test seit Version 1.05.2 keine MOUSE_MOVED oder MOUSE_DRAGGED Events mehr ”wegoptimiert”, die für das Öffnen eines Untermenüs benötigt werden. Es gibt aber vielleicht ähnlich geartete Situationen geben, bei denen die Aufnahme von Zwischenstationen beim Ziehen der Maus sinnvoll sein kann.

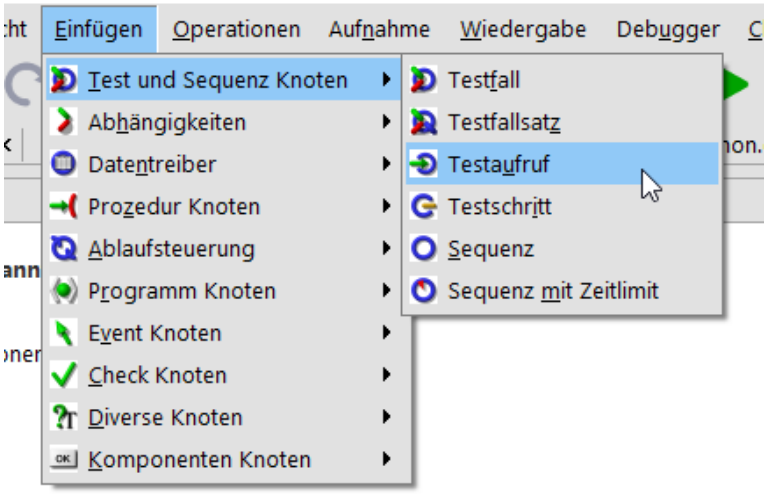


Abbildung 41.16: Ziehen in ein Untermenü

Wie das obige Bild illustriert, könnten Sie in QF-Test einen neuen Testaufruf Knoten in eine Suite einfügen, indem Sie zunächst auf das Eingabe Menü klicken, dann bei gedrückter Maustaste den Zeiger auf Test- und Sequenz-Knoten bewegen, so dass das Untermenü ausklappt und letztlich den Zeiger auf Testaufruf bewegen und dort loslassen. Sofern die obige Option zum Packen von MOUSE_DRAGGED Events gesetzt ist, würde QF-Test das gesamte Ziehen zu einem einzigen MOUSE_DRAGGED Event zusammenfassen, nämlich dem letzten auf den Testaufruf Eintrag im Untermenü. Das Abspielen dieser Sequenz würde scheitern, weil das Ziehen auf den Test- und Sequenz-Knoten Eintrag des Menüs übersprungen wird, so dass das Untermenü nicht ausklappt.

Um dem vorzubeugen, können Sie beim Aufzeichnen der Sequenz eine Weile mit dem Mauszeiger auf dem Test- und Sequenz-Knoten Eintrag des Menüs verharren. Daran erkennt QF-Test dass es einen zusätzlichen MOUSE_DRAGGED Event aufnehmen soll. Wie lange diese Ruhepause sein muss, legen Sie mit dieser Option fest.

Maximaler Abstand beim Ziehen für 'Mausklick'-Event (System)

Es kommt vor, dass der Mauszeiger unabsichtlich zwischen dem Drücken und Loslassen einer Maustaste bewegt wird. Je nach JDK Version und der Größe der Bewegung kann das als MOUSE_DRAGGED Event registriert werden. Für kleine Bewegungen kann QF-Test diesen ignorieren um trotzdem einen 'Mausklick' Event zu erstellen. Diese Option legt fest, welche Distanz QF-Test noch als einen

Klick interpretiert. Ein MOUSE_DRAGGED Event, der darüber hinaus geht, wird unverändert übernommen.

Tastaturevents zu Texteingabe Knoten zusammenfassen (System)

Auch für die simple Eingabe eines kurzen Textes werden massenhaft Events generiert. Würde man nur mit den normalen KEY_PRESSED, KEY_TYPED und KEY_RELEASED Events arbeiten, wäre dies nicht nur unübersichtlich, es wäre auch unmöglich, variablen Text zu erzeugen.

Ist diese Option gesetzt, werden Folgen von Tastaturevents in eine Texteingabe(784) umgewandelt, sofern die Komponente ein Textfeld ist (genauer: von der Klasse java.awt.TextField oder javax.swing.text.JTextField). Dabei werden nur normale Tasten (auch in Kombination mit der Shift Taste) berücksichtigt. Sondertasten und Kombinationen mit Strg oder Alt werden nicht gepackt.

Beim Abspielen einer gepackten Textsequenz werden nur KEY_TYPED Events generiert, die KEY_PRESSED und KEY_RELEASED Events dagegen unterdrückt.

Automatisch Attribut 'Zielkomponente zunächst leeren' von Texteingabe Knoten setzen (System)

Diese Option bestimmt den Wert des Attributs Zielkomponente zunächst leeren(786) eines neu aufgenommenen Texteingabe(784) Knotens. Ist die Option nicht gesetzt, wird auch das Attribut nicht gesetzt, andernfalls wird das Attribut genau dann aktiviert, wenn das Textfeld vor Beginn der Eingabe leer war.

Attribut 'Einzelne Events' von Texteingabe Knoten setzen (System)

Hiermit wird der Wert des Attributs Einzelne Events wiedergeben(786) in neu aufgenommenen Texteingabe(784) Knoten festgelegt. Ist die Option gesetzt, wird das Attribut ebenfalls gesetzt, und umgekehrt. Der sichere Weg ist, diese Option eingeschaltet zu lassen. Bei einem typischen SUT, das keine eigenen KeyListener für Textfelder verwendet, sollte es keine Probleme geben, wenn das Attribut deaktiviert wird, um die Wiedergabe von Texteingabe Knoten zu beschleunigen.

41.2.3 Komponenten

Allgemeine Informationen zur den Einstellungen zur Aufnahme von Klassen:

QF-Test kann Klassen von Komponenten auf unterschiedliche Arten aufzeichnen. Hierzu gliedert QF-Test Komponentenklassen in unterschiedliche Kategorien. Diese Kategorien sind die konkrete Klasse, die technologiespezifische Basisklasse, die generi-

sche Klasse sowie der spezielle Typ der generischen Klasse. Jede Kategorie wird unter Weitere Merkmale(933) aufgezeichnet.

Die Option Generische Klassen für Komponenten aufzeichnen(518) ist die Standardeinstellung und bewirkt, dass generische Klassen aufgezeichnet werden. Mit dieser Einstellung können Ihre Tests auch für unterschiedliche Technologien mit minimalen Anpassungen lauffähig gemacht werden.

Swing

Wenn Sie eine Java-Anwendung haben, die nur auf einer Technologie aufbaut und lieber die ”echten” Java-Klassen sehen möchten, können Sie auch ohne generische Klassen arbeiten. Hierfür sollten Sie aber die Option Nur Systemklassen aufnehmen(519) einschalten. Diese Option bewirkt, dass jeweils die Basisklasse der jeweiligen Java-Technologie aufgezeichnet wird und nicht die konkrete abgeleitete Klasse. Falls Sie diese Option ausschalten, zeichnet QF-Test die konkrete Klasse des Objektes auf, welches zwar eine sehr gezielte Wiedererkennung auf Klassenebene bringt, allerdings bei Änderungen durch Refactoring der Klassen zu Anpassungsaufwand führt. Falls die konkreten Klassen obfuskiert sind, sollte diese Option auf keinen Fall aktiviert werden.

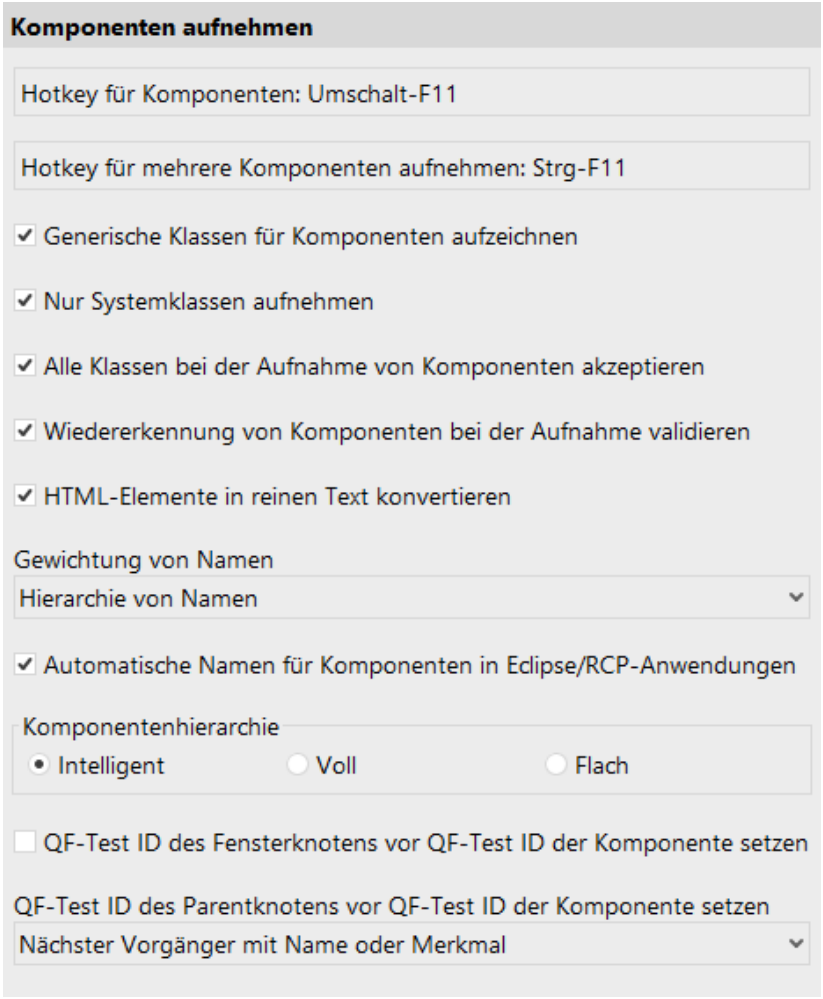


Abbildung 41.17: Option für die Aufnahme von Komponenten

Hotkey für Komponenten (User)

SUT Skript Name: OPT_RECORD_COMPONENT_HOTKEY
Hiermit legen Sie eine Taste bzw. Tastenkombination fest, mit der Sie das SUT in einen speziellen Modus ähnlich dem Aufzeichnungsmodus für Checks schalten können. Um den Wert zu ändern klicken Sie auf den aktuellen Eintrag (hierbei handelt um ein interaktives Feld) und drücken nun die gewünschte Taste bzw. Tastenkombination. Um das Feld zu verlassen, drücken Sie entweder die Tab-Taste oder setzen Sie den Fokus mit der Maus auf ein anderes Feld. Die Standard-Tastenkombination ist Shift-F11 für Window/Linux bzw. ⇧-F11 für Mac.

In diesem Modus wird eine Komponente, die Sie mit der Maus anklicken, an QF-Test übermittelt. Dort wird falls noch nicht vorhanden, wie nach einer Aufnahme ein entsprechender Knoten unterhalb des Fenster und Komponenten(942) Knotens eingefügt. Außerdem steht die QF-Test ID der Komponente im Clipboard zur Übernahme mit Strg-V bereit.

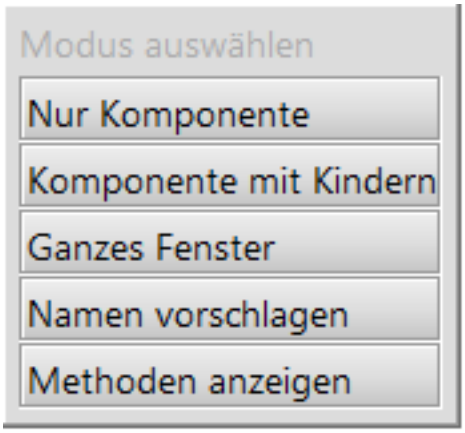


Abbildung 41.18: Popupmenü zum Aufnehmen von Komponenten

Sie können auch eine ganze Hierarchie von Komponenten in einem Schritt aufzeichnen. Klicken Sie hierzu, nachdem Sie in diesen speziellen Modus geschaltet haben, mit der rechten Maustaste auf eine beliebige Komponente. Sie erhalten dann ein Popupmenü, in dem Sie vier Möglichkeiten zur Auswahl haben:

Nur Komponente
Diese Auswahl entspricht dem Klick mit der linken Maustaste. Es wird nur die gewählte Komponente aufgezeichnet.

Komponente mit Kindern
Es werden die ausgewählte Komponente und alle darin enthaltenen Komponenten aufgezeichnet.

Ganzes Fenster
Dient zur Aufnahme aller Komponenten im gesamten Fenster.

Namen vorschlagen
Dies ist eine besondere Funktion zur Unterstützung der Zusammenarbeit zwischen Testern und Entwicklern bei der Vergabe von Namen für Komponenten mittels `setName`. Sie zeichnet alle Komponenten im gesamten Fenster auf und legt diese in einer eigenen Testsuite ab. Alle Komponenten ohne eigenen Namen, für die ein Name zur Verbesserung der Testsicherheit sinnvoll erscheint, werden markiert. Hierzu erhalten diese einen Namen im Format ”SUGGESTED NAME (n): Vorschlag”. Die

fortlaufende Zahl in Klammern dient nur zur Vermeidung von Duplikaten. Den Vorgeschlagenen Namen setzt QF-Test aus der Klasse der Komponente und sonstigen verfügbaren Informationen zusammen. Er sollte nur als Anregung verstanden werden.

Methoden anzeigen

Hierbei handelt es sich um eine weitere Sonderfunktion, die ein Fenster mit den Attributen und Methoden der Klasse der selektierten Komponente öffnet. Weitere Informationen hierzu finden Sie in Abschnitt 5.12(107).

Wenn Sie eine einzelne Komponente abfragen wollen, drücken Sie im SUT einfach die hier eingestellte Taste. Nachdem Sie eine Komponente angeklickt haben und diese übermittelt wurde, schaltet das SUT von selbst in den vorherigen Modus zurück. Möchten Sie dauerhaft in diesen Modus bleiben verwenden Sie die Tastenkombination aus Abschnitt 5.12(107). Zum Beenden drücken Sie einfach noch einmal den ”Hotkey”.

Wenn Sie mehrere Testsuiten geöffnet haben, müssen Sie mit Hilfe des Menüeintrags Aufnahme→Komponenten empfangen festlegen, an welche Suite die auf diese Weise aufgenommenen Komponenten übermittelt werden sollen.

Hotkey für mehrere Komponenten aufnehmen (User)

SUT Skript Name: OPT_RECORD_COMPONENT_CONTINUE_HOTKEY
Hiermit legen Sie eine Taste bzw. Tastenkombination fest, mit der Sie das SUT in den Komponentenaufnahme-Modus schalten können. Um den Wert zu ändern klicken Sie auf den aktuellen Eintrag (hierbei handelt um ein interaktives Feld) und drücken nun die gewünschte Taste bzw. Tastenkombination. Um das Feld zu verlassen, drücken Sie entweder die Tab-Taste oder setzen Sie den Fokus mit der Maus auf ein anderes Feld. Die Standard-Tastenkombination ist STRG-F11 für Window/Linux bzw. ^-F11 für Mac.

Der Unterschied zu Hotkey für Komponenten(516) ist, dass Sie dauerhaft in diesen Modus bleiben um mehrere Komponenten aufzunehmen zu können. Zum Beenden drücken Sie einfach noch einmal den ”Hotkey”.

Generische Klassen für Komponenten aufzeichnen (System)

4.0+

SUT Skript Name: OPT_RECORD_COMPONENT_GENERIC_CLASS
Wenn möglich weist QF-Test Komponenten Abschnitt 5.4.1(62) wie ”Button”, ”Table” oder ”Tree” zusätzlich zu den Java, DOM oder Framework-spezifischen Klassennamen wie ”javax.swing.JButton”, ”javafx.scene.control.Button”, ”INPUT” oder ”X-BUTTON” zu. Diese generischen Klassennamen sind verständlicher und robuster, verbessern die Kompatibilität zwischen verschiedenen Arten von Oberflächen und unterstützen die Entwicklung von generischen Prozeduren.

Generische Klassennamen können für die Wiedererkennung von Komponenten und zur Registrierung von Resolvern verwendet werden. Ist diese Option gesetzt, zeichnet QF-Test den generischen Klassennamen auf, sofern vorhanden. Weitere Informationen finden Sie in im einleitenden Teil dieses Abschnitts.

Nur Systemklassen aufnehmen (System)

SUT Skript Name: OPT_RECORD_COMPONENT_SYSTEM_CLASS_ONLY
Ist diese Option gesetzt, nimmt QF-Test nur Standardklassen für Komponenten[930] auf. Für kundenspezifische Klassen wird die Klassenhierarchie nach oben abgearbeitet, bis eine Standardklasse erreicht ist. Aktivieren Sie diese Option, wenn sich die Namen der GUI-Klassen Ihres Programms hin und wieder ändern. Weitere Informationen finden Sie im einleitenden Teil dieses Abschnitts.

Hinweis Sie müssen diese Option auf jeden Fall aktivieren, wenn Sie die jar Archive Ihrer Applikation durch Obfuscation schützen wollen, oder wenn die GUI Klassen mit einem eigenen ClassLoader geladen werden.

Web Diese Option ist für Web-Clients ohne Bedeutung.

Alle Klassen bei der Aufnahme von Komponenten akzeptieren (System)

4.0+

SUT Skript Name: OPT_RECORD_TOLERANT_CLASS_MATCH
Für die Kompatibilität mit älteren QF-Test Versionen, die keine generischen Klassen kannten, zieht QF-Test bei der Aufnahme zum Vergleich mit bestehenden Komponenten mehrere Klassen einer Komponente heran, die konkrete, die generische und die Systemklasse. Dies ist sehr hilfreich, wenn Sie Ihre alten Komponenten so weit möglich erhalten wollen. Wenn Sie stattdessen lieber generell neue Komponenten basierend auf generischen Klassen aufnehmen möchten, schalten Sie diese Option aus. Für Komponenten, die zum ersten mal aufgenommen werden, richtet sich die Klasse immer nach den beiden vorhergehenden Optionen Generische Klassen für Komponenten aufzeichnen[518] und Nur Systemklassen aufnehmen[519].

Wiedererkennung von Komponenten bei der Aufnahme validieren (System)

3.5+

SUT Skript Name: OPT_VALIDATE_RECORDED_COMPONENTS
Falls Komponenten im SUT nicht-eindeutige Namen zugewiesen wurden, kann QF-Test diese Komponenten mit Hilfe des Weiteren Merkmals[933] `qfs:matchindex` unterscheiden, welches den Index innerhalb der Komponenten mit gleichem Namen angibt. Ist diese Option gesetzt, prüft QF-Test den Namen der Komponente bereits bei der Aufnahme auf Eindeutigkeit und versucht `qfs:matchindex` korrekt zu setzen.

Hinweis Sie sollten diese Option nur dann deaktivieren, wenn Sie sicher sind, dass Namen

von Komponenten weitestgehend eindeutig sind und die Validierung die Performance bei der Aufnahme spürbar beeinträchtigt.

HTML Elemente in reinen Text konvertieren (System)

Swing

SUT Skript Name: OPT_RECORD_COMPONENT_CONVERT_HTML
Swing unterstützt HTML-Auszeichnungen für verschiedene Arten von Labels, Buttons und Unterelementen von komplexen Komponenten. Bei der Identifizierung und Validierung von Komponenten sind die HTML-Tags oft im Weg. Ist diese Option gesetzt, konvertiert QF-Test HTML in normalen Text, indem es alle HTML-Tags entfernt, so dass nur der reine Textinhalt erhalten bleibt.

Gewichtung von Namen (Aufnahme) (System)

Server (automatisch weiter an SUT) Skript Name: OPT_RECORD_COMPONENT_NAME_OVERRIDE
Mögliche Werte: VAL_NAME_OVERRIDE_EVERYTHING, VAL_NAME_OVERRIDE_HIERARCHY, VAL_NAME_OVERRIDE_PLAIN

Hinweis

Es gibt zwei Varianten dieser Option, die sehr eng miteinander verknüpft sind. Diese Variante ist während der Aufnahme aktiv, die andere[548] bei der Wiedergabe. Natürlich sollten beide Optionen immer den selben Wert haben - mit einer Ausnahme: Wenn Sie von einer Einstellung zu einer anderen wechseln wollen, müssen eventuell Komponenten in QF-Test aktualisiert werden. Bei diesem Prozess ist es notwendig, zunächst die Einstellung für die Wiedergabe auf dem alten Wert zu lassen und nur die Aufnahme Option umzustellen. Denken Sie aber unbedingt daran, nach Abschluss der Aktualisierung auch die Wiedergabe Option umzustellen.

Diese Option legt fest, welches Gewicht dem Namen bei der Aufnahme von Komponenten beigemessen wird. Folgende Einstellungen sind möglich:

Name übertrifft alles
Dies ist die wirksamste und flexibelste Möglichkeit, Komponenten zu erkennen. Sie setzt allerdings voraus, dass die Namen der Komponenten zumindest pro Fenster eindeutig sind. Wenn diese Eindeutigkeit gegeben ist, verwenden Sie diese Einstellung.

Web

Verwenden Sie diesen Wert nicht bei Webseiten mit Frames. Für diese ist ”Hierarchie von Namen” besser geeignet.

Hierarchie von Namen
Diese Einstellung sollten Sie verwenden, wenn Namen zwar nicht in jedem Fenster eindeutig vergeben sind, aber Komponenten mit gleichen Namen zumindest in unterschiedlichen Komponenten mit verschiedenen Namen enthalten sind, so

dass sich eine Eindeutige Namenshierarchie ergibt. Damit ist die Wiedererkennung immer noch sehr tolerant gegenüber Veränderungen. Erst wenn Sie eine benannte Komponente in eine andere benannte Komponente verschieben, muss die Testsuite an diese Veränderung angepasst werden.

Normales Attribut
Falls es Komponenten mit identischen Namen im SUT gibt, die zudem in der gleichen Parent Komponente liegen, bleibt nur noch diese Einstellung. Der Name spielt damit immer noch eine wichtige Rolle, aber kaum mehr als das Merkmal(932) Attribut.

Automatische Namen für Komponenten in Eclipse/RCP-Anwendungen (System)

SWT

SUT Skript Name: OPT_RECORD_COMPONENT_AUTOMATIC_RCP_NAMES
Eclipse und Anwendungen, die auf der Rich Client Platform (RCP) basieren, verfügen über ein komplexes GUI mit Unterstützung für wechselnde Perspektiven. Bei einem solchen Wechsel werden die Komponenten neu arrangiert, was die Wiedererkennung für QF-Test schwierig macht, wenn nicht wenigstens für die wichtigsten Komponenten Namen gesetzt werden. Zusätzlich kompliziert wird es dadurch, dass die Struktur der Komponenten nicht der optischen Darstellung entspricht. Die Komponenten sind stattdessen relativ flach in der Workbench angeordnet. Positiv ist dagegen, dass RCP basierte Anwendungen über eine einheitliche innere Struktur basierend auf `Views` und `Editors` verfügen, von denen viele einen Namen besitzen.

Falls diese Option aktiviert ist, versucht QF-Test automatisch die GUI-Komponenten mit Ihren RCP Gegenstücken zu assoziieren und auf dieser Basis Namen für die Komponenten zu ermitteln. Dies kann die Wiedererkennung der Komponenten drastisch verbessern. Falls einige dieser automatisch ermittelten Namen allerdings nicht dauerhaft stabil sein sollten, können diese auch ein Hindernis darstellen. In diesem Fall kann für die Betroffenen Komponenten ein Name gesetzt werden, entweder mittels `setData` wie in Kapitel 5(47) beschrieben, oder mit Hilfe eines `NameResolvers` wie in Abschnitt 54.1.7(1163) beschrieben. Beide Methoden haben Vorrang vor automatisch generierten Namen.

Komponentenhierarchie (System)

Server Skript Name: OPT_RECORD_COMPONENT_HIERARCHY
Mögliche Werte: VAL_RECORD_HIERARCHY_INTELLIGENT,
VAL_RECORD_HIERARCHY_FULL,
VAL_RECORD_HIERARCHY_FLAT
QF-Test bietet verschiedene Sichtweisen auf die Komponenten des SUT.

In der flachen Sicht sind alle Komponenten eines Fensters gleichwertig unter diesem angeordnet. Für ein einfaches Fenster mag diese Sicht genügen, für komplexere Fenster geht nicht nur die Übersicht verloren, sondern auch wertvolle Strukturinformation für die Wiedererkennung.

Das Gegenteil zur flachen Sicht ist die volle Hierarchie. Hier werden alle Komponenten des SUT 1:1 übernommen und in einer entsprechenden Baumstruktur angeordnet. Diese Darstellung kann wertvolle Informationen für Entwickler liefern, hat aber den Nachteil, dass das Einfügen einer weiteren Zwischenkomponente im SUT die Strukturinformationen ungültig macht und die Wiedererkennung aller Childkomponenten dieser neuen Komponente verhindert.

Die Auswahl "Intelligent" liefert einen Kompromiss aus den beiden Extremwerten. Hiermit werden nur die "interessanten" Komponenten aus der Hierarchie des SUT übernommen. Komponenten wie `JPanel`, die ausschließlich der Strukturierung dienen, werden nicht aufgezeichnet. Die Verschachtelungstiefe des Baums wird dadurch deutlich reduziert. Im Moment ist die Entscheidungslogik, welche Komponenten als "interessant" anzusehen sind, hart verdrahtet. In späteren Versionen wird sie konfigurierbar sein.

QF-Test ID des Fensterknotens vor QF-Test ID der Komponente setzen (System)

Server Skript Name: OPT_RECORD_COMPONENT_PREPEND_WINDOW_ID
Ist diese Option aktiviert stellt QF-Test bei der Aufnahme einer Komponente(930) deren QF-Test ID(931) die QF-Test ID des zugehörigen Fenster(919) Knotens voran. Dies ist sinnvoll zur Unterscheidung von Komponenten mit gleichem Namen in verschiedenen Fenstern.

QF-Test ID des Parentknotens vor QF-Test ID der Komponente setzen (System)

Server Skript Name: OPT_RECORD_COMPONENT_PREPEND_PARENT_ID
Mögliche Werte: VAL_RECORD_COMPONENT_PREPEND_PARENT_ALWAYS, VAL_RECORD_COMPONENT_PREPEND_PARENT_NAMED, VAL_RECORD_COMPONENT_PREPEND_PARENT_FEATURE, VAL_RECORD_COMPONENT_PREPEND_PARENT_NEVER
Wenn bei einer Aufnahme eine neue Komponente(930) angelegt wird, vergibt QF-Test automatisch eine QF-Test ID(931). Dieser ID kann wahlweise die QF-Test ID eines direkten oder indirekten Parentknotens der Komponente - durch einen Punkt getrennt - vorangestellt werden. Dies ist vor allem nützlich, um häufig auftretende Komponenten, die keinen eigenen Namen haben, unterscheidbar zu machen.

Beispiel: Angenommen es gibt zwei `JScrollPanes`, von denen eine "TreeScrollPane" und die andere "DetailScrollPane" heißt. Ohne diese Funktion würden die

beiden vertikalen Scrollbars die QF-Test IDs ”scrollbarVertical” und ”scrollbarVertical2” erhalten. Mit dieser Funktion erhalten sie dagegen die QF-Test IDs ”TreeScrollPane.scrollbarVertical” und ”DetailScrollPane.scrollbarVertical”. Damit können Sie bei Events sofort erkennen, worauf sich diese beziehen.

Es gibt vier mögliche Einstellungen:

- ”Niemals” schaltet diese Funktionalität aus.
- ”Nächster Vorgänger mit Name” ist eine sinnvolle Einstellung, wenn Ihre Entwickler allen wichtigen Komponenten mit der Java-Methode `setName` einen Namen gegeben haben. Eine Komponente, die selbst keinen Namen hat, bekommt den Namen des ersten seiner direkten und indirekten Parentknoten vorangestellt, der einen eigenen Namen hat.
- Wenn Sie nicht - oder nur eingeschränkt - mit `setName` arbeiten, ist die Einstellung ”Nächster Vorgänger mit Name oder Merkmal” besser geeignet. Neben dem Namen eines Parentknotens kann auch dessen Merkmal ausschlaggebend sein.
- ”Immer” ist nur dann sinnvoll, wenn Sie die Option Komponentenhierarchie(521) auf den Wert ”Flach” eingestellt haben. Hierbei wird jeder QF-Test ID die QF-Test ID des direkten Parentknotens vorangestellt, was bei tiefer Verschachtelung von Komponenten zu unbrauchbar langen QF-Test IDs führt.

41.2.4 Unterelemente

Bei komplexen Komponenten wie Tabellen oder Bäumen können Events wahlweise so aufgenommen werden, dass sie sich auf Unterelemente der Komponenten beziehen und ihre Koordinaten relativ zu diesen Unterelementen sind.

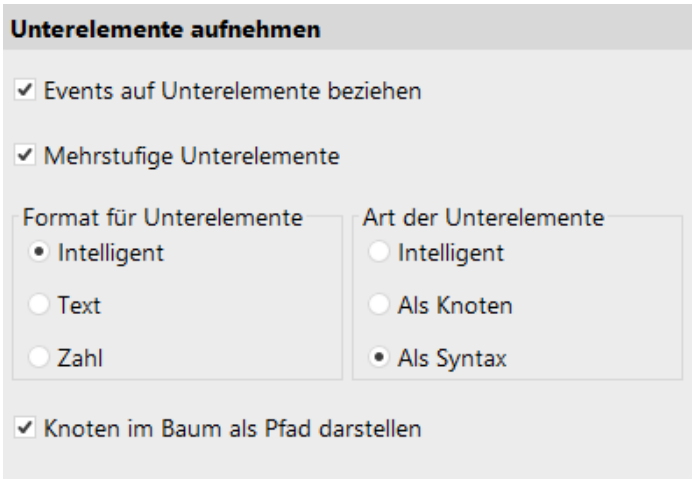


Abbildung 41.19: Option für die Aufnahme von Unterelementen

Events auf Unterelemente beziehen (System)

SUT Skript Name: OPT_RECORD_SUBITEM
Die oben beschriebene Funktionalität wird mit diesem Schalter aktiviert. Ist diese Option ausgeschaltet, beziehen sich neu aufgenommene Events immer auf die Komponente, nicht auf ein Unterelement.

Mehrstufige Unterelemente (System)

4.0+

Server (automatisch weiter an SUT) Skript Name: OPT_RECORD_SUBITEM_MULTILEVEL
Über diese Option können Sie mehrstufige Unterelemente komplett ausschalten (auch für die Wiedergabe), was Sie aber nur tun sollten, wenn Sie Probleme mit alten Testsuiten haben, die ungeschützte Sonderzeichen wie '@' oder '%' im textuellen Index von Unterelementen enthalten. Allerdings wäre es in diesem Fall vorzuziehen, die Testsuiten zu aktualisieren und die Unterelemente korrekt zu schützen, ggf. über die spezielle Variablensyntax ${quoteitem:...} (vgl. Abschnitt 6.8(127)).

Format für Unterelemente (System)

SUT Skript Name: OPT_RECORD_SUBITEM_FORMAT
Mögliche Werte: VAL_RECORD_SUBITEM_FORMAT_INTELLIGENT,
VAL_RECORD_SUBITEM_FORMAT_TEXT,
VAL_RECORD_SUBITEM_FORMAT_NUMBER
Wenn Events sich auf Unterelemente einer Komponente beziehen, können Sie mit diesem Schalter festlegen, ob der Text der Elemente oder deren Position aufgenommen wird. Der Index für das Element wird in der entsprechenden Form als Text(938) bzw. als Zahl(938) angelegt.

Mittels des Schalters "Intelligent" können Sie die Entscheidung QF-Test übertragen. Der Index wird dann als Text erstellt, sofern der Name des Elements innerhalb der komplexen Komponente eindeutig ist. Andernfalls wird der Index als Zahl angelegt.

Art der Unterelemente (System)

Server Skript Name: OPT_RECORD_SUBITEM_TYPE
Mögliche Werte: VAL_RECORD_SUBITEM_TYPE_INTELLIGENT,
VAL_RECORD_SUBITEM_TYPE_NODE,
VAL_RECORD_SUBITEM_TYPE_SYNTAX
Hiermit entscheiden Sie, ob bei der Aufnahme eines Events für ein Unterelement ein Element(936) Knoten für dieses Unterelement in der Testsuite angelegt wird,

oder ob das Element direkt im Attribut QF-Test ID der Komponente(776) des Mausevent(775) Knotens angegeben wird (vgl. Abschnitt 5.9(92)).

Wenn Sie mit dem Schalter ”Intelligent” QF-Test die Wahl überlassen, wird nur dann ein Knoten angelegt, wenn der Index als Text(938) angegeben ist und das Unterelement im SUT nicht editierbar ist.

Knoten im Baum als Pfad darstellen (System)

SUT Skript Name: OPT_RECORD_SUBITEM_TREE_PATH
Es ist nicht ungewöhnlich, dass in einer Baumstruktur Knoten mit gleichen Namen aber verschiedenen Vaterknoten vorkommen, z.B. in einem Linux Dateisystem die Verzeichnisse `/tmp` und `/usr/tmp`. Durch Ausnutzen der hierarchischen Struktur und Verwendung von Pfadnamen in den Elementen(936) kann QF-Test diese Knoten unterscheiden. Dabei kommt das Zeichen ’/’ als Trennzeichen zum Einsatz.

Ist diese Option nicht aktiviert, werden Bäume wie flache Listen betrachtet.

41.2.5 Aufnahmefenster

6.0+

Über die folgenden Einstellungen lässt sich Aussehen und Funktionsweise des Aufnahmefensters anpassen.

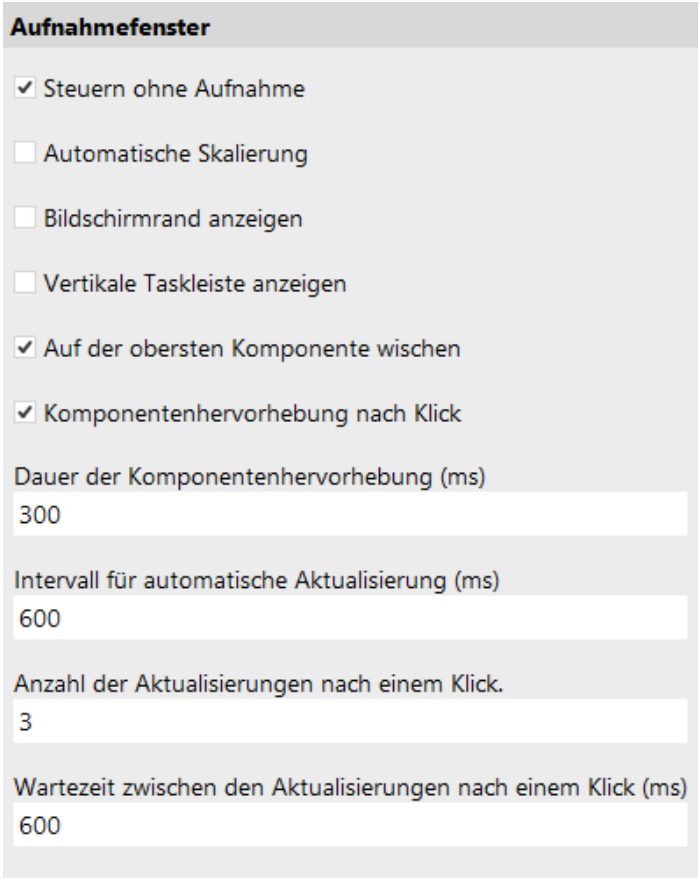


Abbildung 41.20: Optionen für das Aufnahmefenster

Steuern ohne Aufnahme (System)

Server Skript Name: OPT_RECORDING_CONTROL_STATE
Ist diese Option aktiviert werden Klicks und andere Eingaben im Aufnahmefenster an das aktive Gerät oder den aktiven Emulator/Simulator weitergegeben, auch wenn sich QF-Test nicht im Aufnahmemodus befindet.

Automatische Skalierung (System)

Server Skript Name: OPT_RECORDING_DISPLAY_AUTO_SCALING
Wenn diese Option gesetzt ist, wird das Vorschaubild im Aufnahmefenster automatisch anhand der Fenstergröße skaliert.

Bildschirmrand anzeigen (System)

Server Skript Name: OPT_SHOW_BORDER
Zeichnet einen Rahmen um den Rand des virtuellen Bildschirms im Aufnahmefenster.

Dadurch kann der Rand des Gerätes besser erkennbar werden, falls dessen Inhalt eine ähnliche Farbe hat wie das Aufnahmefenster.

Vertikale Taskleiste anzeigen (System)

Server Skript Name: OPT_VERTICAL_TOOLBAR_STATE
Blendet eine zusätzliche Leiste am Rand des Aufnahmefensters ein, die Aktionen zur Bedienung typischer Navigationselemente eines Gerätes enthält. Im Aufnahmemodus werden diese als Events aufgezeichnet.

Auf der obersten Komponente wischen (System)

Server Skript Name: OPT_SWIPE_ON_TOPLEVEL_COMPONENT
Ist diese Option aktiviert, so werden Wischgesten immer auf der obersten Komponente aufgenommen.

Durch unterschiedliche Bildschirmgrößen und -Auflösungen von Geräten kann es vorkommen, dass einzelne Komponenten nicht immer innerhalb des sichtbaren Bereiches liegen. Dadurch können Wischgesten, bei denen die darunterliegende Komponente unwichtig ist, unzuverlässig werden. Besonders die Aufnahme von Navigationsgesten kann durch diese Option verbessert werden.

Komponentenhervorhebung nach Klick (System)

Server Skript Name: OPT_CLICK_HIGHLIGHT
Ist diese Option aktiviert, so wird nach einem Klick im Aufnahmefenster die darunterliegende Komponente für kurze Zeit durch einen Rahmen hervorgehoben.

Dies kann hilfreich sein um schnell zu prüfen, ob ein Klick korrekt interpretiert wurde.

Dauer der Komponentenhervorhebung (ms) (System)

Server Skript Name: OPT_HIGHLIGHT_DURATION
Hiermit steuern Sie, wie lange der Rahmen um eine ausgewählte Komponente angezeigt werden soll.

Intervall für automatische Aktualisierung (ms) (System)

Server Skript Name: OPT_RECORDING_AUTO_REFRESH_INTERVALL

Hiermit steuern Sie, wie oft QF-Test versuchen soll, den Inhalt des Aufnahmefensters zu aktualisieren.

Bitte beachten Sie, dass die maximale Geschwindigkeit der Aktualisierung vom verwendeten Gerät oder Emulator/Simulator abhängt. Ein geringerer Intervall-Wert hat dann keinen Effekt.

Ein zu geringer Wert kann die Systemleistung negativ beeinträchtigen.

Anzahl der Aktualisierungen nach einem Klick (System)

Server Skript Name: OPT_REFRESH_STEPS
Hiermit legen Sie fest, wie oft das Aufnahmefenster nach einem Klick aktualisiert wird. Diese Option ist nur relevant, wenn die automatische Aktualisierung deaktiviert ist.

Je nach der in Android konfigurierbaren Animationsgeschwindigkeit kann es vorkommen, dass das Vorschaufenster während einer laufenden Animation aktualisiert wird. In solchen Fällen kann diese Option hilfreich sein.

Wartezeit zwischen den Aktualisierungen nach einem Klick (ms) (System)

Server Skript Name: OPT_INTERVAL_TIME_AFTER_CLICK
Hiermit legen Sie fest, wie viel Zeit zwischen den Aktualisierungen nach einem Klick vergehen soll. Diese Option ist nur relevant, wenn die automatische Aktualisierung deaktiviert ist.

Bitte beachten Sie, dass die maximale Geschwindigkeit der Aktualisierung vom verwendeten Gerät oder Emulator/Simulator abhängt. Ein geringerer Intervall-Wert hat dann keinen Effekt.

41.2.6 Prozeduren

Die folgenden Optionen haben Einfluss auf den Procedure Builder, der in Kapitel 27(368) detailliert beschrieben wird.

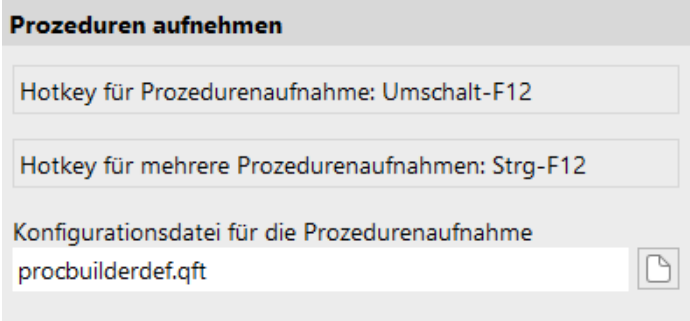


Abbildung 41.21: Procedure Builder Optionen

Hotkey für Prozedurenaufnahme (User)

SUT Skript Name: OPT_RECORD_PROCEDURE_HOTKEY
Diese Option legt eine Taste bzw. Tastenkombination fest, mit der die Aufnahme von Prozeduren direkt vom SUT aus eingeschaltet werden kann. Um den Wert zu ändern klicken Sie auf den aktuellen Eintrag (hierbei handelt um ein interaktives Feld) und drücken nun die gewünschte Taste bzw. Tastenkombination. Um das Feld zu verlassen, drücken Sie entweder die Tab-Taste oder setzen Sie den Fokus mit der Maus auf ein anderes Feld. Die Standard-Tastenkombination ist Shift-F12 für Window/Linux bzw. ⇧-F12 für Mac.

Hotkey für Prozedurenaufnahme (User)

SUT Skript Name: OPT_RECORD_PROCEDURE_CONTINUE_HOTKEY
Diese Option legt eine Taste bzw. Tastenkombination fest, mit der die Aufnahme von Prozeduren direkt vom SUT aus eingeschaltet werden kann. Um den Wert zu ändern klicken Sie auf den aktuellen Eintrag (hierbei handelt um ein interaktives Feld) und drücken nun die gewünschte Taste bzw. Tastenkombination. Um das Feld zu verlassen, drücken Sie entweder die Tab-Taste oder setzen Sie den Fokus mit der Maus auf ein anderes Feld. Die Standard-Tastenkombination ist Strg-F12 für Window/Linux bzw. ^-F12 für Mac.

Der Unterschied zu Hotkey für Prozedurenaufnahme[529] ist, dass Sie dauerhaft in diesen Modus bleiben um mehrere Prozeduren aufzunehmen zu können. Zum Beenden drücken Sie einfach noch einmal den ”Hotkey”.

Konfigurationsdatei für die Prozedurenaufnahme (System)

Hier können Sie eine eigene Konfigurationsdatei für den Procedure Builder festlegen. Ist ein relativer Pfad angegeben, sucht QF-Test im Verzeichnis, aus

dem QF-Test gestartet wurde, und im standard Include-Verzeichnis nach dieser Datei.

41.3 Wiedergabe

Die folgenden Einstellungen beeinflussen das Verhalten von QF-Test beim Abspielen eines Tests.

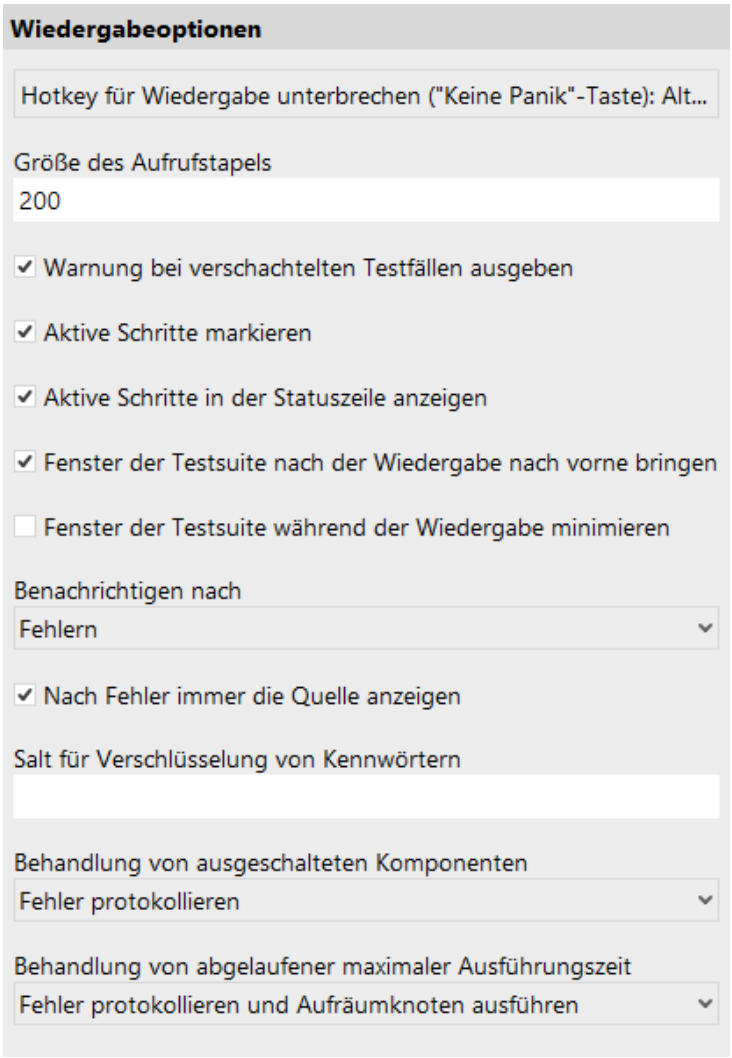


Abbildung 41.22: Wiedergabeoptionen

Hotkey für Wiedergabe unterbrechen (”Keine Panik”-Taste) (User)

Server (automatisch weiter an SUT) Skript Name: OPT_PLAY_DONT_PANIC_HOTKEY
Wenn ein Test mit voller Geschwindigkeit läuft, kann es ziemlich schwierig sein, den Fokus in eines von QF-Tests Fenstern zu bekommen um den Test anzuhalten und etwas anderes zu erledigen, ohne dass einem ständig irgendwelche Fenster um die Ohren fliegen. Noch schwieriger wird das ganze im Batchmodus oder bei gesetzten Mauszeiger tatsächlich bewegen[543] oder Fenster des SUT automatisch nach vorne bringen[542] Optionen.

Mit Hilfe dieser Option definieren Sie eine Tastenkombination (Standard ist Alt-F12), die sofort alle Tests pausiert, wenn sie in einem beliebigen Fenster gedrückt wird, egal ob im SUT oder in QF-Test (es sei denn Sie haben mehrere Instanzen von QF-Test gleichzeitig am laufen). Nochmaliges drücken der selben Tastenkombination setzt alle Tests fort, sofern Sie nicht manuell einen davon gestoppt oder fortgesetzt haben. In diesem Fall wird der Effekt automatisch auf "Pausieren" zurückgesetzt.

Um den Hotkey festzulegen, klicken Sie in das Feld und drücken Sie die gewünschte Taste oder Tastenkombination.

Größe des Aufnahmestapels (System)

Server Skript Name: OPT_PLAY_CALLSTACK_SIZE
Auf dem *Callstack* wird für jeden Prozeduraufruf[675] ein Eintrag mit den Parametern abgelegt, der nach dem Ende der Prozedur wieder entfernt wird. Die Größe des Callstacks gibt somit an, wie tief Prozeduraufrufe geschachtelt sein können. Sie ist beschränkt, um eine endlose Rekursion zu erkennen und zu verhindern. Eine Überschreitung des Wertes führt zu einer `StackOverflowException`[966]. Der Defaultwert von 200 sollte ausreichen, kann aber für sehr komplexe Tests vergrößert werden.

Warnung bei verschachtelten Testfällen ausgeben (System)

Server Skript Name: OPT_PLAY_WARN_NESTED_TEST_CASE
Die Ausführung von Testfall[599] Knoten sollte nicht verschachtelt werden, da solche Testfälle nicht korrekt im Report dargestellt werden können. Falls diese Option aktiviert ist, wird eine Warnung protokolliert, wenn ein Testfall innerhalb eines anderen Testfalls ausgeführt wird.

Aktive Schritte markieren (User)

Hier legen Sie fest, ob während der Wiedergabe die gerade aktiven Schritte im Baum mit einem kleinen Pfeil markiert werden sollen.

Aktive Schritte in der Statuszeile anzeigen (User)

Der gerade ausgeführte Schritt kann auch in der Statuszeile angezeigt werden. Dieses Feature kann mit dieser Option aktiviert werden.

Fenster der Testsuite nach der Wiedergabe nach vorne bringen (User)

Diese Option ist vor allem im Zusammenhang mit der Option Fenster des SUT automatisch nach vorne bringen(542) interessant. Sie sorgt dafür, dass das Fenster der Testsuite nach der Wiedergabe einer Sequenz wieder nach vorne gebracht wird.

Siehe auch Option Beim Nach-vorne-Bringen Fenster in den Vordergrund zwingen(543).

Fenster der Testsuite während der Wiedergabe minimieren (User)

Ist diese Option gesetzt, minimiert QF-Test das Fenster einer Testsuite während ihre Tests ausgeführt werden. Das Fenster erscheint automatisch sobald der Test beendet oder unterbrochen wird. Diese Funktion ist insbesondere für Windows 2000/XP Systeme hilfreich. Dort ist es Programmen nicht erlaubt, ihre Fenster nach vorne zu bringen, so dass QF-Test die Fenster des SUT nicht hinreichend beeinflussen kann.

Benachrichtigen nach (User)

Nachdem die Wiedergabe beendet ist, wird in jedem Fall eine Meldung in der Statuszeile angezeigt, die über die Zahl der aufgetretenen Warnungen, Fehler und Exceptions Auskunft gibt. Zusätzlich kann nach Fehlern, Warnung oder generell am Ende jedes Testlaufs ein Nachrichtendialog angezeigt werden. Diese Option legt fest, welche Fehlerstufe die Anzeige dieser Nachricht auslöst.

Nach Fehler immer die Quelle anzeigen (User)

Ist diese Option gesetzt, wird nach einer Exception bei der Ausführung der Schritt, der zum Fehler geführt hat, angezeigt und selektiert. Dies ist einerseits praktisch, kann andererseits aber störend sein, wenn man gerade etwas editiert. Der zuletzt aufgetretene Fehler kann zudem mittels Wiedergabe→Letzte Fehlerursache anzeigen... lokalisiert werden.

Salt für Verschlüsselung von Kennwörtern (System)

3.0+

QF-Test kann Variablenwerte verschlüsseln und mit Hilfe der speziellen

Variablen-Gruppe `decrypt` im Testablauf verfügbar machen (siehe Abschnitt 6.8[127]). Außerdem können Kennwörter im Attribut Text[786] eines Texteingabe[784] Knotens für ein Passwort-Feld und im Attribut Detail[795] eines Auswahl[793] Knotens für einen Login-Dialog eines Web SUT verschlüsselt abgelegt werden. Bei der Ver- und Entschlüsselung solcher Kennwörter kombiniert QF-Test seinen eigenen Schlüssel mit dem hier angegebenen Wert, dem sogenannten *Salt*. Ohne diesen wäre jeder ohne großen Aufwand in der Lage, Ihre verschlüsselten Kennwörter zu entschlüsseln und im Klartext zu sehen.

Hinweis

Lassen Sie sich von dieser Option nicht zu sehr in Sicherheit wiegen. Jeder der Zugriff auf diesen Salt hat und jeder, der Ihre Tests ausführen kann, ist ebenfalls in der Lage, sich die Klartext-Version Ihrer Kennwörter zu verschaffen. Dennoch ist die Verschlüsselung sinnvoll, um zu verhindern, dass in Testsuiten und Protokollen gespeicherte Kennwörter ganz offensichtlich unverschlüsselt herumliegen. Verschlüsselte Kennwörter bieten einen vernünftigen Schutz vor Personen, die keinen Zugriff auf diesen Salt haben.

Behandlung von ausgeschalteten Komponenten (System)

4.0+

Server Skript Name: OPT_PLAY_ERROR_STATE_DISABLED_COMPONENT
Mögliche Werte: VAL_PLAY_DISABLED_COMPONENT_WARNING,
VAL_PLAY_DISABLED_COMPONENT_ERROR,
VAL_PLAY_DISABLED_COMPONENT_EXCEPTION
Falls Sie eine Aktion wiedergeben und die dazugehörige Zielkomponente in der Testsuite ausgeschaltet ist, können Sie hier das gewünschte Verhalten festlegen. Sie können

- Eine Warnung in das Protokoll schreiben
- Eine Fehlermeldung in das Protokoll schreiben
- Eine `DisabledComponentStepException`[959] werfen

Behandlung von abgelaufener maximaler Ausführungszeit (System)

4.1+

Server Skript Name: OPT_PLAY_ERROR_STATE_EXECUTION_TIMEOUT
Mögliche Werte: VAL_PLAY_EXECUTION_TIMEOUT_WARNING,
VAL_PLAY_EXECUTION_TIMEOUT_ERROR,
VAL_PLAY_EXECUTION_TIMEOUT_EXCEPTION,
VAL_PLAY_EXECUTION_TIMEOUT_WARNING_IMMEDIATE,
VAL_PLAY_EXECUTION_TIMEOUT_ERROR_IMMEDIATE,
VAL_PLAY_EXECUTION_TIMEOUT_EXCEPTION_IMMEDIATE
Falls Sie eine Sequenz mit maximaler Ausführungszeit wiedergeben und diese Ausführungszeit abgelaufen ist, können Sie hier das gewünschte Verhalten festlegen. Sie können

- Eine Warnung in das Protokoll schreiben, welche den Testfall mit möglichen Aufräumaktionen fortsetzt.
- Eine Fehlermeldung in das Protokoll schreiben, welche den Testfall mit möglichen Aufräumaktionen fortsetzt.
- Eine `ExecutionTimeoutExpiredException`(960) werfen, welche den Testfall mit möglichen Aufräumaktionen fortsetzt.
- Eine Warnung in das Protokoll schreiben und den Knoten sofort beenden, ohne Aufräumaktionen.
- Eine Fehlermeldung in das Protokoll schreiben und den Knoten sofort beenden, ohne Aufräumaktionen.
- Eine `ExecutionTimeoutExpiredException`(960) werfen und den Knoten sofort beenden, ohne Aufräumaktionen.

Die obige Definition von möglichen Aufräumaktionen bedeutet, dass Aufräumen- wie auch Catchknoten ausgeführt werden. Keine Aufräumaktionen bedeutet, dass diese Aufräum- und Catchknoten nicht ausgeführt werden.

41.3.1 Client Optionen

Verschiedene Parameter für Prozesse und SUT-Clients können über die folgenden Optionen eingestellt werden:

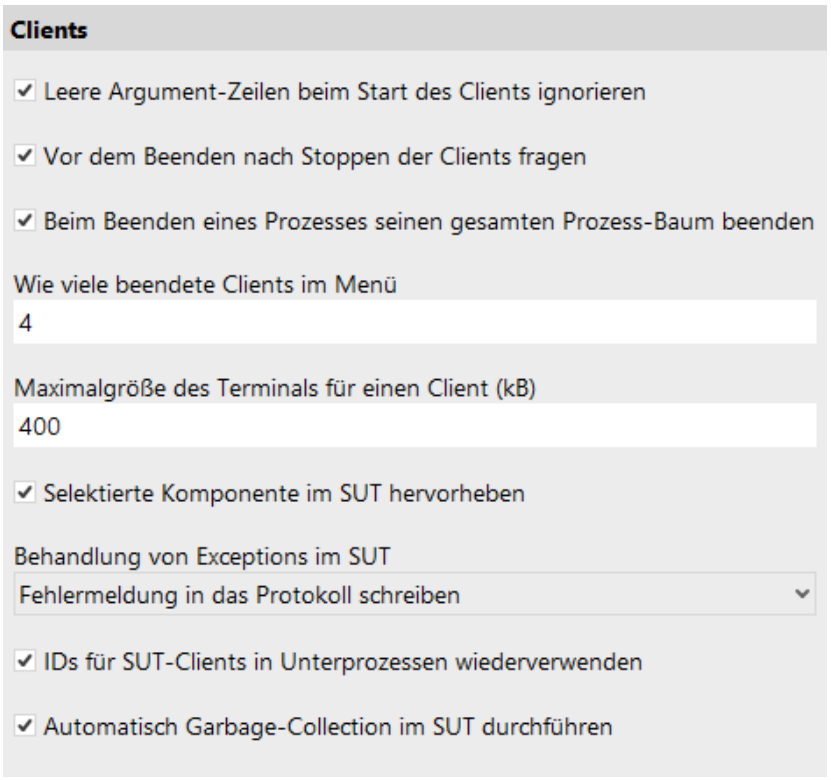


Abbildung 41.23: Client Optionen

Leere Argument-Zeilen beim Start des Clients ignorieren (System)

Server Skript Name: OPT_PLAY_CLIENT_START_IGNORE_EMPTY_ARGUMENT
Wenn die Option gesetzt ist (Standardeinstellung), werden leere Zeilen bei Programm-Parametern und Klassen-Argumenten in 'Starter'-Knoten wie zum Beispiel Java-SUT-Client starten[724], SUT-Client starten[728], Programm starten[731], Web-Engine starten[737], Windows-Anwendung starten[743] oder Android-Emulator starten[749] ignoriert. Dies ist insbesondere hilfreich, wenn Variablen für Parameter verwendet werden, da eine leere Variable dann analog dazu ist, den Parameter aus der Liste zu nehmen. Wird die Option ausgeschaltet, wird stattdessen ein leeres Kommandozeilenargument (also '') an das startende Programm übergeben.

Vor dem Beenden nach Stoppen der Clients fragen (User)

Sind beim Beenden von QF-Test noch Clients aktiv, werden diese nach einer

Rückfrage beendet. Ist diese Option ausgeschaltet, werden die Clients ohne Rückfrage beendet.

Beim Beenden eines Prozesses seinen gesamten Prozess-Baum beenden (System)

Server Skript Name: OPT_PLAY_KILL_PROCESS_TREE
Der Prozess eines SUT oder eines Hilfsprogramms, das während eines Tests gestartet wurde, kann mittels eines Programm beenden(769) Knotens oder manuell über das Client Menü beendet werden. Im Fall eines SUT versucht QF-Test zunächst, mit diesem zu kommunizieren und einen sauberen Aufruf von `System.exit` auszulösen. Nicht-Java-Programme müssen hart terminiert werden. Falls das Programm weitere Kind-Prozesse gestartet hat, können diese dabei je nach Umständen automatisch beendet werden oder nicht.

Es ist normalerweise nicht wünschenswert, solche Kind-Prozesse zu erhalten, da diese Konflikte mit weiteren Tests verursachen oder Dateien blockieren können, die gelöscht oder überschrieben werden sollen. Sofern diese Option nicht deaktiviert wird, versucht QF-Test den gesamten Prozess-Baum für ein aus QF-Test gestartetes Programm zu ermitteln und sicherzustellen, dass beim Beenden dieser Prozess und alles seine Kind-Prozesse explizit terminiert werden.

Wie viele beendete Clients im Menü (User)

Server Skript Name: OPT_PLAY_MAX_CLIENTS
Hiermit legen Sie die Anzahl der bereits beendeten Clients fest, deren Terminal noch über das Clients Menü zugänglich ist.

Maximalgröße des Terminals für einen Client (kB) (User)

3.0+

Server (automatisch weiter an SUT) Skript Name: OPT_PLAY_TERMINAL_SIZE
Die maximale Menge an Text (in Kilobyte), die ein individuelles Client-Terminal aufnimmt. Ist diese Schwelle überschritten, wird alter Text entfernt wenn neuer Text hinzukommt. Der Wert 0 steht für unbegrenzten Text.

Hinweis

Diese Option legt auch die Menge an Ausgaben fest, die über die speziellen Variablen `${qftest:client.output.<name>}`, `${qftest:client.stdout.<name>}` und `${qftest:client.stderr.<name>}` verfügbar sind.

Selektierte Komponente im SUT hervorheben (User)

Server Skript Name: OPT_PLAY_HIGHLIGHT_COMPONENTS
Ist diese Option gesetzt, hebt QF-Test die zugehörige Komponente im SUT

optisch hervor, wenn ein Komponente(930) Knoten oder ein Knoten, der eine Komponente referenziert, selektiert wird.

Behandlung von Exceptions im SUT (System)

SUT Skript Name: OPT_PLAY_SUT_EXCEPTION_LEVEL
Mögliche Werte: VAL_PLAY_EXCEPTION_LEVEL_WARNING,
VAL_PLAY_EXCEPTION_LEVEL_ERROR,
VAL_PLAY_EXCEPTION_LEVEL_EXCEPTION
Exceptions, die während der Abarbeitung eines Events im SUT auftreten, deuten sehr wahrscheinlich auf einen Fehler im SUT hin. Mit dieser Option legen Sie fest, welche Konsequenzen dieser Fall hat. Sie können

- Eine Warnung in das Protokoll schreiben
- Eine Fehlermeldung in das Protokoll schreiben
- Eine `UnexpectedClientException`(965) werfen

IDs für SUT-Clients in Unterprozessen wiederverwenden (System)

Server Skript Name: OPT_PLAY_REUSE_SUT_IDS
Dies ist eine komplexe Option die Sie hoffentlich nie benötigen werden. Wenn ein SUT Client einen Unterprozess startet, der sich seinerseits mit QF-Test verbindet, erhält der neue SUT-Client einen speziellen Namen. Dieser wird aus dem Client-Namen des ursprünglichen SUT gebildet, gefolgt von ':' und einer numerischen ID. Die erste solche ID ist immer 2, mit aufsteigenden Zahlen für weitere Unterprozesse.

Wird ein Unterprozess beendet und ein neuer gestartet, kann QF-Test entweder die ID des beendeten Prozesses wiederverwenden, oder die IDs weiter hochzählen und eine neue ID vergeben.

In den meisten Fällen ist es besser die IDs für Unterprozesse wiederzuverwenden. Ein typischer Fall ist, dass ein Unterprozess gestartet, beendet und wieder gestartet wird. Wenn Sie diese Option aktivieren können Sie in einem solchen Fall den Unterprozess immer mit dem gleichen Namen adressieren.

In einer komplexeren Situation können eventuell diverse Unterprozesse relativ willkürlich gestartet und beendet werden, je nach Ablauf eines Tests. In diesem Fall ist die Vergabe einer immer neuen ID deterministischer.

In jedem Fall wird der Zähler wieder zurückgesetzt, wenn der ursprüngliche SUT Client neu gestartet wird.

Automatisch Garbage-Collection im SUT durchführen (System)

SUT Skript Name: OPT_PLAY_SUT_GARBAGE_COLLECTION
Normalerweise führt QF-Test im SUT automatisch eine volle Garbage-Collection durch, nachdem einige hundert SUT-Skript(720) Knoten ausgeführt wurden. Dies ist notwendig, da der standard Mechanismus zur Garbage-Collection in Java zu einem OutOfMemoryError im sogenannten PermGen Space führen kann, obwohl dort problemlos Speicher durch eine Garbage-Collection wiedergewonnen werden könnte.

Wenn Sie versuchen, das Speicherverhalten Ihrer Anwendung genauer zu analysieren, kann diese explizite Garbage-Collection die Ergebnisse beeinflussen. Für diesen Fall können Sie mit Hilfe dieser Option die Garbage-Collection durch QF-Test unterbinden.

41.3.2 Terminal Optionen

Verschiedene Parameter für das gemeinsame Terminal können über die folgenden Optionen eingestellt werden.

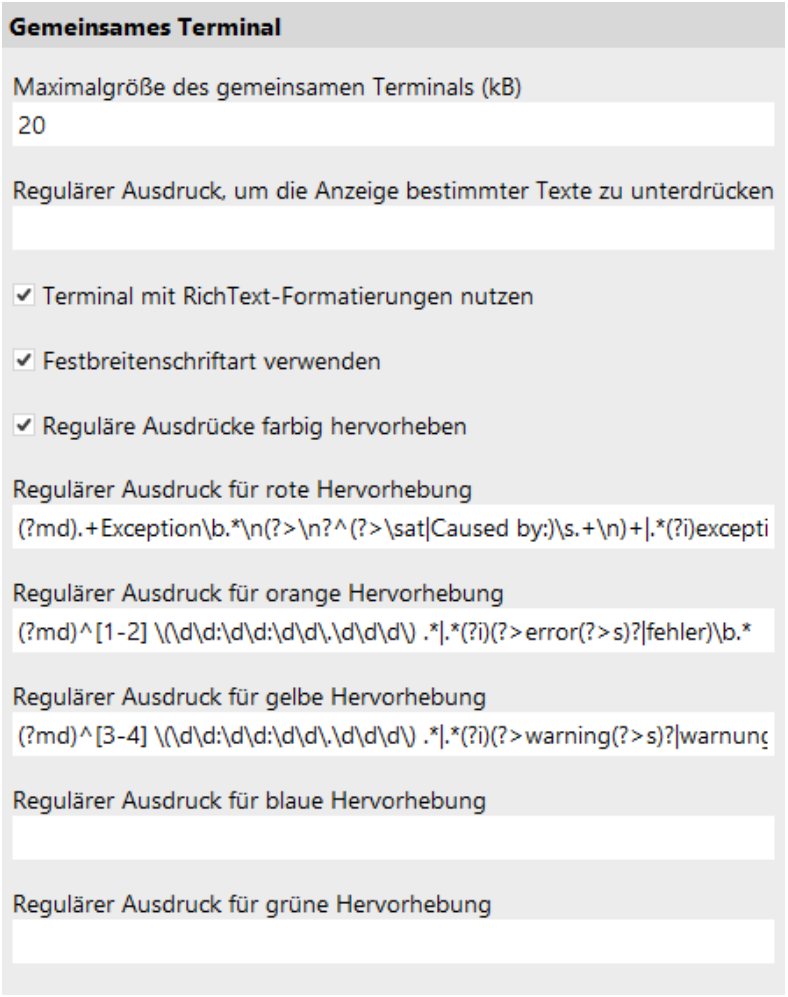


Abbildung 41.24: Terminal options

Maximalgröße des gemeinsamen Terminals (kB) (User)

Server Skript Name: OPT_PLAY_SHARED_TERMINAL_SIZE
Die maximale Menge an Text (in Kilobyte), die das gemeinsame Terminal aufnimmt. Ist diese Schwelle überschritten, wird alter Text entfernt wenn neuer Text hinzukommt. Der Wert 0 steht für unbegrenzten Text.

Regulärer Ausdruck, um die Anzeige bestimmter Texte zu unterdrücken (User)

4.0+

Durch Angabe eines regulären Ausdrucks in dieser Option können bestimmte Texte in der Terminalausgabe unterdrückt werden.

Standardwert ist leer.
Siehe auch Reguläre Ausdrücke - *Regexps*(1023).

Terminal mit RichText-Formatierungen nutzen (User)

4.0+

Aktiviert das RichText Terminal, das eine monospaced Font und die farbige Hervorhebung von selbstdefinierten regulären Ausdrücken erlaubt. Deaktivieren Sie diese Option, wenn sie zum einfachen Terminal zurückwechseln möchten, wie es vor QF-Test Version 4 war.

Hinweis

QF-Test muss neu gestartet werden, um eine Änderung dieser Option sichtbar zu machen.

Monospaced Font verwenden (User)

4.0+

Wenn aktiviert, wird für das gemeinsame Terminal ein monospaced Font verwendet.

Diese Option hat nur einen Effekt, wenn Terminal mit RichText-Formatierungen nutzen(540) aktiv ist.

Reguläre Ausdrücke farbig hervorheben (User)

4.0+

Mit dieser Option kann die farbige Hervorhebung von selbstdefinierten regulären Ausdrücken ein- und ausgeschaltet werden.

Diese Option hat nur einen Effekt, wenn Terminal mit RichText-Formatierungen nutzen(540) und Reguläre Ausdrücke farbig hervorheben(540) aktiv sind.

Regulärer Ausdruck für rote Hervorhebung (User)

4.0+

In dieser Option kann ein regulärer Ausdruck für Ausgaben definiert werden, die rot hervorgehoben werden sollen.

Diese Option hat nur einen Effekt, wenn Terminal mit RichText-Formatierungen nutzen(540) und Reguläre Ausdrücke farbig hervorheben(540) aktiv sind.

Standardwert ist:
(?md).+Exception\b.*\n(?%gt;\n?^(?%gt;\sat|Caused by:)\s.+\n)+|.*(?i)exception(?%gt;s)?\b.*
Dadurch werden auch typische Java-Stack-Traces bei Exceptions rot hervorgehoben.
Siehe auch Reguläre Ausdrücke - *Regexps*(1023).

Regulärer Ausdruck für orange Hervorhebung (User)

4.0+

In dieser Option kann ein regulärer Ausdruck für Ausgaben definiert werden, die orange hervorgehoben werden sollen.

Diese Option hat nur einen Effekt, wenn Terminal mit RichText-Formatierungen nutzen(540) und Reguläre Ausdrücke farbig hervorheben(540) aktiv sind.

Standardwert ist:
(?md)^[1-2] \(\d\d:\d\d:\d\d\.\d\d\d\) .*|.*(?i)(?%gt;error(?%gt;s)?|fehler)\b.*
Dadurch werden auch Fehler-Protokollmeldungen hervorgehoben.
Siehe auch Reguläre Ausdrücke - *Regexps*(1023).

Regulärer Ausdruck für gelbe Hervorhebung (User)

4.0+

In dieser Option kann ein regulärer Ausdruck für Ausgaben definiert werden, die gelb hervorgehoben werden sollen.

Diese Option hat nur einen Effekt, wenn Terminal mit RichText-Formatierungen nutzen(540) und Reguläre Ausdrücke farbig hervorheben(540) aktiv sind.

Standardwert ist:
(?md)^[3-4] \(\d\d:\d\d:\d\d\.\d\d\d\) .*|.*(?i)(?%gt;warning(?%gt;s)?|warnung(?%gt;en)?)\b.*
Dadurch werden auch Warnungs-Protokollmeldungen hervorgehoben.
Siehe auch Reguläre Ausdrücke - *Regexps*(1023).

Regulärer Ausdruck für blaue Hervorhebung (User)

4.0+

In dieser Option kann ein regulärer Ausdruck für Ausgaben definiert werden, die blau hervorgehoben werden sollen.

Diese Option hat nur einen Effekt, wenn Terminal mit RichText-Formatierungen nutzen(540) und Reguläre Ausdrücke farbig hervorheben(540) aktiv sind.

Standardwert ist leer.
Siehe auch Reguläre Ausdrücke - *Regexps*(1023).

Regulärer Ausdruck für grüne Hervorhebung (User)

4.0+

In dieser Option kann ein regulärer Ausdruck für Ausgaben definiert werden, die grün hervorgehoben werden sollen.

Diese Option hat nur einen Effekt, wenn Terminal mit RichText-Formatierungen nutzen(540) und Reguläre Ausdrücke farbig hervorheben(540) aktiv sind.

Standardwert ist leer.
Siehe auch Reguläre Ausdrücke - *Regexps*(1023).

41.3.3 Events

Diese Optionen beeinflussen einige Details bei der Wiedergabe von Events.

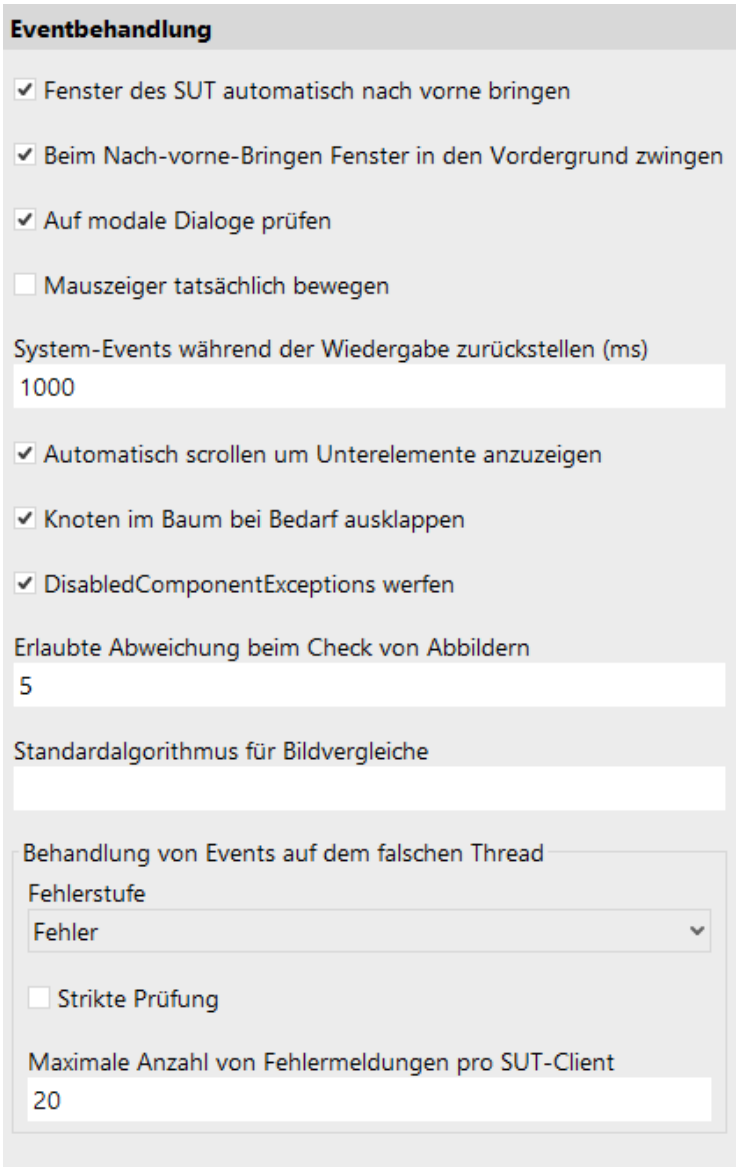


Abbildung 41.25: Optionen zur Eventbehandlung

Fenster des SUT automatisch nach vorne bringen (System)

SUT Skript Name: OPT_PLAY_RAISE_SUT_WINDOWS
Ist diese Option gesetzt, werden Fenster des SUT, für die ein Mouse- oder

KeyEvent simuliert wird, bei der Aktivierung nach vorne gebracht. Das vereinfacht den Wechsel zwischen QF-Test und dem SUT, um den Ablauf einer Sequenz zu beobachten.

Siehe auch Optionen Fenster der Testsuite nach der Wiedergabe nach vorne bringen[532] und Beim Nach-vorne-Bringen Fenster in den Vordergrund zwingen[543].

Beim Nach-vorne-Bringen Fenster in den Vordergrund zwingen (System)

3.4.1+

SUT Skript Name: OPT_PLAY_RAISE_SUT_WINDOWS_FORCED

Hinweis

Diese Option wird nur für Windows Systeme unterstützt.

Windows erlaubt einer Anwendung nur dann, ein eigenes Fenster nach vorne zu bringen, wenn diese Anwendung bereits den Fokus besitzt. Dies kann es für QF-Test schwierig machen, Fenster des SUT in den Vordergrund zu bringen oder automatisch zwischen dem SUT und QF-Test zu wechseln. Ist diese Option aktiv, setzt QF-Test vorübergehend die Eigenschaft ”Immer im Vordergrund”, um ein Fenster nach vorne zu zwingen.

Siehe auch Optionen Fenster der Testsuite nach der Wiedergabe nach vorne bringen[532] und Fenster des SUT automatisch nach vorne bringen[542].

Auf modale Dialoge prüfen (System)

SUT Skript Name: OPT_PLAY_CHECK_MODAL

Modale Dialoge sind solche Dialoge, die alle anderen Fenster blockieren. Sie werden unter anderem für Fehlermeldungen verwendet. Da QF-Test die Events quasi durch die Hintertür an den Client schickt, werden diese durch einen modalen Dialog nicht blockiert und können weitere Aktionen im Hauptfenster auslösen, was die Situation nach einem Fehler nur noch verschlimmern kann. Daher wird - sofern diese Option gesetzt ist - vor der Ausführung jedes Events überprüft, ob das Zielfenster durch einen modalen Dialog blockiert ist und in diesem Fall eine `ModalDialogException`[959] ausgelöst.

Auch diese Option sollte immer eingeschaltet bleiben und wird evtl. ganz entfernt werden.

Mauszeiger tatsächlich bewegen (System)

SUT Skript Name: OPT_PLAY_MOVE_MOUSE_CURSOR

Ist diese Option gesetzt, wird der Mauszeiger bei der Simulation von Mausevents tatsächlich über den Bildschirm bewegt. Diese Funktion benötigt einen funktionsfähigen AWT Robot.

Obwohl diese Option hauptsächlich dem visuellen Feedback dient, kann sie einen positiven Einfluss auf die Zuverlässigkeit von Tests haben, da sie die Nebeneffekte von Systemevents reduziert die den Test beeinträchtigen könnten. Allerdings sollte diese Option für solche Tests ausgeschaltet sein, bei denen es auf präzise Mausbewegungen ankommt, z.B. für ein Zeichenwerkzeug.

System-Events während der Wiedergabe zurückstellen (ms) (System)

SUT Skript Name: OPT_PLAY_DELAY_HARD_EVENTS
Greift nur bei Swing- und SWT-Anwendungen.

Während der Wiedergabe filtert oder verzögert QF-Test verschiedene Events, die nicht von QF-Test ausgelöst wurden, sondern vom System kommen, z.B. weil der Anwender die Maus bewegt. Insbesondere Popupfenster für Menüs oder Comboboxen sind sehr empfindlich gegenüber solchen Störeinflüssen. Diese Funktionalität erhöht daher die Stabilität von Tests.

Mit dieser Option lässt sich die maximale Verzögerung für solche Events einstellen. Für den unwahrscheinlichen Fall dass diese Filterung unerwünschte Nebeneffekte hat, kann sie mit dem Wert 0 ausgeschaltet werden.

Automatisch scrollen um Unterelemente anzuzeigen (System)

SUT Skript Name: OPT_PLAY_SCROLL_ITEM
Ist diese Option gesetzt, werden die Unterelemente von komplexen Komponenten, die sich in einer Scrollpane befinden, automatisch in den sichtbaren Bereich gescrollt, wenn QF-Test darauf zugreift. In diesem Fall können Sie die meisten aufgenommene Events auf Scrollbars oder Scrollbuttons entfernen, da sie für eine korrekte Wiedergabe nicht mehr benötigt werden.

Knoten im Baum bei Bedarf ausklappen (System)

SUT Skript Name: OPT_PLAY_EXPAND_TREE
Wenn die Knoten eines Baums als hierarchische Unterelemente angesprochen werden, können Knoten als Ziel angegeben werden, die im Moment nicht sichtbar sind, weil ein übergeordneter Knoten nicht expandiert ist. Ist diese Option gesetzt, werden in diesem Fall alle übergeordneten Knoten automatisch expandiert. Andernfalls führt diese Situation zu einer `ComponentNotFoundException`[958].

DisabledComponentExceptions werfen (System)

SUT Skript Name: OPT_PLAY_THROW_DISABLED_EXCEPTION
Wenn QF-Test Events für eine Komponente abspielt, die im Moment deaktiviert ist, werden diese Events einfach ignoriert. Diese Situation

deutet praktisch immer auf einen Fehler hin, der durch werfen einer `DisabledComponentException`(959) signalisiert wird.

Ältere Testsuiten sind eventuell nicht auf diese Exception vorbereitet. Diese Testsuiten sollten angepasst werden. Als schneller Workaround können `DisabledComponentExceptions` aber auch durch deaktivieren dieser Option unterdrückt werden.

Erlaubte Abweichung beim Check von Abbildern (System)

SWT

SUT Skript Name: OPT_PLAY_IMAGE_TOLERANCE

Hinweis

Dies Option war zunächst nur für SWT/Gtk gedacht, hat sich aber als universell anwendbar und nützlich erwiesen.

Die Darstellung von Grafik ist in Java-Anwendungen und Web-Browsern nicht immer ganz deterministisch. Selbst innerhalb desselben Laufs einer Anwendung auf einem Display mit begrenzter Farbtiefe können die RGB Werte einer Icon Abbildung geringfügig variieren und bei der Ausführung von Tests auf verschiedenen Rechnern sind stärkere Abweichungen möglich. Grafiktreiber, JDK Version und Einstellungen des Betriebssystems spielen auch eine Rolle. Dies mach strikte Checks von Abbildungen unter Umständen fast unbrauchbar.

Um das Problem zu umgehen legt diese Option eine Toleranzschwelle für klassische Checks von Abbildungen fest, bis zu der Abweichungen in den einzelnen Farbanteilen rot, grün und blau eines Pixels erlaubt sind. Mit einem Wert von 0 lassen sich somit exakte Checks erzwingen, allerdings ist hierfür der ”identity” Algorithmus besser geeignet (vgl. Details des Algorithmus zum Bildvergleich(1309)). Der Standardwert von 5 ist ein guter Kompromiss, bei dem Checks mit Abweichungen, die normalerweise nicht visuell wahrnehmbar sind, erfolgreich sein können.

Standardalgorithmus für Bildvergleiche (System)

9.0+

Server Skript Name: OPT_PLAY_IMAGE_CHECK_DEFAULT_ALGORITHM

Mithilfe des Algorithmus zum Bildvergleich(831) Attributes eines Check Abbild(828) kann man festlegen, welcher Algorithmus für den Bildvergleich verwendet werden soll. Auch `rc.checkImageAdvanced` kennt ein Algorithmus-Argument. Sollte dieses Attribut bzw. im Falle von `rc.checkImageAdvanced` dieses Argument leer gelassen werden, so wird der über diese Option verwendete Algorithmus verwendet.

Behandlung von Events auf dem falschen Thread (System)

Swing

SUT Skript Name: OPT_PLAY_WRONG_THREAD_ERROR_LEVEL

Mögliche Werte: VAL_PLAY_THREAD_LEVEL_WARNING, VAL_PLAY_THREAD_LEVEL_ERROR,

VAL_PLAY_THREAD_LEVEL_EXCEPTION
Ein häufiger Fehler in Swing basierten Java-Anwendungen ist der Zugriff auf GUI Komponenten von einem falschen Thread. Da Swing nicht thread-safe ist, dürfen solche Aufrufe nur vom AWT Event Dispatch Thread kommen. Andernfalls können Race-Conditions oder Deadlocks die Folge sein. Erstere können zu subtilen und schwer auffindbaren Fehlern führen, bei letzteren friert die Anwendung komplett ein und ist nicht mehr verwendbar. Hintergrund-Informationen zu diesem Thema finden Sie unter
http://download.oracle.com/javase/tutorial/uiswing/concurrency/index.html,
speziell die Abschnitte zu ”Initial Threads” und ”The Event Dispatch Thread”.

Wenn QF-Test einen Event auf einem anderen als dem AWT Event Dispatch Thread erhält, gibt es eine Fehlermeldung zusammen mit einem aktuellen Stacktrace aus, der bei der Beseitigung des Fehlers hilfreich sein kann. Diese Gruppe von Optionen legt die Fehlerstufe der Meldung fest, ob strikte Tests durchgeführt werden und wie viele Meldungen maximal ausgegeben werden.

Die möglichen Werte für die Option ”Fehlerstufe” sind ”Fehler” und ”Warnung”. Wir raten dringend dazu, die Standardeinstellung ”Fehler” beizubehalten und derartige Probleme in der Anwendung umgehend zu beseitigen, da sie ein hohes Risiko darstellen.

Strikte Prüfung (System)

Swing

SUT Skript Name: OPT_PLAY_WRONG_THREAD_STRICT
Falls die Option ”Strikte Prüfung” aktiviert ist, werden für sämtliche Events von einem falschen Thread Meldungen ausgegeben. Andernfalls werden ”weniger wichtige” Events ignoriert. Die Unterscheidung ist willkürlich und beruht auf der Tatsache, dass es diverse Java-Literatur gibt (inklusive früherer Java-Dokumentation von Sun), in der es als korrekt dargestellt wird, Swing Komponenten auf einem beliebigen Thread zu initialisieren, solange diese nicht angezeigt werden. Viele Java-Programme sind so implementiert und das Risiko ist in diesem Fall in der Tat gering. Für derartigen Code verhindert das Abschalten von ”Strikte Prüfung”, dass deswegen massenhaft Fehler gemeldet werden und ggf. ernstere Probleme dadurch nicht mehr wahrgenommen werden. Wenn Sie dagegen alle Thread-Verstöße beseitigen wollen, was wir grundsätzlich empfehlen, sollten Sie ”Strikte Prüfung” aktivieren.

Maximale Anzahl von Fehlermeldungen pro SUT-Client (System)

Swing

SUT Skript Name: OPT_PLAY_WRONG_THREAD_MAX_ERRORS
Wenn Ihre Anwendung Code enthält, der die Thread-Vorgaben verletzt, kann es sein dass daraus eine sehr große Menge von Fehlermeldungen resultiert, was zu einem starken Einbruch der Performance führen kann. Andererseits bringen

diese Fehlermeldungen nach der ersten Handvoll keine neue Information mehr. Mittels der Option ”Maximale Anzahl von Fehlermeldungen pro SUT-Client” kann die Zahl der Fehlermeldungen begrenzt werden.

41.3.4 Wiedererkennung

Mit diesen Optionen können Sie die Wiedererkennung von Komponenten zur Laufzeit eines Tests beeinflussen. Die vorgegebenen Werte sollten im Normalfall gute Ergebnisse liefern, aber wenn Komponenten nicht erkannt werden, können andere Einstellungen vielleicht helfen.

Die Bedeutung der einzelnen Werte ist Abschnitt 48.1[1015] erläutert.

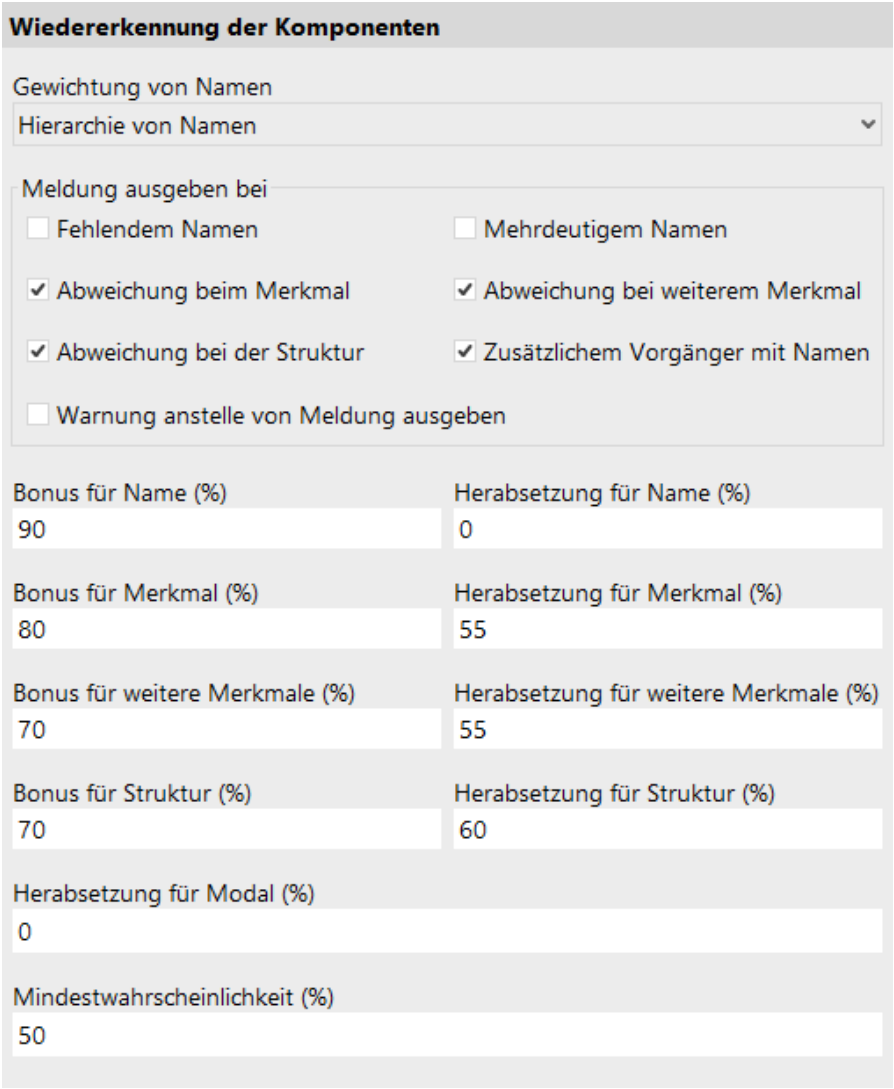


Abbildung 41.26: Optionen zur Wiedererkennung

Der Name einer Komponente spielt eine besondere Rolle. Die folgende Optionen beeinflusst das Gewicht, das QF-Test den Namen beimisst:

Gewichtung von Namen (Wiedergabe) (System)

SUT Skript Name: OPT_PLAY_RECOGNITION_NAME_OVERRIDE
Mögliche Werte: VAL_NAME_OVERRIDE_EVERYTHING,
VAL_NAME_OVERRIDE_HIERARCHY,
VAL_NAME_OVERRIDE_PLAIN

Hinweis Es gibt zwei Varianten dieser Option, die sehr eng miteinander verknüpft sind.

Diese Variante ist während der Wiedergabe aktiv, die andere[520] bei der Aufnahme. Natürlich sollten beide Optionen immer den selben Wert haben - mit einer Ausnahme: Wenn Sie von einer Einstellung zu einer anderen wechseln wollen, müssen eventuell Komponenten in QF-Test aktualisiert werden. Bei diesem Prozess ist es notwendig, zunächst die Einstellung für die Wiedergabe auf dem alten Wert zu lassen und nur die Aufnahme Option umzustellen. Denken Sie aber unbedingt daran, nach Abschluss der Aktualisierung auch die Wiedergabe Option umzustellen.

Diese Option legt fest, welches Gewicht dem Namen bei der Wiedererkennung von Komponenten beigemessen wird. Folgende Einstellungen sind möglich:

Name übertrifft alles
Dies ist die wirksamste und flexibelste Möglichkeit, Komponenten zu erkennen. Sie setzt allerdings voraus, dass die Namen der Komponenten zumindest pro Fenster eindeutig sind. Wenn diese Eindeutigkeit gegeben ist, verwenden Sie diese Einstellung.

Web Verwenden Sie diesen Wert nicht bei Webseiten mit Frames. Für diese ist ”Hierarchie von Namen” besser geeignet.

Hierarchie von Namen
Diese Einstellung sollten Sie verwenden, wenn Namen zwar nicht in jedem Fenster eindeutig vergeben sind, aber Komponenten mit gleichen Namen zumindest in unterschiedlichen Komponenten mit verschiedenen Namen enthalten sind, so dass sich eine Eindeutige Namenshierarchie ergibt. Damit ist die Wiedererkennung immer noch sehr tolerant gegenüber Veränderungen. Erst wenn Sie eine benannte Komponente in eine andere benannte Komponente verschieben, muss die Testsuite an diese Veränderung angepasst werden.

Normales Attribut
Falls es Komponenten mit identischen Namen im SUT gibt, die zudem in der gleichen Parent Komponente liegen, bleibt nur noch diese Einstellung. Der Name spielt damit immer noch eine wichtige Rolle, aber kaum mehr als das Merkmal[932] Attribut.

Der Algorithmus zur Wiedererkennung von Komponenten ist sehr tolerant und darauf ausgerichtet, nach Möglichkeit einen Treffer zu finden. Falls die beste Übereinstimmung nicht ganz vollständig ist, gibt QF-Test Information über die verbliebenen Abweichungen aus, wahlweise als Warnung oder einfache Meldung, abhängig von folgenden Optionen:

Meldung ausgeben bei fehlendem Namen (System)

SUT Skript Name: OPT_PLAY_WARN_MISSING_NAME
Ist diese Option gesetzt, wird eine Meldung im Protokoll ausgegeben, wenn beim Ablauf eines Tests eine Komponente angesprochen wird, die noch keinen Namen

hat, aber einen brauchen könnte. Soweit möglich wird dabei ein sinnvoller Name vorgeschlagen.

Meldung ausgeben bei mehrdeutigem Namen (System)

SUT Skript Name: OPT_PLAY_WARN_AMBIGUOUS_NAME
Ist die Option Gewichtung von Namen (Wiedergabe)[548] auf ”Name übertrifft alles” oder ”Hierarchie von Namen” gesetzt, wird eine Meldung im Protokoll ausgegeben wenn QF-Test auf mehr als eine mögliche Zielkomponente mit dem selben Namen trifft. Diese Meldung kann mit Hilfe dieser Option unterdrückt werden.

Meldung ausgeben bei Abweichung beim Merkmal (System)

SUT Skript Name: OPT_PLAY_WARN_FEATURE_MISMATCH
Wird eine Komponente im SUT als die bestgeeignete für einen Event oder Check ausgewählt obwohl in einer oder mehrerer Ebenen der Hierarchie das aufgenommene Merkmal[932] nicht mit der Komponente übereinstimmt, sprechen wir von einer ”Abweichung beim Merkmal”. Ist diese Option aktiviert, wird in einem solchen Fall eine Meldung ausgegeben, die Sie darauf hinweist, dass es eine gute Idee sein könnte, die betroffenen Komponenten zu aktualisieren.

Meldung ausgeben bei Abweichung bei weiterem Merkmal (System)

SUT Skript Name: OPT_PLAY_WARN_EXTRA_FEATURE_MISMATCH
Eine ”Abweichung bei weiterem Merkmal” ist analog zur oben beschriebenen Abweichung beim Merkmal, nur dass sie sich auf die weiteren Merkmale mit Status ”Sollte übereinstimmen” bezieht. Ist diese Option aktiviert, wird im Fall einer Abweichung eine Meldung ausgegeben, die Sie darauf hinweist, dass es eine gute Idee sein könnte, die betroffenen Komponenten zu aktualisieren.

Meldung ausgeben bei Abweichung bei der Struktur (System)

SUT Skript Name: OPT_PLAY_WARN_STRUCTURE_MISMATCH
Eine ”Abweichung bei der Struktur” ist analog zur oben beschriebenen Abweichung beim Merkmal, nur dass es hier statt dem Merkmal die Struktur-Attribute Index[935] und Insgesamt[935] sind, die nicht übereinstimmen. Ist diese Option aktiviert, wird in einem solchen Fall eine Meldung ausgegeben, die Sie darauf hinweist, dass es eine gute Idee sein könnte, die betroffenen Komponenten zu aktualisieren.

Meldung ausgeben bei zusätzlichem Vorgänger mit Namen (System)

SUT Skript Name: OPT_PLAY_WARN_NAMED_ANCESTOR
Eine direkte oder indirekte Parent-Komponente der ausgewählten Zielkomponente im SUT, die nicht in der Hierarchie in QF-Test auftaucht obwohl sie einen Namen hat, stellt eine Abweichung vergleichbar der bei Merkmal oder Struktur dar, sofern die Option Gewichtung von Namen (Wiedergabe)[548] auf den Wert ”Hierarchie von Namen” gesetzt ist. Ist diese Option aktiviert, wird in einem solchen Fall eine Meldung ausgegeben, die Sie darauf hinweist, dass es eine gute Idee sein könnte, die betroffenen Komponenten zu aktualisieren.

Warnung anstelle von Meldung ausgeben (System)

4.2+

SUT Skript Name: OPT_PLAY_COMPONENT_WARNINGS
Ist diese Option aktiviert, werden Abweichungen bei der Wiedererkennung als Warnungen statt einfacher Meldungen ausgegeben. Es kann hilfreich sein, diese Option vorübergehend einzuschalten, um die Sichtbarkeit solcher Abweichung zu erhöhen, z.B. um die Informationen für die Komponenten zu aktualisieren, oder ungewollt erkannten Komponenten auf die Spur zu kommen. Im normalen Testbetrieb erzeugt dies zu viel Rauschen und führt dazu, dass wichtigere Warnungen übersehen werden.

Erklärungen zu den übrigen Optionen für die Wiedererkennung finden Sie in Abschnitt 48.1[1015]. Die zugehörigen SUT-Skript Namen dieser Optionen lauten:
OPT_PLAY_RECOGNITION_BONUS_NAME
OPT_PLAY_RECOGNITION_PENALTY_NAME
OPT_PLAY_RECOGNITION_BONUS_FEATURE
OPT_PLAY_RECOGNITION_PENALTY_FEATURE
OPT_PLAY_RECOGNITION_BONUS_EXTRAFEATURE
OPT_PLAY_RECOGNITION_PENALTY_EXTRAFEATURE
OPT_PLAY_RECOGNITION_BONUS_STRUCTURE
OPT_PLAY_RECOGNITION_PENALTY_STRUCTURE
OPT_PLAY_RECOGNITION_PENALTY_MODAL
OPT_PLAY_RECOGNITION_MINIMUM_PROBABILITY

41.3.5 Verzögerungen

Hier können Sie Standardwerte für allgemeine Verzögerungen festlegen.

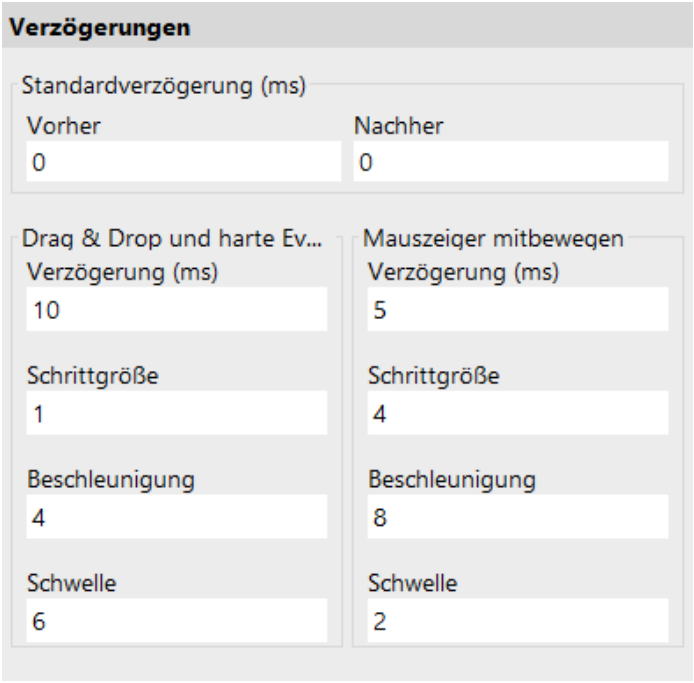


Abbildung 41.27: Verzögerungsoptionen

Standardverzögerung (System)

Server Skript Name: OPT_PLAY_DELAY_BEFORE, OPT_PLAY_DELAY_AFTER
Diese Werte legen die Verzögerung vor und nach der Ausführung jedes Schrittes fest, der keine eigene Verzögerung gesetzt hat. Für Demonstrationszwecke oder zum Debuggen von Tests kann es sinnvoll sein, eine gewisse Verzögerung einzustellen. Im Normalbetrieb sollten Tests problemlos ohne Standardverzögerung laufen.

Drag&Drop und Interpolation von Mausbewegungen

Die Simulation von Drag&Drop ist knifflig und nur möglich, indem ”harte” Mausevents generiert werden, die den Mauszeiger tatsächlich bewegen. Unter Windows kann es dabei unter anderem zu Konflikten mit verschiedenen Maustreibern kommen. Informationen zu Drag&Drop finden Sie auch unter Abschnitt 49.1 (1021).

Um Drag&Drop so zuverlässig wie möglich zu machen, sind die Bewegungen des Mauszeigers vielfältig konfigurierbar. Wegen der großen Unterschiede in den Anforderungen für Drag&Drop und harte Mausevents einerseits und Mausbewegungen, die nur zur Visualisierung dienen, andererseits, gibt es zwei komplette Sätze von Optionen für Maus-

bewegungen. Die Werte für Demo-Mausbewegungen werden ignoriert, wenn die zugehörige Option Mauszeiger tatsächlich bewegen(543) nicht gesetzt ist.

Im Normalfall sollten die Bewegungen für Drag&Drop und harte Events langsamer sein und mehr Interpolationsschritte verwenden, als Demo-Mausbewegungen. Letztere könnten ansonsten Tests deutlich verlangsamen. Alle der folgenden Optionen haben Einfluss auf die Geschwindigkeit des Mauszeigers und Sie werden eventuell ein wenig experimentieren müssen, um den gewünschten Effekt zu erzielen.

Verzögerung (ms) (System)

SUT Skript Name: OPT_PLAY_DND_DELAY, OPT_PLAY_MOVEMOUSE_DELAY
Nach jeder einzelnen Mausbewegung wartet QF-Test bis die angegebene Anzahl Millisekunden verstrichen ist. Dieser Wert sollte zwischen 2 und 20 liegen, falls Interpolation eingeschaltet ist, andernfalls zwischen 20 und 200. Mit Interpolation ist 10 ein guter Wert für Drag&Drop , 5 für Demo-Mausbewegungen.

Schrittgröße (System)

SUT Skript Name: OPT_PLAY_DND_STEP, OPT_PLAY_MOVEMOUSE_STEP
Gibt die Größe der Schritte bei der Interpolation der Mausbewegungen an. Ein Wert von 0 schaltet die Interpolation aus. Gute Werte liegen zwischen 1 und 3 für Drag&Drop und zwischen 2 und 10 für Demo-Mausbewegungen.

Beschleunigung (System)

SUT Skript Name: OPT_PLAY_DND_ACCELERATION, OPT_PLAY_MOVEMOUSE_ACCELERATION
Um sinnlose Verzögerungen zu vermeiden, können längere Mausbewegungen beschleunigt werden. Ein Wert von 0 schaltet die Beschleunigung aus. Sinnvolle Werte liegen zwischen 1 für sehr geringe und 10 oder mehr für starke Beschleunigung. Gut geeignet sind Werte zwischen 3 und 5 für Drag&Drop und zwischen 6 und 20 für Demo-Mausbewegungen.

Schwelle (System)

SUT Skript Name: OPT_PLAY_DND_THRESHOLD, OPT_PLAY_MOVEMOUSE_THRESHOLD
Um sicherzustellen, dass kleine Bewegungen, sowie Beginn und Ende einer Bewegung präzise ausgeführt werden, wird die Beschleunigung für Mausbewegungen ausgeschaltet, die weniger Schritte als diese Schwelle benötigen. Gute Werte liegen zwischen 4 und 8 für Drag&Drop und zwischen 0 und 6 für Demo-Mausbewegungen.

41.3.6 Automatische Timeouts

Diese automatischen Timeouts reduzieren den Aufwand beim Erstellen einer Testsuite gewaltig. Sie legen fest, wie lange im SUT auf ein Ereignis wie das Erscheinen einer benötigten Komponente gewartet wird, bevor ein Fehler auftritt.

Sie sollten die folgenden Werte nicht zu klein wählen, damit ein ”kleiner Schluckauf” des Testrechners nicht einen ansonsten einwandfreien Test abbricht. QF-Test wartet außerdem nicht bei jedem Schritt, bis die maximale Zeit verstrichen ist, sondern arbeitet sofort weiter, sobald die Voraussetzungen erfüllt sind.

Abbildung 41.28: Timeout Optionen

Erkennen von Deadlocks (s) (System)

Server Skript Name: OPT_PLAY_TIMEOUT_DEADLOCK
Wenn das SUT für den angegebenen Zeitraum nicht reagiert wird eine

DeadlockTimeoutException[(960)] geworfen. Ein Wert von 0 unterdrückt die Erkennung von Deadlocks.

Warten auf nicht vorhandene Komponente (ms) (System)

SUT Skript Name: OPT_PLAY_TIMEOUT_COMPONENT
Die maximale Zeit, die QF-Test darauf wartet, dass die Zielkomponente eines Events oder eines Checks im SUT verfügbar ist. Direkt nach Herstellung der Verbindung mit dem SUT wird diese Option vorübergehend auf mindestens 30000 gesetzt, um dem SUT Zeit zur Initialisierung zu geben.

Warten auf GUI-Engine (ms) (System)

SUT Skript Name: OPT_PLAY_TIMEOUT_ENGINE
Diese Option ist hilfreich für SUTs mit mehreren Engines, z.B. Eclipse mit eingebetteten Swing Komponenten. Ein Warten auf Client[(758)] Knoten ist fertig, sobald sich die erste Engine mit QF-Test verbindet, sofern nicht sein GUI-Engine[(759)] Attribut das Warten auf eine bestimmte Engine vorschreibt. Um einen darauf folgenden Warten auf Komponente[(876)] Knoten für eine Komponente der falschen Engine vor dem sofortigen Scheitern zu bewahren, wartet QF-Test zunächst die hier angegebene Zeitspanne ab, um der zweiten GUI-Engine eine Chance zu geben, sich ebenfalls mit QF-Test zu verbinden.

Warten auf nicht vorhandenes Element (ms) (System)

SUT Skript Name: OPT_PLAY_TIMEOUT_ITEM
Bezieht sich ein Event auf ein Unterelement einer Komponente, wartet QF-Test wie oben beschrieben zunächst darauf, dass die Komponente im SUT verfügbar ist. Anschließend gibt es dem SUT für den hier festgelegten Zeitraum die Chance, das gewünschte Unterelement bereitzustellen.

Standard-Wartezeit für Checks (ms) (System)

Server Skript Name: OPT_PLAY_CHECK_TIMEOUT
Diese Option legt einen Standardwert für das Wartezeit[(809)] Attribute von Check Knoten fest, bei denen dieses Attribute nicht explizit gesetzt ist und die einen ”echten” Check im Report darstellen und nicht zur Testablaufsteuerung dienen, d.h. die weder eine Exception werfen noch eine Ergebnisvariable setzen oder einen @report Doctag haben.

Falls Ihre Tests viele Check Knoten ohne explizites Wartezeit enthalten, bei denen davon ausgegangen wird, dass sie fehlschlagen - was für die obigen ”echten”

Checks unwahrscheinlich ist - können Sie gegebenenfalls die Tests beschleunigen, indem Sie diese Option auf 0 setzen. Allerdings wäre es in diesem Fall vorzuziehen, stattdessen die Wartezeit Attribute der betroffenen Knoten auf 0 zu setzen und diese Option unverändert zu lassen, da sie die Stabilität der Ausführung von Checks generell verbessert.

Warten bei modalem Dialog (ms) (System)

SUT Skript Name: OPT_PLAY_TIMEOUT_MODAL
Wird ein Event an eine Komponente geschickt, deren Fenster von einem modalen Dialog blockiert ist, wird eine `ModalDialogException`(959) geworfen. Allerdings werden modale Dialoge oft nur temporär angezeigt, um den Anwender über einen etwas länger dauernden Vorgang zu informieren. Ist diese Option auf einen Wert größer 0 gesetzt, wartet QF-Test zunächst die angegebene Zeit, bevor die Exception geworfen wird. Verschwindet der Dialog vorher, wird der Testlauf normal fortgesetzt. Dadurch wird die Behandlung von temporären modalen Dialogen stark vereinfacht.

Hinweis

Ist die Option Öffnen eines Fensters in Warten auf Komponente(876) konvertieren (510) aktiviert, kann bei der Aufnahme einer Sequenz während der ein temporärer modaler Dialog angezeigt wird, ein Warten auf Komponente(876) Knoten angelegt werden. Wird der Dialog nur kurz angezeigt, sollte dieser Knoten entfernt werden, um Timing Probleme zu vermeiden. Falls das SUT häufig temporäre modale Dialoge einsetzt, ist es sinnvoll, die Option Öffnen eines Fensters in Warten auf Komponente(876) konvertieren (510) zu deaktivieren.

Warten bei 'busy' GlassPane (ms) (System)

Swing

SUT Skript Name: OPT_PLAY_TIMEOUT_GLASSPANE
Alternativ zu temporären modalen Dialogen wird in manchen Fällen eine sogenannte GlassPane in Verbindung mit einem Mauszeiger in Sanduhrform eingesetzt um anzuzeigen, dass die Anwendung beschäftigt, also 'busy' ist. Eine GlassPane ist eine unsichtbare Komponente, die ein ganzes Fenster überdeckt und alle Events für dieses Fenster abfängt. Dadurch wird die normale Eventverarbeitung verhindert, was einen Testlauf völlig aus der Bahn werfen kann.

Diese Situation behandelt QF-Test automatisch, indem es auf das Verschwinden dieser 'busy' GlassPane wartet bevor es einen Event, Check etc. ausführt. Ist die GlassPane nach Verstreichen des in dieser Option angegebenen Timeouts immer noch aktiv, wird eine `BusyPaneException`(960) geworfen.

Ist diese Option auf 0 gesetzt, wird nicht auf eine 'busy' GlassPane geprüft, sondern der Event auf jeden Fall ausgeliefert. Eine `BusyPaneException` wird in diesem Fall nie geworfen.

Ein Sonderfall ist der Warten auf Komponente(876) Knoten. Wird auf eine Komponente gewartet (nicht auf deren Abwesenheit), die von einer 'busy' GlassPane verdeckt wird, so wird das Attribut Wartezeit(878) des Knotens gleichzeitig als Wartezeit für die 'busy' GlassPane herangezogen. Auf diesem Weg können Fälle behandelt werden, in denen die Anwendung erwartungsgemäß lange beschäftigt ist, ohne dafür diese Option generell hochzusetzen.

Warten auf Button/Menü-Aktivierung (ms) (System)

SUT Skript Name: OPT_PLAY_TIMEOUT_ENABLED
Ein Mausklick, der an ein Menü oder einen Button geschickt wird, wird einfach ignoriert, wenn das Menü oder der Button noch nicht aktiviert sind. Der weitere Ablauf des Tests kommt damit ziemlich sicher durcheinander.

Mit diesem Wert legen Sie fest, wie lange zunächst gewartet werden darf, ob die Zielkomponente vielleicht doch noch aktiviert wird, bevor eine `DisabledComponentException`(959) geworfen wird. Diese Exception kann durch Deaktivieren der Option DisabledComponentExceptions werfen(544) unterdrückt werden.

Warten auf Fokus (ms) (System)

SUT Skript Name: OPT_PLAY_TIMEOUT_FOCUS
Ist dieser Wert gesetzt, wartet QF-Test vor dem Abspielen von Tastaturevents(780) darauf, dass die Zielkomponente den Fokus besitzt. Diese Option kann einen Test signifikant ausbremsen, wenn die Komponente den Fokus nicht bekommt, daher sollten Sie sie nicht höher als 100 setzen. Ein guter Wert ist 20.

Zeitabstand für Suche nach Komponenten (ms) (System)

SUT Skript Name: OPT_PLAY_POLL_COMPONENT
Wenn QF-Test im SUT auf das Erscheinen von Komponenten wartet, kann es sich nicht alleine auf den Event Mechanismus verlassen, sondern muss in regelmäßigen Abständen die Suche wiederholen. Der Zeitabstand für diese Suche wird mit Hilfe dieser Option festgelegt.

Zeitabstand für Suche nach Unterelementen (ms) (System)

SUT Skript Name: OPT_PLAY_POLL_ITEM
Um im SUT auf ein nicht vorhandenes Unterelement einer komplexen Komponente zu warten, kann QF-Test sich nicht auf den Eventmechanismus verlassen, sondern muss immer wieder die Komponente nach dem Unterelement absuchen. Diese Option bestimmt, in welchen Zeitabständen diese Suche ausgeführt wird.

Zeitabstand für Checkwiederholung (ms) (System)

SUT Skript Name: OPT_PLAY_POLL_CHECK
Schlägt ein Check[805] fehl, für den eine Wartezeit[809] vorgesehen ist, überprüft QF-Test immer wieder den Zustand der Komponente, bis dieser den Vorgaben entspricht oder die Wartezeit verstrichen ist. Hiermit legen Sie die Zeitabstände fest, in denen diese Überprüfung durchgeführt wird.

41.3.7 Rückwärtskompatibilität

Diese Optionen stellen älteres Verhalten von QF-Test wieder her, welches im Laufe des Entwicklungszykluses so stark verändert wurde, dass die Rückwärtskompatibilität nicht mehr garantiert werden konnte.

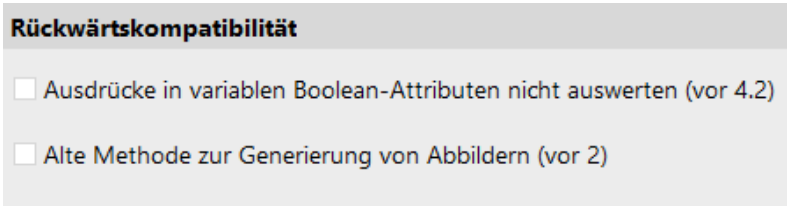


Abbildung 41.29: Optionen für Wiedergabe Rückwärtskompatibilität

Ausdrücke in variablen Boolean-Attributen nicht evaluieren (vor 4.2) (System)

4.2+

Server Skript Name: OPT_PLAY_DONT_EVALUATE_BOOLEAN_OPTIONS
In variablen Attributen, die einen boolean Wert beinhalten, z.B. das Als ”harten” Event wiedergeben Attribut von Mausevent Knoten oder das Modal Attribut von Fenster Knoten, wird ein angegebener Ausdruck seit 4.2.0 von Jython ausgewertet.

Alte Methode zur Generierung von Abbildern (vor 2) (System)

Swing

SUT Skript Name: OPT_RECORD_CHECK_IMAGE_OLD_STYLE
Früher wurde bei der Aufnahme von Check Abbild[828] Knoten für durchsichtige Komponenten immer ein schwarzer Hintergrund aufgenommen. Dadurch stimmte eventuell die Darstellung nicht mit dem Bild überein, das der Anwender zu sehen bekommt. Dieser Fehler wurde korrigiert, so dass normalerweise nun der Hintergrund korrekt gezeichnet wird. Mit dieser Option kann wieder auf die alte, fehlerhafte Methode umgeschaltet werden, falls bereits eine große Zahl von Checks von durchsichtigen Komponenten mit der alten Methode erstellt wurden.

41.4 SmartID und qfs:label

7.0+

Die folgenden Einstellungen legen Details der Aufnahme und Wiedergabe von SmartIDs und qfs:label* Varianten fest. Ausführliche Informationen hierzu finden Sie in Abschnitt 5.6[81] und Abschnitt 5.4.4[74].

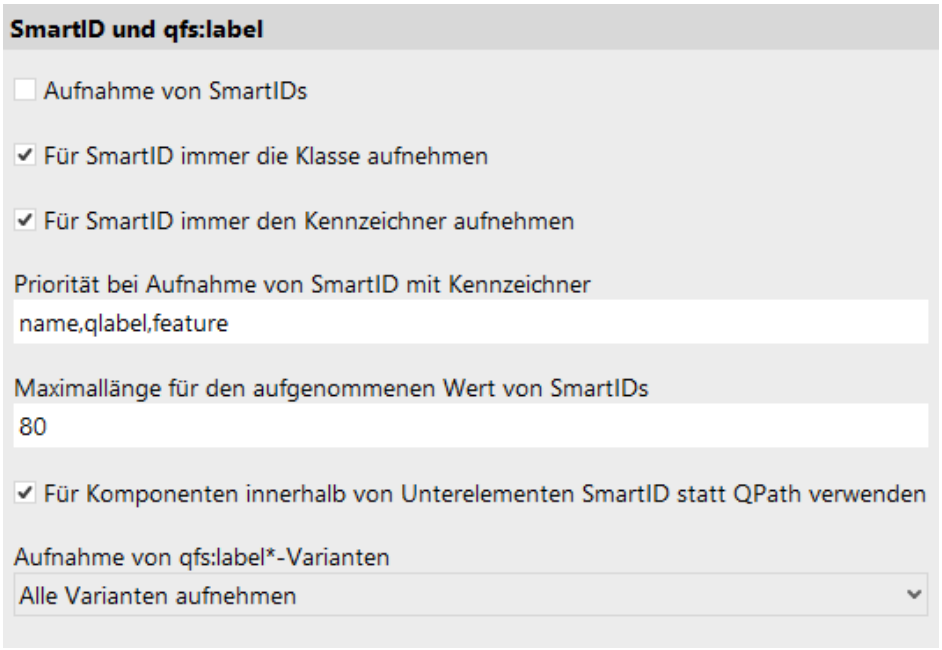


Abbildung 41.30: SmartID und qfs:label-Optionen

Aufnahme von SmartIDs (System)

6.0+

Server (automatisch weiter an SUT) Skript Name: OPT_RECORD_SMARTID
Ist diese Option aktiv, werden bei der Aufnahme - so weit möglich - SmartIDs statt Komponenten aufgezeichnet.

Für SmartID immer die Klasse aufnehmen (System)

6.0+

Server (automatisch weiter an SUT) Skript Name: OPT_RECORD_SMARTID_CLASS
Für SmartIDs ist das Voranstellen der Klasse der Zielkomponente optional. Mit dieser Option wird festgelegt, ob die Klasse beim Aufzeichnen von SmartIDs immer vorangestellt wird oder nur, wenn es für die Eindeutigkeit notwendig ist. Die Option ist standardmäßig aktiv, da die Klasse in der SmartID - neben der Lesbarkeit und Klarheit - die Performanz bei der Wiedergabe deutlich verbessert.

Für SmartID immer den Kennzeichner aufnehmen (System)

7.0+

Server (automatisch weiter an SUT) Skript Name: OPT_RECORD_SMARTID_QUALIFIER
Diese Option legt fest, ob für SmartIDs der Kennzeichner aufgenommen wird. In folgenden Situationen hat die Option keine Wirkung, so dass der Kennzeichner immer aufgenommen wird:

- Wenn es sich bei der SmartID um eine qfs:label*-Variante handelt und die Option Aufnahme von qfs:label*-Varianten(561) auf ”Alle Varianten aufnehmen” oder ”Nur spezifisches Label aufnehmen” gesetzt ist.
- Wenn es sich bei der SmartID um ein weiteres Merkmal handelt, das nicht zu den qfs:label*-Varianten gehört und über die Option Priorität bei Aufnahme von SmartIDs mit Kennzeichner(560) aufgenommen wird.
- Wenn die Option Priorität bei Aufnahme von SmartIDs mit Kennzeichner(560) von ihrem Standardwert abweicht und die aufgenommene SmartID nicht auf dem Namen der Komponente basiert.

Priorität bei Aufnahme von SmartIDs mit Kennzeichner (System)

7.0+

Server (automatisch weiter an SUT) Skript Name: OPT_RECORD_SMARTID_PRIORITIES
Diese komma-getrennte Liste der Kennzeichner gibt an, in welcher Reihenfolge die Wiedererkennungskriterien analysiert werden sollen. Für den Standardwert ”`name`,`qlabel`,`feature`” wird bei der Aufnahme zunächst auf einen Namen geprüft. Ist dieser vorhanden, wird er für die SmartID verwendet. Andernfalls folgt als nächstes der Test auf eine qfs:label*-Variante und schließlich auf ein Merkmal. Die Namen der verfügbaren Kennzeichner finden Sie in SmartID(81).

Hinweis

Bei der Wiedergabe mit einer SmartID wird zunächst nur nach dem Namen gesucht, dann nach qfs:label* und Merkmal, die als gleichwertig gelten und implizit kombiniert werden.

Maximallänge für den aufgenommenen Wert von SmartIDs (System)

7.0+

Server (automatisch weiter an SUT) Skript Name: OPT_SMARTID_MAX_VALUE_LENGTH
In einzelnen Fällen kann das Merkmal oder zugeordnete Label einer Komponente sehr lang sein. Dies ist kein Problem als solches und wird oft gar nicht bemerkt, wenn der Wert in einem Komponente(930) Knoten gespeichert ist. Bei einer SmartID kann es allerdings sehr unhandlich sein. Der Einfachheit halber werden daher Werte, die länger sind, als in dieser Option angegeben, automatisch in einen regulären Ausdruck der passenden Länge umgewandelt.

Für Komponenten innerhalb von Unterelementen SmartID statt QPath verwenden (System)

7.0+

Server (automatisch weiter an SUT) Skript Name: OPT_RECORD_SMARTID_INSTEAD_OF_QPATH
Eine Komponente innerhalb eines Unterelements, z.B. eine CheckBox in einer Tabellenzelle, muss durch eine spezielle Syntax für Pseudo-Elemente repräsentiert werden. Ab QF-Test Version 7 löst dafür SmartID das veraltete QPath-Modell ab. Nach Deaktivieren dieser Option wird QPath verwendet, sofern die generelle Aufnahme von SmartIDs ebenfalls deaktiviert ist.

Aufnahme von qfs:label*-Varianten (System)

7.0+

SUT Skript Name: OPT_RECORD_QFSLABEL_MODE
Mögliche Werte: VAL_RECORD_QFSLABEL_MODE_ALL,
VAL_RECORD_QFSLABEL_MODE_SPECIFIC,
VAL_RECORD_QFSLABEL_MODE_BEST,
VAL_RECORD_QFSLABEL_MODE_LEGACY
Die Option gibt an, welche `qfs:label*`-Varianten(74) als Weitere Merkmale(933) aufgezeichnet werden:

- Der Wert ”Alle Varianten aufnehmen” bewirkt, dass alle für die Komponente gefundenen `qfs:label*`-Varianten in den Weiteren Merkmalen abgespeichert werden. Die Beste Beschriftung(76) erhält den Status ”Sollte übereinstimmen”, die anderen den Status ”Ignorieren”. Bei einer SmartID(81) wird der spezifische Kennzeichner, siehe Tabelle `qfs:label*`-Varianten(75), aufgezeichnet.
- Mit dem Wert ”Nur spezifisches Label aufnehmen” wird nur die als Beste Beschriftung(76) bewertete `qfs:label*`-Variante in den Weiteren Merkmalen abgespeichert. Bei einer SmartID(81) wird der spezifische Kennzeichner, siehe Tabelle `qfs:label*`-Varianten(75), aufgezeichnet.
- Der Wert ”Nur qfs:labelBest aufnehmen” bewirkt, dass nur die als Beste Beschriftung(76) bewertete `qfs:label*`-Variante aufgenommen und in den Weiteren Merkmalen unter dem Namen `qfs:labelBest` abgespeichert wird. Bei der Aufzeichnung einer SmartID(81) mit dieser Einstellung, wird je nach dem wie die Option Für SmartID immer den Kennzeichner aufnehmen(559) gesetzt ist, entweder kein Kennzeichner oder der für die Beschriftung spezifische Kennzeichner aufgezeichnet.
- Beim Wert ”Alter qfs:label-Modus” wird die beste Beschriftung über den vor QF-Test 7.0 verwendeten Algorithmus ermittelt und `qfs:label` in den Weiteren Merkmalen abgespeichert. Bei einer SmartID(81) wird `qlabel` als Kennzeichner aufgezeichnet, jedoch nicht standardmäßig. ().

41.5 Android

6.0+

Die folgenden Einstellungen haben Einfluss auf Tests von Android-Anwendungen.

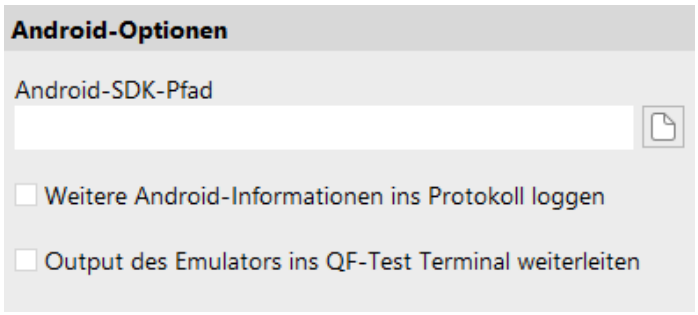


Abbildung 41.31: Android-Optionen

Android-SDK-Pfad (System)

Server Skript Name: OPT_ANDROID_SDK_PATH
Geben Sie hier den Installationspfad Ihres Android-SDK an. Der Name dieses Verzeichnisses lautet üblicherweise `sdk` und enthält unter anderem ein Unterverzeichnis `tools` oder `cmdline-tools`.

Diese Einstellung muss nur gesetzt werden, falls QF-Test den Pfad nicht automatisch ermitteln kann.

Weitere Android-Informationen ins Protokoll loggen (System)

Server Skript Name: OPT_ANDROID_DEBUG
Wenn diese Option aktiviert ist werden weitere Informationen in das Protokoll geloggt.

Output des Emulators ins QF-Test Terminal weiterleiten (System)

Server Skript Name: OPT_ANDROID_FORWARD_EMULATOR_OUTPUT
Wenn diese Option aktiviert ist und ein Emulator über den Android-Emulator starten Knoten gestartet wurde, dann werden die stdout/stderr Ausgaben des gestarteten Emulators an das QF-Test Terminal weitergegeben.

41.6 iOS

8.0+

Diese Optionen stehen für das Testen von iOS Anwendungen zur Verfügung:

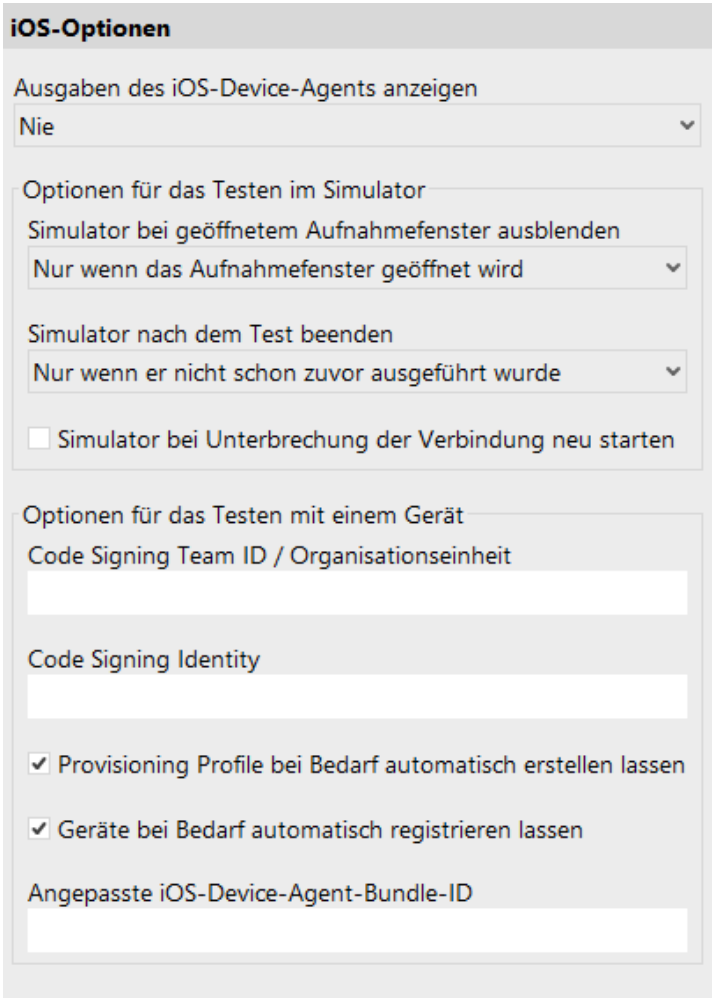


Abbildung 41.32: Options for iOS Tests

Ausgaben des iOS-Device-Agents anzeigen (User)

SUT Skript Name: OPT_IOS_PRINT_AGENT_OUTPUT
Mögliche Werte: VAL_IOS_PRINT_AGENT_OUTPUT_NONE,
VAL_IOS_PRINT_AGENT_OUTPUT_INSTRUMENT,
VAL_IOS_PRINT_AGENT_OUTPUT_EXEC,
VAL_IOS_PRINT_AGENT_OUTPUT_ALL
Während des Starts und der Ausführung von iOS Tests wird über Xcode der

benötigte Geräteagent gebaut und ausgeführt. Bei normaler Ausführung wird die detaillierte Prozessausgabe nicht in das Terminal geschrieben.

Wenn man aber Fehler sucht, kann die Prozessausgabe im Terminal hilfreich sein. Sie kann für die Instrumentierung des Geräts, die Ausführung oder beides aktiviert werden.

Simulator bei geöffnetem Aufnahmefenster ausblenden (User)

SUT Skript Name: OPT_IOS_AUTO_HIDE_SIMULATOR
Mögliche Werte: VAL_IOS_AUTO_HIDE_SIMULATOR_ALWAYS,
VAL_IOS_AUTO_HIDE_SIMULATOR_NEVER,
VAL_IOS_AUTO_HIDE_SIMULATOR_ONOPEN
Die Interaktion mit der Anwendung bei der Testaufnahme und beim Untersuchen von Komponenten erfolgt über ein dediziertes Fenster, analog zu Android - siehe Aufnamen und Checks bei iOS(282). Wenn die zu testende Anwendung auf einem Simulator ausgeführt wird, kann es verwirrend sein, wenn die Benutzeroberfläche doppelt angezeigt wird - im Simulator und im Aufnahmefenster.

Um diese Verwirrung zu vermeiden, kann QF-Test den Simulator automatisch ausblenden, wenn das Aufnahmefenster geöffnet wird. Wenn dann explizit zum Simulator gewechselt wird, erscheint dessen Fenster wieder. Man kann jedoch mit der Option ”Immer” einstellen, dass beim Auswählen des Simulators dieser immer wieder ausgeblendet und stattdessen das Aufnahmefenster aktiviert wird - solange dieses vorhanden ist.

Simulator nach dem Test beenden (User)

SUT Skript Name: OPT_IOS_AUTO_CLOSE_SIMULATOR
Mögliche Werte: VAL_IOS_AUTO_CLOSE_SIMULATOR_YES,
VAL_IOS_AUTO_CLOSE_SIMULATOR_NO,
VAL_IOS_AUTO_CLOSE_SIMULATOR_AUTO
Für die Steuerung des iOS-Geräts oder des Simulators verwendet QF-Test einen Controller. Bei Bedarf wird der iOS Simulator von QF-Test gemeinsam mit dem Controller gestartet. Standardmäßig wird der Simulator ebenfalls beendet, wenn der Controller gestoppt wird.

Bei der Testentwicklung kann es hilfreich sein, den Simulator laufen zu lassen, auch wenn der QF-Test Controller gestoppt wurde. Hier kann festgelegt werden, dass der Simulator nicht gemeinsam mit den Controller beendet werden soll, beziehungsweise nur, wenn er vom Controller gestartet wurde.

Simulator bei Unterbrechung der Verbindung neu starten (System)

SUT Skript Name: OPT_IOS_RESTART_SIMULATOR

Wenn die Verbindung zu einem echten iOS-Gerät bei einem Testlauf unterbrochen wird, führt QF-Test automatisch einen Neustart des Geräteagenten aus, um die Verbindung wiederherzustellen. Wenn das iOS-Zielgerät simuliert ist, geht QF-Test standardmäßig davon aus, dass der Simulator absichtlich geschlossen wurde, um den Testlauf zu beenden. Mit dieser Option ist es möglich, die automatische Wiederherstellung auch für Simulatorverbindungen einzustellen, einschließlich Neustart der Simulator App.

Code Signing Team ID / Organisationseinheit (System)

SUT Skript Name: OPT_XCODE_DEVELOPMENT_TEAM
QF-Test kann Test auf Apps ausführen, die direkt auf einem iOS-Gerät laufen. Wegen Betriebssystemeinschränkungen muss der Geräteagent, der temporär auf dem Gerät installiert ist, um die benötigten Interaktionen durchzuführen, automatisch mit einem gültigen iPhone Entwicklerzertifikat signiert werden. Für die Identifizierung des Zertifikats muss die Team ID (auch als ”Certificate Organizational Unit” bekannt) zur Verfügung gestellt werden.

Die Team-ID ist eine eindeutige von Apple generierte Zeichenfolge mit 10 Zeichen, die Ihrem Team zugewiesen wird. Die Team ID finden Sie im Feld ”Organisationseinheit” im iPhone-Entwicklerzertifikat Ihres Schlüssels. Sie können Ihre Team ID über Ihren Entwicker-Account finden. Melden Sie sich bei https://developer.apple.com/account an und gehen Sie zu ”Mitgliedschaftsdetails”. Ihre Team ID steht im Bereich zu den Mitgliedschaftsinformationen unter dem Teamnamen.

Um ein Entwicklerzertifikat zu erstellen, öffnen Sie die Einstellungen von Xcode, selektieren den Reiter für den Account und fügen Ihren Entwicker-Account unter Verwendung Ihrer Apple ID hinzu.

Code Signing Identity (System)

SUT Skript Name: OPT_XCODE_CODE_SIGN_IDENTITY
Normalerweise kann dieser Wert leer bleiben, da Xcode automatisch die Signing ID aus dem Zertifikat, das über die Team ID spezifiziert ist, ableitet. Manchmal jedoch muss eine dedizierte Signing ID bereitgestellt werden (üblicherweise Apple Developer oder iPhone Developer).

Provisioning Profile bei Bedarf automatisch erstellen lassen (System)

SUT Skript Name: OPT_XCODE_ALLOW_PROVISIONING_UPDATES
Wenn aktiv, kann von Xcode automatisch ein ”Provisioning Profile” bei der Instrumentierung des Geräts erstellt werden, um den Agenten auf dem angebundenen Gerät auszuführen.

Geräte bei Bedarf automatisch registrieren lassen (System)

SUT Skript Name: OPT_XCODE_ALLOW_PROVISIONING_DEVICE_REGISTRATION
Wenn aktiv, kann ein neues Gerät bei der Instrumentierung automatisch von Xcode registriert werden.

Angepasste iOS-Device-Agent-Bundle-ID (System)

SUT Skript Name: OPT_IOS_AGENT_BUNDLE_ID
Es kann vorkommen, dass Xcode bei der Erstellung eines ”Provisioning Profile” scheitert - insbesondere, wenn ein freier Entwicker-Account verwendet wird. Hiermit ist es möglich, die Bundle ID für den Agenten manuell so zu ändern, das XCode sie akzeptiert.

41.7 Web-Optionen

Web

Die folgenden Optionen sind speziell für das Testen von Web-Anwendungen von Bedeutung.

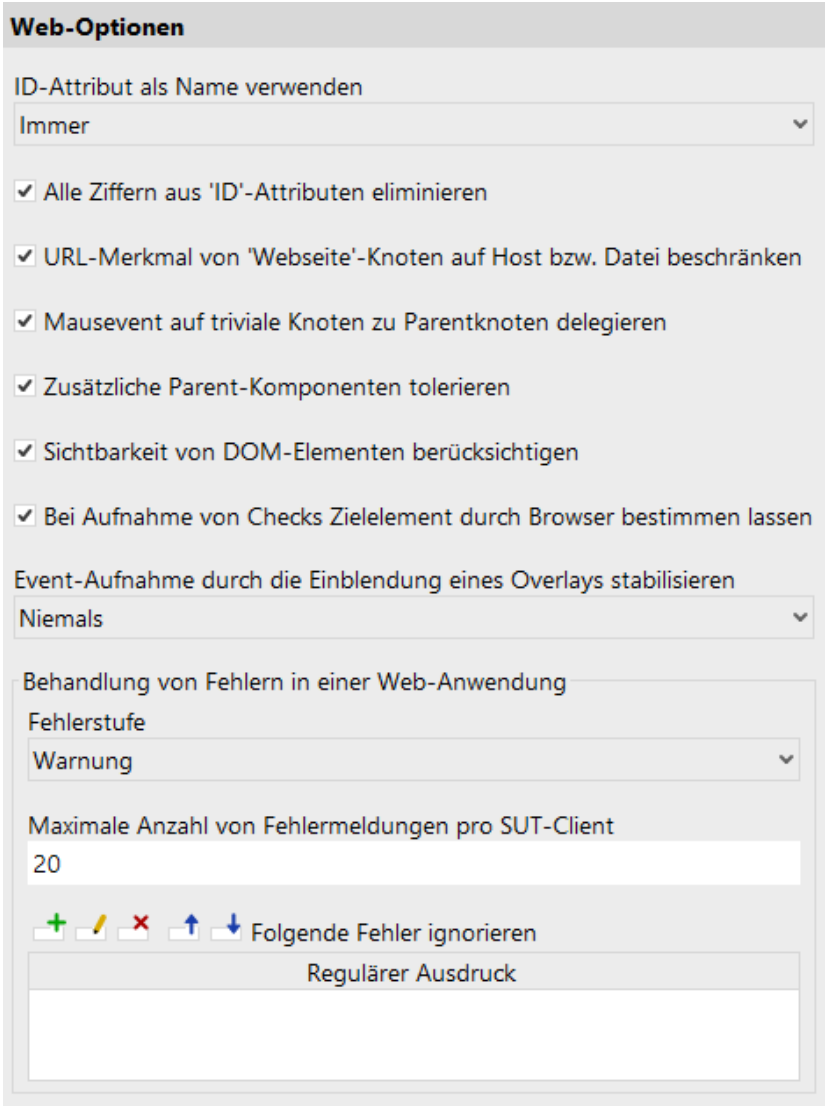


Abbildung 41.33: Web-Optionen

ID-Attribut als Name verwenden (System)

8.0+

SUT Skript Name: OPT_WEB_ID_AS_NAME
Die ID eines DOM-Knotens wird von QF-Test standardmäßig als Name der Komponente verwendet (Wert ”Immer”). Mit dieser Option kann dieses Verhalten deaktiviert werden (Wert ”Nie”), oder der sepzielle Algorithmus aus QF-Test Versionen vor 8.0 angewendet werden (Wert ”Nur wenn eindeutig”).

In letzterem Fall wird die ID eines DOM-Knotens nur dann als Name der Komponente verwendet, wenn die ID hinreichend eindeutig ist. Hierbei wird Eindeutigkeit

pro Knotentyp und in Abhängigkeit von den Optionen Gewichtung von Namen (Wiedergabe)(548) und Gewichtung von Namen (Aufnahme)(520) interpretiert. Details finden Sie im Abschnitt "Web" in Komponentenbezeichner(66).

Alle Ziffern aus 'ID'-Attributen eliminieren (System)

SUT Skript Name: OPT_WEB_SUPPRESS_NUMERALS

Hinweis

Natürlich beeinflusst diese Option nur die Art und Weise, in der QF-Test 'ID'-Attribute behandelt. Die Attribute selbst werden nicht geändert, sonst würde die Anwendung sehr wahrscheinlich nicht mehr funktionieren.

Ist diese Option aktiv, entfernt QF-Test alle Ziffern aus 'ID'-Attributen, um dadurch Probleme mit automatisch generierten IDs zu verhindern, wie sie häufig in Ajax Frameworks wie GWT vorkommen. Solche dynamischen IDs können sich bei der kleinsten Modifikation der Anwendung komplett ändern, was zum Fehlschlagen von Tests führt, besonders wenn IDs als Namen verwendet werden. Durch Entfernen der dynamischen Anteile werden IDs zwar weniger hilfreich, da sie nicht mehr eindeutig sind, aber auch weniger schädlich. Um die Eindeutigkeit bei der Verwendung als Name kümmert sich QF-Test. Da IDs auch als Basis für Merkmal(932) und Weitere Merkmale(933) Attribute fungieren, ist diese Option selbst dann hilfreich, wenn IDs nicht als Namen verwendet werden.

URL-Merkmal von 'Webseite'-Knoten auf Host bzw. Datei beschränken (System)

SUT Skript Name: OPT_WEB_LIMIT_URL

Falls diese Option gesetzt ist werden alle Webseiten, die vom selben Host kommen, als die selbe Seite aufgenommen. Hierzu wird die URL im Merkmal auf den Host reduziert. Dies ist oft hilfreich wenn die Seiten ähnliches Aussehen und Navigations-Struktur haben.

Für Datei-URLs wird die URL entsprechend auf den Dateinamen beschränkt, die Verzeichnisse werden entfernt.

Mausevent auf triviale Knoten zu Parentknoten delegieren (System)

SUT Skript Name: OPT_WEB_RETARGET_MOUSE_EVENT

Beim Aufnehmen von Mausevent auf DOM-Knoten einer Webseite ist es oft sinnvoll "triviale" Knoten zu ignorieren und sich auf die wesentlichen Knoten zu konzentrieren. Beim Klick auf einen Text-Hyperlink ist es z.B. normalerweise nicht von Interesse, ob das erste Wort fett formatiert ist. Der Link ist entscheidend.

Ist diese Option gesetzt, nimmt QF-Test nicht einfach den Event für den tiefstliegenden DOM-Knoten unter dem Mauszeiger auf, sondern geht die Hierarchie nach oben, bis es einen "interessanten" Knoten findet. In obigem Beispiel würde

der Event mit gesetzter Option auf den A Knoten, ohne die Option auf den darin enthaltenen B Knoten aufgenommen.

Zusätzliche Parent-Komponenten tolerieren (System)

SUT Skript Name: OPT_WEB_TOLERATE_INTERMEDIATE_PARENT
Normalerweise ist die Wiedererkennung von Komponenten in QF-Test so flexibel, dass Änderungen an der Hierarchie der Komponenten weitgehend toleriert werden. Für Webseiten mit tief verschachtelten Tabellen kann dies zu Einbrüchen bei der Performanz führen, da die möglichen Varianten zur Erkennung der Zielkomponente exponentiell mit der Tiefe der Verschachtelung wachsen. Wenn Sie auf solche Probleme stoßen, versuchen Sie diese Option zu deaktivieren. Dies wird die Flexibilität reduzieren, sollte aber bei der Performanz helfen.

Hinweis

Die wesentlich bessere Lösung ist die Vergabe von eindeutigen 'ID'-Attributen für die verschiedenen Tabellen und andere Komponenten, so dass QF-Test's Mechanismus für die Namenserkennung greifen kann. Dies beschleunigt die Wiedererkennung nicht nur drastisch, es macht sie auch robuster gegenüber Änderungen.

Sichtbarkeit von DOM-Elementen berücksichtigen (System)

SUT Skript Name: OPT_WEB_TEST_VISIBILITY
Wie bei AWT/Swing oder SWT erkennt QF-Test normalerweise nur sichtbare DOM-Knoten als Zielkomponenten. Allerdings ist die Sichtbarkeit bei DOM-Knoten nicht immer so wohldefiniert wie bei Komponenten in einem Java-GUI. So ist es z.B. möglich, dass ein unsichtbarer DOM-Knoten selbst sichtbare Kinder hat. Insbesondere bei Webseiten, die nicht standardkonformes HTML enthalten, kann es vorkommen, dass ein DOM-Knoten als unsichtbar eingestuft wird, obwohl er im Browser-Fenster dargestellt wird. Falls Sie Probleme mit einem solchen Fall haben, können Sie diese Option ausschalten.

Bei Aufnahme von Checks Zielelement durch Browser bestimmen lassen (System)

SUT Skript Name: OPT_WEB_CHECK_VIA_BROWSER
Beim Aufnehmen von Checks, Komponenten oder Prozeduren muss QF-Test die jeweilige Zielkomponente ermitteln, die sich gerade unter dem Mauszeiger befindet. Bei überlappenden Knoten gibt es zwei Möglichkeiten, den korrekten zu ermitteln. Standardmäßig überlässt QF-Test dem Browser die Entscheidung, was normalerweise die beste Variante ist. Da sich die verschiedenen Browser nicht immer zuverlässig gleich verhalten, können Sie bei Problemen durch ausschalten dieser Option auf den älteren Mechanismus wechseln, der auf der Anordnung der

Elemente basiert. Für die Wiedergabe von Checks hat diese Option keine Bedeutung.

Event-Aufnahme durch die Einblendung eines Overlays stabilisieren (User)

SUT Skript Name: OPT_INTERACTION_OVERLAY_MODE
Mögliche Werte: VAL_INTERACTION_OVERLAY_MODE_NONE,
VAL_INTERACTION_OVERLAY_MODE_MUTATION,
VAL_INTERACTION_OVERLAY_MODE_TRACKER
Bei der Aufnahme von Web-Events werden von QF-Test Komponentenknoten erzeugt. Dazu ermittelt QF-Test - mit Unterstützung der Resolver - die aktuellen Werte für die Komponenteninformationen. Je nach Komplexität der Webseite und der bereitgestellten Resolver kann dies ein paar Augenblicke dauern. Mit dieser Option kann QF-Test ein Overlay einblenden, während es damit beschäftigt ist, die Webseite und ihre Komponenten für die Wiedererkennung zu analysieren. Dies kann den Verarbeitungsprozess für den Anwender transparenter machen.

Behandlung von Fehlern in einer Web-Anwendung (System)

SUT Skript Name: OPT_WEB_JAVASCRIPT_ERROR_LEVEL
Mögliche Werte: VAL_WEB_JAVASCRIPT_LEVEL_WARNING,
VAL_WEB_JAVASCRIPT_LEVEL_ERROR
Dynamisches HTML wird mit Hilfe großer Mengen von JavaScript Code implementiert, der im Browser ausgeführt wird. Tritt in einem solchen Skript ein Fehler auf, wird dieser vom Browser entweder einfach ignoriert, oder es wird ein Fehlerdialog mit Details dazu angezeigt, je nach den persönlichen Einstellungen des Anwenders. Viele dieser Fehler sind harmlos, andere können schwerwiegend sein. QF-Test fängt diese Fehler ab und gibt die Meldung als Fehler oder Warnung im Protokoll aus. Diese Gruppe von Optionen legt die Fehlerstufe der Meldung fest und wie viele Meldungen maximal ausgegeben werden.

Die möglichen Werte für die Option ”Fehlerstufe” sind ”Fehler” und ”Warnung”. Wir raten zur Einstellung ”Fehler” und dafür zu sorgen, dass derartige Probleme umgehend an die Entwicklung gemeldet werden, da sie einen Fehler in der von Ihnen getesteten Anwendung darstellen können. Bekannte Meldungen, die von der Entwicklung nicht beseitigt werden, können über die Option Folgende Fehler ignorieren[571] von der Prüfung ausgenommen werden.

Maximale Anzahl von Fehlermeldungen pro SUT-Client (System)

SUT Skript Name: OPT_WEB_JAVASCRIPT_MAX_ERRORS
Wenn eine Webseite fehlerhaften Code enthält kann es sein, dass daraus eine große Menge von Fehlermeldungen resultiert, was zu einem Einbruch der

Performance führen kann. Außerdem bringen diese Fehlermeldungen nach der ersten Handvoll keine neue Information mehr. Mittels der Option ”Maximale Anzahl von Fehlermeldungen pro SUT Client” kann die Zahl derartiger Fehlermeldungen begrenzt werden.

Folgende Fehler ignorieren (System)

3.5+

SUT Skript Name: OPT_WEB_JAVASCRIPT_ERROR_FILTERS
Falls ein JavaScript Fehler nicht behoben werden kann, z.B. weil der Code von Dritten stammt, ist es sinnvoll, bekannte Fehler gezielt zu ignorieren und unerwartete Fehler trotzdem zu melden. Wenn der Browser einen Fehler meldet, durchsucht QF-Test die Fehlermeldung nach den in dieser Option angegebenen regulären Ausdrücken. Im Fall einer Übereinstimmung wird der Fehler ignoriert. Sind keine Ausnahmen definiert oder gibt es keinen Treffer, wird der Fehler entsprechend den vorhergehenden Optionen protokolliert.

41.7.1 HTTP-Requests

Diese Optionen beeinflußen den Umgang mit HTTP-Requests.

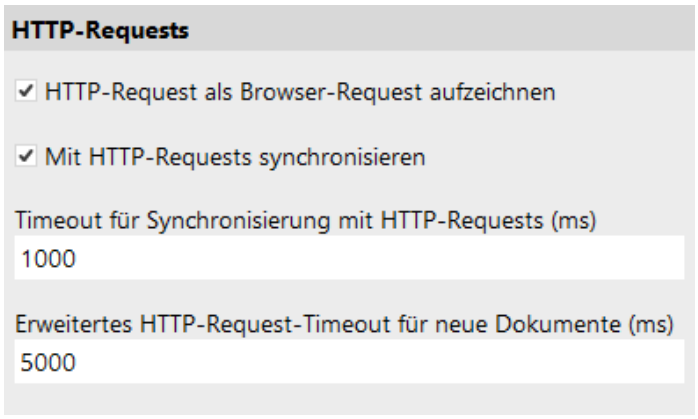


Abbildung 41.34: Optionen für HTTP-Requests

HTTP-Request als Browser Request aufzeichnen (System)

4.1+

SUT Skript Name: OPT_WEB_RECORD_CLIENT_REQUEST_STEP
Beim Aufnehmen von HTTP-Requests wird standardmäßig ein Browser-HTTP-Request(915) erzeugt. Dieser wird direkt im Browser abgespielt, so dass der Response entsprechend sichtbar wird und der Testablauf im Browser

fortgeführt werden kann. Ist diese Option deaktiviert wird stattdessen ein Server-HTTP-Request[910] aufgezeichnet. Die Wiedergabe erfolgt dann direkt aus QF-Test heraus und hat keine Auswirkung auf den Browser. Der Response steht nur in QF-Test zur Verfügung.

Mit HTTP-Requests synchronisieren (System)

4.1+

SUT Skript Name: OPT_WEB_TRACK_HTTP_REQUESTS

Hinweis

Die Verfolgung von HTTP Anfragen ist nur für Browser im QF-Driver oder CDP-Driver-Verbindungsmodus möglich. Im WebDriver-Verbindungsmodus und bei in Java eingebetteten Browsern wie WebView findet dieser Ansatz keine Anwendung.

Da bei JavaScript-basierten Web-Anwendungen vieles asynchron abläuft, ist eine der größten Herausforderungen bei der Automatisierung solcher Anwendungen das Timing. QF-Test nutzt verschiedene Techniken zur Synchronisierung mit dem SUT und diese Option steuert eine davon. Ist sie aktiviert, verfolgt QF-Test alle HTTP Anfragen vom Browser an den Server. Vor und nach der Wiedergabe von Events wartet QF-Test zunächst darauf, dass keine Anfragen mehr offen sind. Die folgenden beiden Optionen Timeout für Synchronisierung mit HTTP-Requests (ms)[572] und Erweitertes HTTP-Request-Timeout für neue Dokumente (ms)[572] dienen zur Feinjustierung dieser Funktion.

Timeout für Synchronisierung mit HTTP-Requests (ms) (System)

4.1+

SUT Skript Name: OPT_WEB_HTTP_REQUEST_TIMEOUT

Bei der Synchronisierung mit dem SUT durch Überwachen von HTTP Anfragen - wie für die Option Mit HTTP-Requests synchronisieren[572] beschrieben - kann QF-Test nicht beliebig lange auf ausstehende Anfragen warten, das würde die Performance der Tests zu sehr beeinträchtigen. Diese Option legt die maximale Zeitspanne fest, die QF-Test in normalen Situationen auf ausstehende Anfragen wartet. Die folgende Option Erweitertes HTTP-Request-Timeout für neue Dokumente (ms)[572] greift hingegen direkt nach dem Laden einer neuen Webseite.

Erweitertes HTTP-Request-Timeout für neue Dokumente (ms) (System)

4.1+

SUT Skript Name: OPT_WEB_HTTP_REQUEST_TIMEOUT_DC

Bei der Synchronisierung mit dem SUT durch Überwachen von HTTP Anfragen - wie für die Option Mit HTTP-Requests synchronisieren[572] beschrieben - kann QF-Test nicht beliebig lange auf ausstehende Anfragen warten, das würde die Performance der Tests zu sehr beeinträchtigen. Direkt nach dem Laden einer neuen Webseite schicken JavaScript-Anwendungen oft viele Anfragen und es

kann eine Weile dauern, bis die endgültige Oberfläche mittels JavaScript aufgebaut ist. Diese Option legt die maximale Zeitspanne fest, die QF-Test in dieser Situation auf ausstehende Anfragen wartet. Die vorhergehende Option Timeout für Synchronisierung mit HTTP-Requests (ms)[572] greift in allen anderen Fällen.

41.7.2 Rückwärtskompatibilität

Diese Optionen stellen älteres Verhalten von QF-Test wieder her, welches im Laufe des Entwicklungszykluses so stark verändert wurde, dass die Rückwärtskompatibilität nicht mehr garantiert werden konnte.

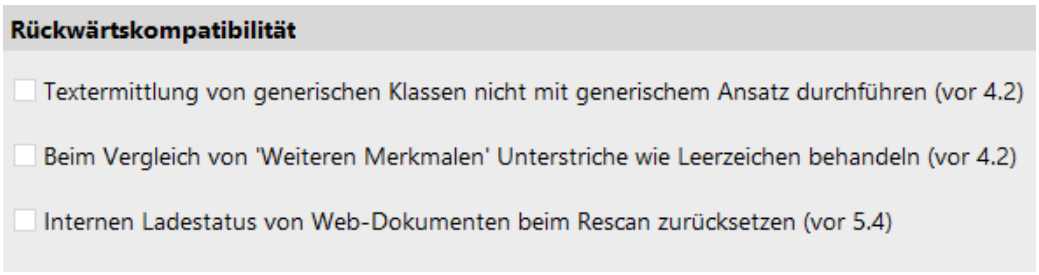


Abbildung 41.35: Optionen für Web Rückwärtskompatibilität

Textermittlung von generischen Klassen nicht mit generischem Ansatz durchführen (vor 4.2) (System)

4.2+

SUT Skript Name: OPT_WEB_TEXT_DONT_TRAVERSE_ALL_NODES_FOR_GENERICS
Vor 4.2.0 wurde in manchen Fällen von Check Text[806] oder Text auslesen[839] Knoten zu viel bzw. zu wenig Text für Komponenten mit generischen Klassen ermittelt. Hiervon waren meistens `SELECT` Komponenten wie auch `TableCell` Komponenten, die Textfelder beinhaltet haben, betroffen. Jetzt werden alle Kindkomponenten mit einer generischen Klassen für die Textermittlung herangezogen.

Beim Vergleich von Weitere Merkmale Unterstriche wie Leerzeichen behandeln (vor 4.2) (System)

4.2+

SUT Skript Name: OPT_WEB_TREAT_UNDERSCORES_AS_BLANKS_IN_EF
In älteren QF-Test Versionen wurden beim Vergleich der Weitere Merkmale alle Unterstriche automatisch wie Leerzeichen behandelt. Dies konnte bei absichtlichen Suchen nach Unterstrichen zu Problemen führen.

Internen Ladestatus von Web-Dokumenten beim Rescan zurücksetzen (vor 5.4) (System)

5.4+

SUT Skript Name: OPT_WEB_RESET_DOCUMENT_KNOWN_STATE_DURING_RESCAN
Vor Version 5.4 enthielt QF-Test einen internen Fehler, bei dem der Ladezustand eines Dokuments fälschlicherweise zurückgesetzt werden konnte. Außerdem wurden Dokumente innerhalb von Frames diesbezüglich nicht korrekt behandelt. Beides konnte dazu führen, dass ein Warten auf Laden des Dokuments(880) nur aufgrund der Existenz eines Dokuments erfolgreich war, ohne dass dieses tatsächlich neu geladen wurde.

Ab QF-Test Version 5.4 wird das Neu-Laden wieder präziser geprüft. Dadurch kann es zu neuen Fehlern Tests kommen, in denen zu viele oder falsch platzierte Warten auf Laden des Dokuments(880) Knoten enthalten sind. Wenn eine Korrektur der Tests zu umfangreich erscheint, kann das vorherige Verhalten durch Aktivieren dieser Option wieder hergestellt werden.

41.8 SWT-Optionen

SWT

Die folgenden Optionen sind speziell für das Testen von SWT-Anwendungen von Bedeutung.

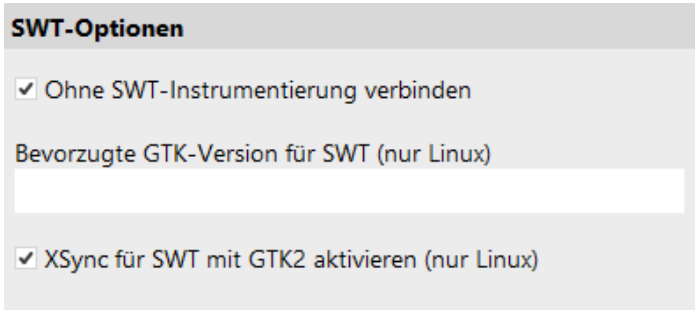


Abbildung 41.36: SWT-Optionen

Ohne SWT-Instrumentierung verbinden (System)

4.5+

Server Skript Name: OPT_PLAY_SWT_VIA_AGENT
Ist diese Option aktiviert, müssen SWT-Anwendungen (bis auf ältere SWT-Versionen unter Linux) nicht instrumentiert werden. Detaillierte technische Informationen hierzu finden Sie in Abschnitt 47.2(1012).

Bevorzugte GTK-Version für SWT (nur Linux) (System)

4.5+

Server Skript Name: OPT_PLAY_SWT_PREFERRED_GTK_VERSION
Unter Linux können Eclipse/SWT-Anwendungen der SWT-Versionen 4.3 bis 4.9 wahlweise mit GTK2 oder GTK3 ausgeführt werden. Diese Option kann auf ”2” oder ”3” gesetzt werden, um eine spezifische GTK-Version zu erzwingen oder leer gelassen werden, um den zur jeweiligen SWT-Version passenden Standard zu verwenden.

XSync für SWT mit GTK2 aktivieren (nur Linux) (System)

4.5+

Server Skript Name: OPT_PLAY_SWT_GTK2_XSYNC
SWT-Anwendungen mit GTK2 werden auf neueren Linux Systemen zunehmend instabil und können unter starker Last, welche bei Ausführung mit QF-Test in maximaler Geschwindigkeit nicht ungewöhnlich ist, abstürzen. Solche Fehler können durch Aktivierung von XSync behoben werden, einer X11 spezifischen Option mit der X11 Events synchron verarbeitet werden. Diese kann allerdings die Geschwindigkeit beeinträchtigen. Wenn Sie Ihre SWT-Anwendung mit GTK2 betreiben müssen und diese unter QF-Test langsam erscheint, deaktivieren Sie probeweise diese Option um zu sehen, ob die Tests damit schneller werden, ohne dass das SUT gelegentlich abstürzt.

41.9 UI-Inspektor-Optionen

Die folgenden Optionen sind speziell für den UI-Inspektor.

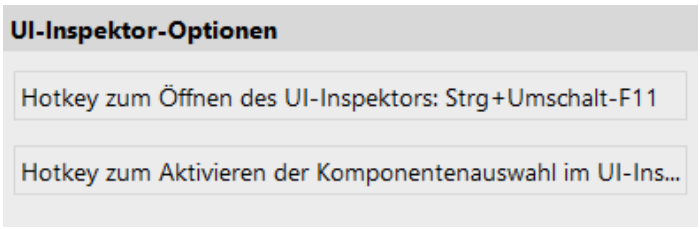


Abbildung 41.37: UI-Inspektor-Optionen

Hotkey zum Öffnen des UI-Inspektors (System)

SUT Skript Name: OPT_INSPECTOR_HOTKEY
Hiermit legen Sie eine Taste bzw. Tastenkombination fest, mit der Sie den

UI-Inspektor direkt vom SUT aus öffnen können. Um den Wert zu ändern klicken Sie auf den aktuellen Eintrag (hierbei handelt um ein interaktives Feld) und drücken nun die gewünschte Taste bzw. Tastenkombination. Um das Feld zu verlassen, drücken Sie entweder die Tab-Taste oder setzen Sie den Fokus mit der Maus auf ein anderes Feld. Die Standard-Tastenkombination ist Strg-Shift-F11 für Window/Linux bzw. ⇧-^-F11 für Mac. Detaillierte Informationen zum UI-Inspektor finden Sie in Abschnitt 5.12.2[108].

Hotkey zum Aktivieren der Komponentenauswahl im UI-Inspektor (System)

SUT Skript Name: OPT_INSPECTOR_MODE_HOTKEY
Hiermit legen Sie eine Taste bzw. Tastenkombination fest, mit der Sie direkt vom SUT aus die Komponentenauswahl im UI-Inspektor aktivieren können. (Der UI-Inspektor wird auch geöffnet, falls nicht bereits geschehen.) Um den Wert zu ändern klicken Sie auf den aktuellen Eintrag (hierbei handelt um ein interaktives Feld) und drücken nun die gewünschte Taste bzw. Tastenkombination. Um das Feld zu verlassen, drücken Sie entweder die Tab-Taste oder setzen Sie den Fokus mit der Maus auf ein anderes Feld. Die Standard-Tastenkombination ist Strg-Shift-F12 für Window/Linux bzw. ⇧-^-F12 für Mac. Detaillierte Informationen zum Inspektor finden Sie in Abschnitt 5.12.2[108].

41.10 Debugger-Optionen

Diese Optionen haben Einfluss auf das Verhalten des Debuggers.

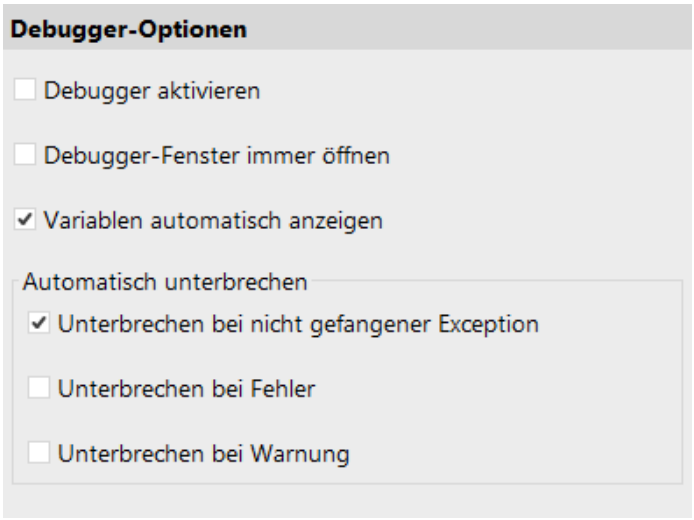


Abbildung 41.38: Debugger-Optionen

Debugger aktivieren (User)

Normalerweise ist der Debugger inaktiv so lange diese Option, die auch über den Menüeintrag Debugger→Debugger aktivieren zugänglich ist, ausgeschaltet ist.

Der Debugger wird automatisch aktiv, wenn der Test auf einen Breakpoint trifft oder explizit durch drücken des ”Pause” Buttons unterbrochen wird. Wird ein Testlauf mittels ”Einzelschritt ausführen” oder ”Gesamten Knoten ausführen” gestartet, ist der Debugger für die Dauer dieses Testlaufs ebenfalls aktiv.

Debugger-Fenster immer öffnen (User)

Wenn ein Testlauf unterbrochen und der Debugger aufgerufen wird, kann QF-Test optional ein eigenes Fenster für den Debugger öffnen. Diese Option legt fest, ob für das Debuggen dieses separate Fenster oder das normalen Testsuite-Fenster genutzt werden soll.

Variablen automatisch anzeigen (User)

Wenn ein Testlauf unterbrochen und der Debugger aufgerufen wird, kann QF-Test die aktuell gebundenen Variablen im Workbench-Fenster darstellen. Ist diese Option gesetzt, werden die Variablen-Ansicht immer automatisch geöffnet, wenn ein Testlauf zum ersten mal den Debugger betritt. Alternativ können die Variablen im Debugger-Fenster oder über das Menü Debugger→Variablen anzeigen im Workbench-Fenster angezeigt werden.

Automatisch unterbrechen (User)

Diese Optionen legen fest, in welchen Situationen die Ausführung eines Tests unterbrochen und der Debugger aufgerufen wird.

Unterbrechen bei nicht gefangener Exception
: Der Debugger unterbricht die Ausführung wenn eine Exception geworfen wird, die von keinem Catch(707) Knoten behandelt wird.

Unterbrechen bei Fehler
: Der Debugger unterbricht die Ausführung wenn ein Fehler auftritt.

Unterbrechen bei Warnung
: Der Debugger unterbricht die Ausführung wenn eine Warnung auftritt.

41.11 Protokoll

Die folgenden Optionen haben Einfluss darauf, welche Informationen in einem Protokoll aufgezeichnet werden, wann dieses angezeigt und wie es gespeichert wird.

41.11.1 Allgemeine Protokoll-Optionen

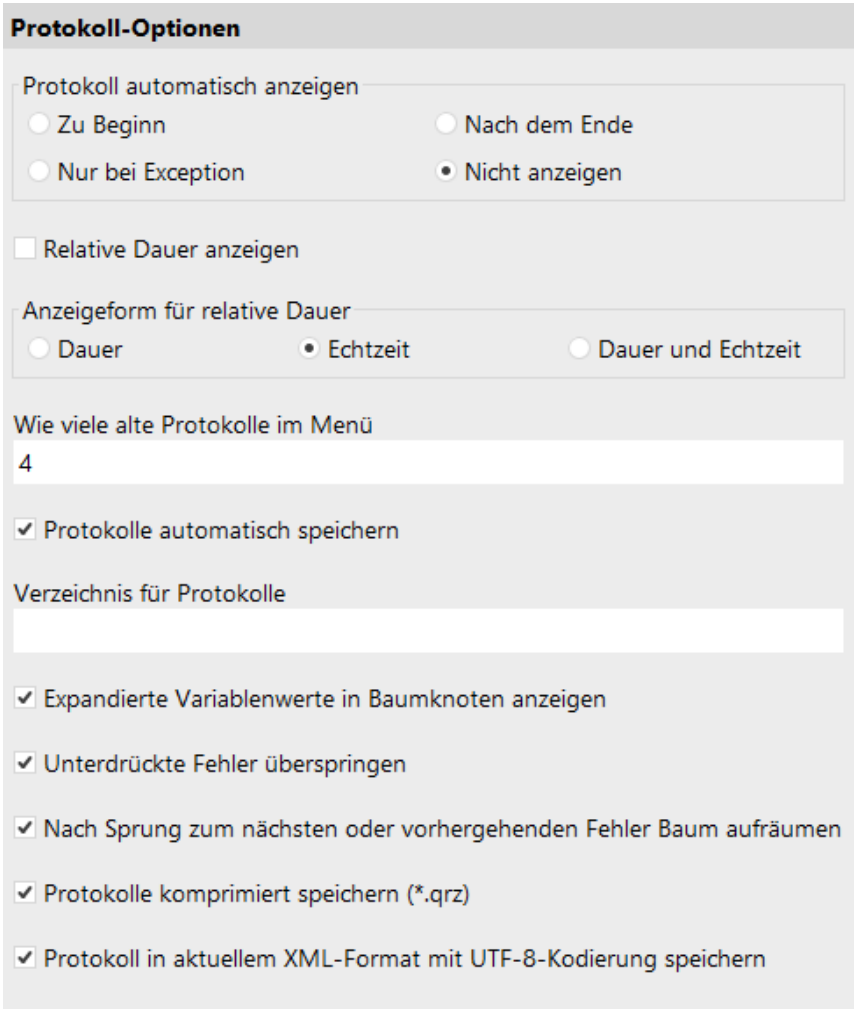


Abbildung 41.39: Protokoll-Optionen

Protokoll anzeigen (User)

Hier lässt sich einstellen, ob und unter welchen Bedingungen das Protokoll, das

für jeden Testlauf erzeugt wird, angezeigt werden soll. Die Protokolle der letzten Testläufe sind in jedem Fall über das Menü Wiedergabe abrufbar. Die Tastenkombination Strg-L ruft das jeweils letzte Protokoll ab.

Zu Beginn
Legt fest, dass das Protokoll bereits beim Start eines Testlaufs angezeigt wird.

Nach dem Ende
Bei dieser Einstellung wird das Protokoll angezeigt, nachdem ein Testlauf beendet ist.

Nur bei Exception
In diesem Fall wird das Protokoll nur angezeigt, wenn ein Testlauf durch eine Exception abgebrochen wird.

Nicht anzeigen
Hier wird das Protokoll nicht automatisch angezeigt, ist aber dennoch über das Wiedergabe Menü oder mittels Strg-L. zugänglich.

Relative Dauer anzeigen (User)

6.0+

Server Skript Name: OPT_LOG_DURATION_INDICATORS
Um das Laufzeitverhalten eines Tests zu analysieren hilft ein schneller Blick darauf, in welchen Zweigen die meiste Zeit verbracht wird. Zu diesem Zweck kann über diese Option die Anzeige der relativen Dauer im Protokoll aktiviert werden. Die Option ist auch direkt über das Anzeige Menü im Protokoll zugänglich.

Die Länge der angezeigten Balken entspricht dem relativen Anteil der Dauer des Knotens an der Dauer seines Parentknotens.

Anzeigeform für relative Dauer (User)

6.0+

Server Skript Name: OPT_LOG_DURATION_INDICATORS_KIND
Mögliche Werte: VAL_LOG_DURATION_INDICATORS_KIND_DURATION,
VAL_LOG_DURATION_INDICATORS_KIND_REALTIME,
VAL_LOG_DURATION_INDICATORS_KIND_BOTH
Über diese Option wird festgelegt, ob sich die Anzeige der relativen Dauer im Protokoll auf die Dauer, die Echtzeit oder beides bezieht.

Hinweis

Unterschiede zwischen den Werten ”Dauer” und ”Echtzeit” entstehen zum Beispiel durch explizite Verzögerungen in Knoten mittels ’Verzögerung vorher/nachher’ oder bei Unterbrechungen durch den Benutzer.

Wie viele alte Protokolle im Menü (User)

Hier können Sie einstellen, wie viele Protokolle im Wiedergabe Menü

aufgehoben werden sollen. Werden mehr Testläufe durchgeführt als Protokolle aufgehoben werden sollen, werden die jeweils ältesten Protokolle verworfen. Sofern sie nicht gespeichert wurden, sind sie damit unwiederbringlich verloren.

Protokolle automatisch speichern (User)

3.0+

Um übermäßigen Speicherverbrauch zu vermeiden und außerdem die letzten Protokolle zwischen verschiedenen QF-Test Sessions persistent zu machen, werden die aktuellen Protokolle im Wiedergabe Menü automatisch im benutzerspezifischen Konfigurationsverzeichnis(12) gespeichert, bzw. in dem Verzeichnis, das in der Option Verzeichnis für Protokolle(580) spezifiziert wurde. Der Dateiname für ein Protokoll wird aus einem Zeitstempel gebildet. QF-Test sperrt diese Dateien, um Konflikte und versehentliches Löschen bei mehreren parallelen QF-Test Sessions zu vermeiden und hält das benutzerspezifische Konfigurationsverzeichnis(12) durch regelmäßiges Löschen der unbenutzten Protokolle sauber. Es sollte also keinen Grund geben, diese Funktion auszuschalten, aber wenn Sie darauf bestehen, können Sie es mittels dieser Option tun.

Verzeichnis für Protokolle (User)

4.0+

Standardmäßig werden bei interaktiver Ausführung von QF-Test Protokolle im benutzerspezifischen Konfigurationsverzeichnis(12). Mit dieser Option können Sie ein alternatives Zielverzeichnis angeben.

Hinweis

Diese Option wird von QF-Test bei jedem Start eines Tests ausgewertet. Zu diesem Zeitpunkt sind globale Variablen und Variablen der Testsuite bereits definiert und im Gegensatz zu andere Option darf hier die QF-Test Variablensyntax verwendet werden, einschließlich spezieller Variablen wie ${env:HOME} um Umgebungsvariablen auszuwerten oder sogar ${qftest:suite.dir} um das Protokoll neben der Testsuite abzulegen. Falls das Verzeichnis, wie im letzten Fall, dynamisch ist, kann QF-Test eventuell alte Protokolle nicht regelmäßig aufräumen. Fehler bei der Expansion von Variablen werden ohne Meldung ignoriert und das benutzerspezifische Konfigurationsverzeichnis(12) stattdessen verwendet.

Expandierte Variablenwerte in Baumknoten anzeigen (User)

3.1+

Die Knoten der Baumansicht eines Protokolls können entweder mit expandierten Variablen - mit Werten vom Zeitpunkt der Ausführung - dargestellt werden, oder mit den Variablen selbst. Beide Ansichten sind nützlich, daher können Sie mittels dieser Option, oder einfacher über den Menüeintrag Ansicht→Knoten expandiert darstellen, zwischen ihnen wechseln.

Rückgabewerte von Prozeduren anzeigen (User)

7.0+

Ist diese Option aktiv, werden Rückgabewerte von Prozedur(672) Knoten im Baum neben dem entsprechenden Prozeduraufruf(675) Knoten angezeigt. In einem Protokoll kann der Wert dieser Option auch einfach über den Menüeintrag Ansicht→Rückgabewerte von Prozeduren anzeigen.

Unterdrückte Fehler überspringen (User)

Wie die vorhergehende, bezieht sich auch diese Option auf die Suche nach Fehlern im Protokoll. Ist sie aktiviert, werden nur solche Warnungen, Fehler oder Exceptions gefunden, die im Protokoll nach oben durchgereicht wurden. Exceptions, die durch ein Try(704)/Catch(707) Konstrukt abgefangen oder Meldungen, die durch das Maximaler Fehler(619) Attribut unterdrückt wurden, werden übersprungen.

Diese Option kann sowohl über den Optionen-Dialog, als auch direkt über den Menüeintrag Bearbeiten→Unterdrückte Fehler überspringen verändert werden.

Nach Sprung zum nächsten oder vorhergehenden Fehler Baum aufräumen (User)

3.0+

Werden mehrfach Fehler in einem Protokoll angesprungen, kann dieses leicht durch die vielen expandierten Knoten unübersichtlich werden. Ist diese Option aktiviert, räumt QF-Test automatisch nach jedem Sprung zu einem Fehler den Baum auf, so dass nur die Parent-Knoten des aktuellen Fehlers expandiert sind.

Hinweis

Beim Bearbeiten von geteilten Protokollen werden Teil-Protokolle, die einen Fehler enthalten, so lange im Speicher gehalten wie ihre Knoten expandiert sind. Durch Aktivieren dieser Option können Sie sicherstellen, dass Teil-Protokolle so bald wie möglich freigegeben werden und der Speicherverbrauch auch beim Betrachten der Fehler eines sehr großen Protokolls im Rahmen bleibt.

Protokolle komprimiert speichern (*.qrz) (System)

Server Skript Name: OPT_LOG_SAVE_COMPRESSED
Protokolle können als normale oder als komprimierte XML-Datei gespeichert werden. Für große Protokolle ohne Bildschirmabbilder kann der Kompressionsfaktor durchaus 10 und mehr betragen, so dass es ratsam ist, Kompression zu verwenden. Das einzige Argument dagegen ist, wenn Sie die XML-Protokolle nachträglich transformieren wollen. Da QF-Test aber den Standard gzip Algorithmus zur Kompression verwendet, können komprimierte und normale Protokolle jederzeit mittels `gzip` ineinander umgewandelt werden.

Wenn Sie ein Protokoll interaktiv speichern, können Sie jederzeit über die Kompression entscheiden indem Sie im Dialog den entsprechenden Filter auswählen oder der Datei die Entsprechende Endung `.qrl` oder `.qrz` geben.

Im Batch Modus werden Protokolle normalerweise komprimiert geschrieben. Um ein unkomprimiertes Protokoll zu erstellen geben Sie einfach mit dem Kommandozeilenargument `-runlog [<Datei>]`[989] eine Datei mit der Endung `.qrl` an.

Protokoll in aktuellem XML-Format mit UTF-8-Kodierung speichern (System)

7.0+

Server Skript Name: OPT_XML_LOG_NEW
Ab QF-Test Version 7.0 werden Protokolle im aktuellen XML-Format mit UTF-8-Kodierung, ohne Einrückung und beliebiger Zeilenlänge gespeichert. Falls Sie für die Auswertung in externen Programmen auf das alte Format für Protokolle mit ISO-8859-1 Kodierung, Einrückung von 2 Zeichen und Zeilenlänge 78 angewiesen sind, können Sie es durch Deaktivieren dieser Option wieder herstellen.

41.11.2 Optionen zur Aufteilung von Protokollen

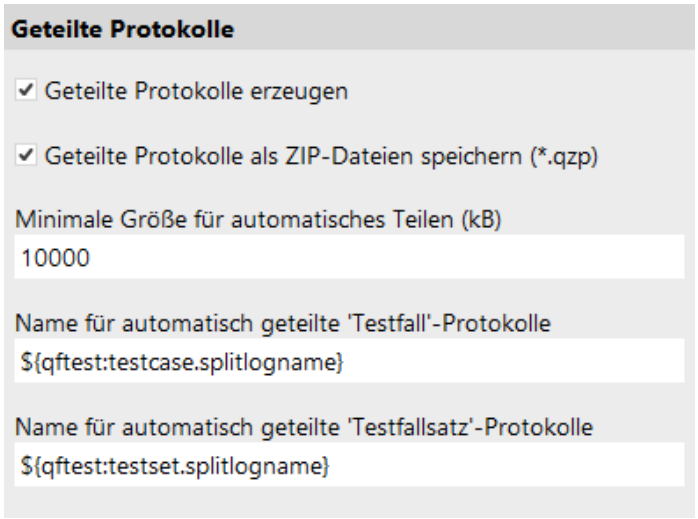


Abbildung 41.40: Optionen zur Aufteilung von Protokollen

Geteilte Protokolle erzeugen (User)

3.0+

Server Skript Name: OPT_LOG_SAVE_SPLIT

Ein Protokoll kann durch setzen des Attributs Name für separates Protokoll(648) eines Datentreiber(646) Knotens oder eines der verschiedenen Test Knoten in mehrere Teile zerlegt werden. Sie können diese Funktion deaktivieren, indem Sie diese Option ausschalten. So müssen Sie keine Name für separates Protokoll Attribute ändern, wenn Sie zwischendurch ein komplettes Protokoll erstellen wollen.

Weitere Informationen zu geteilten Protokollen finden Sie in Abschnitt 7.1.6(143).

Geteilte Protokolle als ZIP-Dateien speichern (*.qzp) (User)

3.0+

Server Skript Name: OPT_LOG_SAVE_SPLIT_ZIP
Geteilte Protokolle können entweder als einzelne ZIP-Datei mit der Endung `.qzp` gespeichert werden, die das Hauptprotokoll und alle davon abgeteilten Protokolle enthält, oder als normales `.qrl` oder `.qrz` Protokoll. Letzteres wird um ein Verzeichnis mit dem selben Basisnamen und dem Anhang `_logs` ergänzt, welches die abgeteilten Protokolle aufnimmt, z.B. die Datei `runlog.qrz` plus das Verzeichnis `runlog_logs`. Diese Option legt das Format fest, mit dem Protokolle automatisch im interaktiven Modus geschrieben werden. Wird die Option Protokolle automatisch speichern(580) ausgeschaltet, ist diese Option ohne Bedeutung.

Weitere Informationen zu geteilten Protokollen finden Sie in Abschnitt 7.1.6(143).

Minimale Größe für automatisches Teilen (kB) (System)

3.4+

Server Skript Name: OPT_LOG_AUTO_SPLIT_SIZE

Hinweis

Diese Option findet nur für Testfall und Testfallsatz Knoten Anwendung. An anderen Stellen werden Protokolle nur bei expliziter Angabe von Name für separates Protokoll geteilt.

Geteilte Protokolle sind der einzig sichere Weg um zu vermeiden, dass bei sehr lang laufenden Tests oder bei vielen Bildschirmabbildern oder großen Ausgaben vom SUT der Speicher ausgeht. Zudem ist die Transformation in Reports bei geteilten Protokollen effizienter. Das explizite Setzen von Name für separates Protokoll Attributen erfordert allerdings eine Kenntnis der Thematik und Entscheidungen, wo ein Protokoll am besten aufgeteilt wird oder führt zu ermüdender Tipparbeit beim Versuch einer feinen Unterteilung.

Als Kompromiss berechnet QF-Test sehr grob die Größe des Protokolls bei der Ausführung und bezieht dabei die Größe von Bildschirmabbildern und Programmausgaben mit ein. Immer wenn bei der Testausführung ein Testfall oder Testfallsatz abgeschlossen ist und die ungefähre Größe des zum Knoten gehörigen Protokolls den in dieser Option angegebenen Schwellwert überschreitet, wird das Teil-Protokoll abgetrennt und separat gespeichert. Ein Wert von 0 verhindert das automatische Teilen.

Weitere Informationen zu geteilten Protokollen finden Sie in Abschnitt 7.1.6[143].

Name für automatisch geteilte 'Testfall'-Protokolle (System)

3.4+

Server Skript Name: OPT_LOG_AUTO_SPLIT_TESTCASE_NAME
Diese Option bestimmt den Namen für ein separates Protokoll, das in Abhängigkeit von der vorhergehenden Option nach Ausführung eines Testfalls geschrieben wird. Variablen können ebenso verwendet werden wie die '%...' Platzhalter, die beim Attribut Name für separates Protokoll[602] beschrieben sind.

Die spezielle Variable ${qftest:testcase.splitlogname} ist eine gute Basis. Sie wird zu einem Pfadnamen expandiert, der aus den Namen des Testfall Knotens und eventuellen Testfallsatz Parentknoten als Verzeichnissen besteht.

Weitere Informationen zu geteilten Protokollen finden Sie in Abschnitt 7.1.6[143].

Name für automatisch geteilte 'Testfallsatz'-Protokolle (System)

3.4+

Server Skript Name: OPT_LOG_AUTO_SPLIT_TESTSET_NAME
Diese Option bestimmt den Namen für ein separates Protokoll, das in Abhängigkeit von der Option Minimale Größe für automatisches Teilen (kB)[583] nach Ausführung eines Testfalls geschrieben wird. Variablen können ebenso verwendet werden wie die '%...' Platzhalter, die beim Attribut Name für separates Protokoll[610] beschrieben sind.

Die spezielle Variable ${qftest:testset.splitlogname} ist eine gute Basis. Sie wird zu einem Pfadnamen expandiert, der aus den Namen des Testfallsatz Knotens und eventuellen Testfallsatz Parentknoten als Verzeichnissen besteht.

Weitere Informationen zu geteilten Protokollen finden Sie in Abschnitt 7.1.6[143].

41.11.3 Optionen für den Inhalt von Protokollen

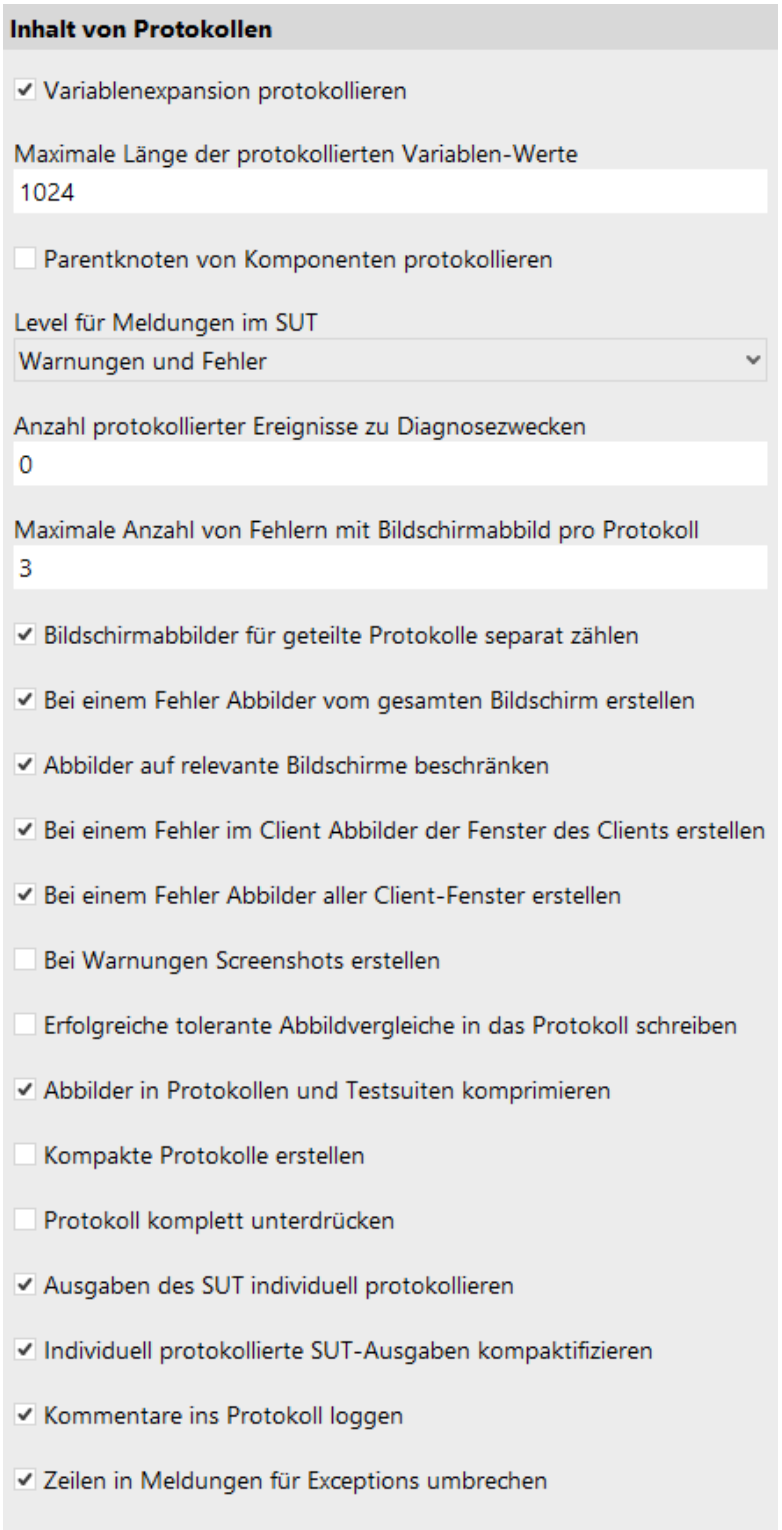


Abbildung 41.41: Optionen für den Inhalt von Protokollen

Variablenexpansion protokollieren (System)

Server Skript Name: OPT_LOG_CONTENT_VARIABLE_EXPANSION
Ist diese Option gesetzt, wird im Protokoll jedes mal, wenn für ein Attribut mit variablem Inhalt die Variablenwerte eingesetzt werden, eine Meldung erstellt.

Maximale Länge der protokollierten Variablen-Werte (System)

Server Skript Name: OPT_LOG_CONTENT_VARIABLE_MAX_VALUE_LENGTH
Diese Option bestimmt, wie viele Zeichen des Inhalts von Variablen beim Setzen bzw. auslesen maximal im Protokoll abgelegt werden. Längere Werte werden auf die angegebene Länge gekürzt. Bei einem negativen Options-Wert wird keine Kürzung vorgenommen, bei '0' werden keine Variablenwerte geloggt.

Parentknoten von Komponenten protokollieren (System)

Server Skript Name: OPT_LOG_CONTENT_PARENT_COMPONENTS
Für jeden Event, Check etc. wird die Zielkomponente im Protokoll gespeichert. Das Setzen dieser Option bewirkt, dass dabei zusätzlich alle direkten und indirekten Parentknoten der Komponente mitgespeichert werden.

Level für Meldungen im SUT (System)

Server Skript Name: OPT_LOG_CONTENT_SUT_LEVEL
Mögliche Werte: VAL_LOG_SUT_LEVEL_MESSAGE,
VAL_LOG_SUT_LEVEL_WARNING,
VAL_LOG_SUT_LEVEL_ERROR
Während des Ablaufs eines Test werden im SUT Meldungen mit verschiedener Bedeutung automatisch generiert, z.B. zur Analyse der Wiedererkennung der Komponenten. Diese Option legt fest, welche davon im Protokoll ankommen: Alle Meldungen, Warnungen und Fehlermeldungen oder nur Fehlermeldungen. Auf Meldungen, die explizit mit dem Nachricht Knoten, `rc.logMessage` oder `qf.logMessage` erstellt werden, hat diese Option keinen Einfluss.

Anzahl protokollierter Ereignisse zu Diagnosezwecken (System)

SUT Skript Name: OPT_LOG_CONTENT_DIAGNOSIS
Beim Abspielen eines Tests zeichnet QF-Test im SUT Events und andere interne Vorgänge auf. Diese Informationen werden normalerweise schnell wieder verworfen, außer im Fall eines Fehlers. Dann werden sie in einen Speziellen Knoten im Protokoll geschrieben. Diese Informationen können zwar auch für Entwickler von Interesse sein, dienen aber in erster Linie zur Fehlerdiagnose, wenn Support von Quality First Software GmbH angefordert wird.

Diese Option legt die Zahl der Ereignisse fest, die QF-Test zwischenspeichert. Ein Wert von 0 schaltet dieses Feature komplett aus. Sie sollten den Wert aber nicht ohne guten Grund auf weniger als ca. 400 reduzieren. Da die Informationen nur im Fehlerfall ausgegeben werden, ist der Aufwand für ihre Erstellung vernachlässigbar.

Maximale Anzahl von Fehlern mit Bildschirmabbild pro Protokoll (System)

Server Skript Name: OPT_LOG_CONTENT_SCREENSHOT_MAX
Die Zahl der Bildschirmabbilder, die QF-Test während eines Testlaufs aufnimmt und im Protokoll speichert, kann mit dieser Option limitiert werden, um den Speicherverbrauch zu begrenzen. Ein Wert von 0 schaltet Bildschirmabbilder komplett aus, ein negativer Wert bedeutet unbegrenzte Bildschirmabbilder.

Bildschirmabbilder für geteilte Protokolle separat zählen (System)

Server Skript Name: OPT_LOG_CONTENT_SCREENSHOT_PER_SPLIT_LOG
Ist diese Option gesetzt, kann jeder Teil eines geteilten Protokolls das oben angegeben Maximum an Bildschirmabbilder enthalten, ohne die Bildschirmabbilder für das Hauptprotokoll zu beeinflussen. Andernfalls gilt das Maximum für alle Teile zusammen.

Weitere Information zu geteilten Protokollen finden Sie in Abschnitt 7.1.6(143).

Bei einem Fehler Abbilder vom gesamten Bildschirm erstellen (System)

Server Skript Name: OPT_LOG_CONTENT_SCREENSHOT_FULLSCREEN
Das Aktivieren dieser Option veranlasst QF-Test ein Abbild des gesamten Bildschirms zu generieren und im Protokoll zu speichern, wenn dies aufgrund einer Exception oder eines Fehlers während des Testlaufs angestoßen wird.

Abbilder auf relevante Bildschirme beschränken (System)

6.0+

Server Skript Name: OPT_LOG_CONTENT_SCREENSHOT_RELEVANT_ONLY
Wenn mehrere Monitore angeschlossen sind, ist es nicht immer sinnvoll, Abbilder von allen Bildschirmen zu nehmen. Insbesondere beim Arbeiten im interaktiven Modus auf dem eigenen Rechner können gelangen sonst leicht private oder vertrauliche Informationen in das Protokoll.

Ist diese Option aktiviert, was standardmäßig der Fall ist, versucht QF-Test die Bildschirme zu identifizieren, auf denen Fenster des SUT oder von QF-Test zu sehen sind und ignoriert den Rest.

Bei einem Fehler im Client Abbilder der Fenster des Clients erstellen (System)

Server Skript Name: OPT_LOG_CONTENT_SCREENSHOT_WINDOW
Ist diese Option gesetzt, generiert QF-Test während der Testausführung Abbilder von allen Fenster des SUT, von dem eine Exception oder ein Fehler ausgelöst wird und speichert diese im Protokoll. Dies funktioniert oft auch in Situationen, wo ein Fenster von einem anderen Fenster überdeckt wird oder für Fälle, in denen ein Abbild des gesamten Bildschirms nicht möglich ist, z.B. wenn der Bildschirm gesperrt ist.

Bei einem Fehler Abbilder aller Client-Fenster erstellen (System)

4.3+

Server Skript Name: OPT_LOG_CONTENT_SCREENSHOT_ALL_WINDOWS
Ist diese Option aktiviert werden Abbilder von allen Fenstern aller verbundener SUTs im Protokoll gespeichert, unabhängig von der Quelle der auslösenden Exception bzw. des Fehlers. Diese Option beeinflusst auch, welche Bildschirme, abhängig von den SUT Fenstern, als relevant betrachtet werden.

Bei Warnungen Screenshots erstellen (System)

6.0+

Server Skript Name: OPT_LOG_CONTENT_WARNING_SCREENSHOTS
Ist diese Option aktiviert, werden auch dann Abbilder erstellt, wenn Warnungen im Protokoll gespeichert werden. Andernfalls geschieht dies nur bei Fehlern und Exceptions.

Erfolgreiche tolerante Abbildvergleiche in das Protokoll schreiben (System)

3.4+

Server Skript Name: OPT_LOG_CONTENT_SCREENSHOT_ADVANCED_ALWAYS
Ist diese Option gesetzt, werden auch bei erfolgreichem Abbildvergleich bei Verwendung von toleranten Imagevergleichen die Ergebnisse des Algorithmus inklusive der transformierten Bilder in das Protokoll geschrieben. Andernfalls werden diese Details nur bei fehlgeschlagenem Vergleich gespeichert.

Diese Option kann den Speicherverbrauch des Protokolls drastisch erhöhen, daher sollten Sie sie auf jeden Fall mit kompakten und/oder geteilten Protokollen kombinieren.

Abbilder in Protokollen und Testsuiten komprimieren (System)

4.5+

Server Skript Name: OPT_LOG_CONTENT_COMPRESS_SCREENSHOTS
Ist diese Option gesetzt, so werden neu erzeugte Abbilder von Komponenten und Bildschirmabbilder in Protokollen und Run logs verlustfrei komprimiert gespeichert.

Diese Option kann den Speicherverbrauch des Protokolls und von Test suiten auf dem Datenträger und im Hauptspeicher signifikant senken. Allerdings benötigt das Komprimieren und Dekomprimieren der Grafiken etwas Rechenzeit, so dass die Option bei sehr zeitkritischen Ausführungen deaktiviert werden kann.

Kompakte Protokolle erstellen (System)

Server Skript Name: OPT_LOG_CONTENT_COMPACT
Ist diese Option aktiviert, verwirft QF-Test alle Knoten eines Protokolls, die weder für die Fehlerdiagnose, noch für den XML/HTML-Report benötigt werden. Nach einem Fehler oder einer Exception, sowie am Ende eines Testlaufs, bleiben die zuletzt ausgeführten 100 Knoten im Protokoll erhalten, so dass die wichtigsten Informationen erhalten bleiben sollten.

Selbst größere Tests sollten keine Speicherprobleme auslösen, sofern die Option Geteilte Protokolle erzeugen[582] aktiviert ist und wie in Geteilte Protokolle[143] beschrieben verwendet wird. Wenn der Speicherbedarf dennoch zu groß wird, kann diese Option helfen, ihn deutlich zu reduzieren.

Diese Option wird nur ausgewertet, wenn QF-Test im interaktiven Modus ausgeführt wird. Im Batchmodus (vgl. Abschnitt 1.7[13]) wird die Option ignoriert, um versehentlichen Informationsverlust zu vermeiden. Kompakte Protokolle können im Batchmodus über die Kommandozeilenoption `-compact`[979] aktiviert werden.

Protokoll komplett unterdrücken (System)

Server Skript Name: OPT_LOG_CONTENT_SUPPRESS
Für besonders lang laufende Tests oder Demos kann der Speicherverbrauch von Protokollen ein Problem darstellen, welchem nun mit geteilten Protokollen beizukommen ist. Bevor es geteilte Protokolle gab, stellte diese Option die letzte Möglichkeit dar, besonders lang laufende Tests durchzuführen. Nun ist sie nur noch aus Gründen der Rückwärtskompatibilität vorhanden.

Hinweis

Im Batchmodus wird diese Option ignoriert, stattdessen kann dort durch die Angabe von `-nolog`[984] das Protokoll unterdrückt werden.

Ausgaben des SUT individuell protokollieren (System)

3.0+

SUT Skript Name: OPT_LOG_CONTENT_SPLIT_IO
Ist diese Option gesetzt, werden alle Ausgaben eines SUT-Clients auf seine stdout oder stderr Streams auch individuell im Protokoll abgelegt. Für jede Interaktion mit dem SUT sammelt QF-Test dabei den vorher angefallenen Text, sowie den während der Event Synchronisation ausgegebenen Text. Hierdurch können Ausgaben wie z.B. ein Exception StackTrace direkt mit dem Knoten

assoziiert werden, der diese ausgelöst hat. Dies ist nahezu unmöglich, wenn alle Ausgaben nur in einem großen Block vorhanden sind.

Individuell protokollierte SUT-Ausgaben kompaktifizieren (System)

3.0+

Server Skript Name: OPT_LOG_CONTENT_COMPACTIFY_SPLIT_IO
Ausgaben eines SUT-Clients können, wenn sie gehäuft auftreten, signifikante Mengen an Speicher verbrauchen. Ist diese Option aktiviert, werden individuell protokollierte Ausgaben zusammen mit dem auslösenden Knoten aus einem kompakten Protokoll entfernt, sofern dieser Knoten nicht weiter interessant ist. Nähere Informationen zu kompakten Protokollen finden Sie bei der Option Kompakte Protokolle erstellen(590).

Kommentare ins Protokoll loggen (System)

5.0+

Server Skript Name: OPT_LOG_COMMENTS_TO_RUNLOG
Ist diese Option aktiviert, werden Kommentar(851) Knoten in das Protokoll übernommen.

Zeilen in Meldungen für Exceptions umbrechen (System)

5.3+

Server Skript Name: OPT_LOG_WRAP_EXCEPTION_MESSAGE
Meldungen für Exceptions im Protokoll oder in Fehlerdialogen werden an Wortgrenzen umgebrochen, sofern diese Option aktiviert ist.

41.11.4 Optionen für Verweise zwischen Verzeichnissen mit Testsuiten

4.0+

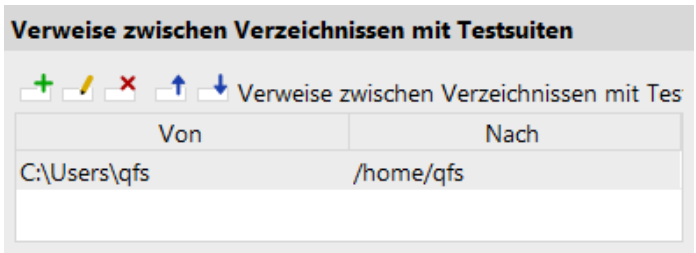


Abbildung 41.42: Optionen für Verweise zwischen Verzeichnissen mit Testsuiten

Verweise zwischen Verzeichnissen mit Testsuiten (System)

Beim Analysieren eines Protokolls ist es oft hilfreich, schnell zwischen dem Protokoll und der zugehörigen Testsuite hin- und herwechseln zu können. Werden automatisierte Tests allerdings auf verschiedenen Systemen ausgeführt, z.B. Windows und Linux, unterscheiden sich die Verzeichnisse, aus denen die Testsuiten für den Test geladen werden. QF-Test kann verschiedene Verzeichnisstrukturen aufeinander abbilden, wenn Sie es dabei mit Hilfe dieser Option unterstützen.

Die 'Von'-Spalte ist ein Muser, das vom Beginn des Pfads der Testsuite bis zum Ende eines Verzeichnisses mit diesem Pfad übereinstimmen muss. Die 'Nach'-Spalte ist die Ersetzung für den übereinstimmenden Teil, die ebenfalls Muster enthalten kann. Bei der Suche nach der referenzierten Testsuite arbeitet QF-Test diese Liste von oben nach unten ab, führt die Ersetzung im Fall einer Übereinstimmung aus und der erste Treffer, der zu einer existierenden Testsuite führt, wird verwendet.

Hinweis

Die hier verwendeten Muster sind keine regulären Ausdrücke, sondern eine einfachere Form, die oft in Entwicklungswerkzeugen eingesetzt wird. Ein '*' steht für 0 oder mehr Zeichen exklusive '/', '**' für 0 oder mehr beliebige Zeichen inklusive '/'. Einige Beispiele:

`**/Testsuiten`
: Alle Verzeichnisse namens `Testsuiten` in beliebiger Tiefe.

`T:/Test/SUT_*`
: Alle Verzeichnisse beginnend mit `SUT_` im Verzeichnis `T:/Test`

41.12 Variablen

Die folgenden Optionen beziehen sich auf das Binden von Variablen:

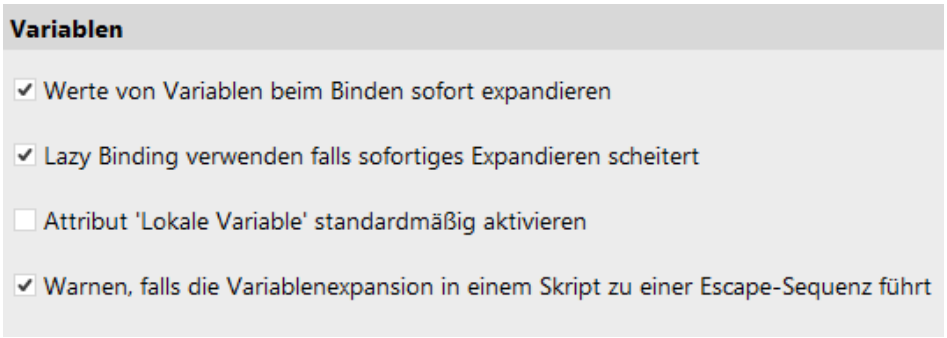


Abbildung 41.43: Variablen Optionen

Werte von Variablen beim Binden sofort expandieren (System)

3.0+

Server Skript Name: OPT_VARIABLE_IMMEDIATE_BINDING
Wird ein Satz von Variablen auf einen Stapel gelegt, können weitere Variablen-Bezüge in deren Werten entweder sofort expandiert werden (”Immediate Binding”), oder unverändert abgelegt und erst bei Bedarf aufgelöst werden (”Lazy Binding”). Dies ist im Detail in Abschnitt 6.9(134) beschrieben.

Für Immediate Binding schalten Sie diese Option ein, für Lazy Binding aus.

Lazy Binding verwenden falls sofortiges Expandieren scheitert (System)

3.0+

Server Skript Name: OPT_VARIABLE_LAZY_FALLBACK
Alte Tests, die auf der Grundlage von Lazy Binding entwickelt wurden, können gegebenenfalls mit dem neuen Standard von Immediate Binding fehlschlagen. Ist diese Option aktiviert, werden Variablen, die auf Grund von Referenzen zu noch undefinierten Variablen nicht sofort gebunden werden können, als Lazy Binding behandelt. Die Warnung, die in diesem Zusammenhang protokolliert wird, können Sie loswerden indem Sie einfach in der Definition der betreffenden Variablen explizit Lazy Binding mittels ’$_’ verwenden. Nähere Informationen hierzu finden Sie in Abschnitt 6.9(134).

Attribut ’Lokale Variable’ standardmäßig aktivieren (System)

5.1+

Viele Knoten haben ein Attribut ’Lokale Variable’, welches bestimmt, ob eine Ergebnisvariable lokal erstellt, oder bei den globalen Variablen abgelegt wird. Ist diese Option aktiviert, wird für neu erstellte Knoten das Attribut ’Lokale Variable’ gesetzt. Weitere Informationen zu lokalen und globalen Variablen finden Sie im Abschnitt Variablenebenen(120).

Warnen, falls direkt expandierte Variablen in Skriptausdrücken Escape-Zeichen enthalten (System)

9.0+

Server Skript Name: OPT_WARN_FOR_ESCAPE_CHARS_IN_EXPANSIONS_FOR_SCRIPTS
Für Skriptausdrücke(189) ist es möglich, Variablen in der Form `$(name)` oder `${group:name}` zu verwenden. Manchmal enthält der Wert der Variable aber unbeabsichtigterweise Escpape-Zeichen: Wenn zum Beispiel die Variable `path` den Wert `C:\Users\test\new` (anstelle von `C:\\Users\\test\\new`) hat und dann in einem Skript mit `”$(path)”==””` verwendet wird, so werden die Sequenzen `\U`, `\t` und `\n` als Escape-Zeichen ausgewertet, was zu überraschenden Fehlern führen kann. Wenn diese Option aktiv ist, dann loggt QF-Test in diesem Fall eine Warnung. Um das Problem zu vermeiden kann man im Skript die Methode `rc.getStr` verwendet werden (vgl. Abschnitt 11.3.3(192)).

Des weiteren können hier verschiedene Arten von Variablen definiert werden. Diese werden im Detail in Kapitel 6(116) beschrieben.

41.13 Nur zur Laufzeit

Manche eher technische Optionen können nicht über die Oberfläche gesetzt werden, insbesondere wenn sie nur für Sonderfälle gedacht sind und im Allgemeinen mehr schaden als nutzen.

Wie schon eingangs des Kapitels erwähnt, können Optionen in Server-Skript(717) oder SUT-Skript(720) Knoten mittels `rc.setOption`(Options.<OPTION_NAME>, <value>) gesetzt werden (vgl. Abschnitt 50.5(1030)). Welche Art von Knoten genutzt werden sollte, ist am Text "Server-Skript Name" bzw. "SUT-Skript Name" in der Dokumentation zur jeweiligen Option zu sehen.

Über QF-Test Agent verbinden (System)

4+

Server Skript Name: OPT_PLAY_CONNECT_VIA_AGENT
In QF-Test Version 4 wurde ein neuer Mechanismus zur Verbindung mit dem SUT eingeführt, der auf Java Agents basiert. Er ist sehr viel mächtiger und flexibler als seine Vorgänger und für die meisten aktuellen SUTs sogar eine zwingende Voraussetzung. Daher sollte diese Option nicht ohne sehr guten Grund ausgeschaltet werden.

Hinweis

Ohne Agent wird ein auf Java 9 oder neuer basierendes SUT gar nicht erst starten. Zudem ist der Agent Voraussetzung für den Zugriff auf JavaFX und eingebettete Browser sowie die volle Funktionalität von live Unittests.

AWT EventQueue instrumentieren (System)

4.1+

Server Skript Name: OPT_PLAY_INSTRUMENT_EVENT_QUEUE

Swing

In älteren Versionen ersetzte QF-Test die AWT System-EventQueue mit seiner eigenen. Der Umgang mit anwendungsspezifischen EventQueues und vielen anderen subtilen Details ist dabei sehr trickreich. Ab QF-Test Version 4.1 wird stattdessen normalerweise die EventQueue Klasse instrumentiert, was weniger Einfluss auf das SUT hat, Spezialfälle besser löst und so gut wie immer die bessere Wahl ist. Sollten trotzdem Verbindungsprobleme auftreten, kann der alte Mechanismus durch deaktivieren dieser Option wieder hergestellt werden.

Hinweis

Das Instrumentieren der AWT EventQueue ist nur mit dem QF-Test Agent möglich.

Kapitel 42

Bestandteile einer Testsuite

42.1 Die Testsuite und ihre Struktur

Eine Testsuite kann aus mehr als 60 verschieden Arten von Knoten bestehen, die in diesem Kapitel einzeln aufgeführt werden. Jeder Knoten hat bestimmte Eigenschaften, darunter seine Attribute, die QF-Test in der Detailansicht des Editors darstellt. Für jedes Attribut wird angegeben, ob bei der Ausführung eine Variablenexpansion durchgeführt wird (vgl. Kapitel 6(116)) und welchen Einschränkungen der Wert des Attributs unterworfen ist.

Weitere Eigenschaften von Knoten sind mögliche Einschränkungen bzgl. zulässiger Parent- und Childknoten, sowie das Verhalten des Knotens bei der Ausführung.

42.1.1 Testsuite

Der Wurzelknoten des Baums repräsentiert die Testsuite selbst. Seine grundlegende Struktur ist fest definiert. Der Wurzelknoten enthält eine beliebige Anzahl von Testfallsätze(606) oder Testfälle(599), gefolgt von den Prozeduren(682), den Extrasequenzen(629) und dem Fenster und Komponenten(942) Knoten. Bei seiner Ausführung werden die Testfallsätze der obersten Ebene der Reihe nach ausgeführt.

Enthalten in: Keiner

Kinder: Eine beliebige Anzahl von Testfallsätze(606) oder Testfallsätze(606), gefolgt von den Prozeduren(682), den Extrasequenzen(629) und dem Fenster und Komponenten(942) Knoten.

Ausführung: Die Testfallsatz Knoten der obersten Ebene werden der Reihe nach ausgeführt.

Attribute:

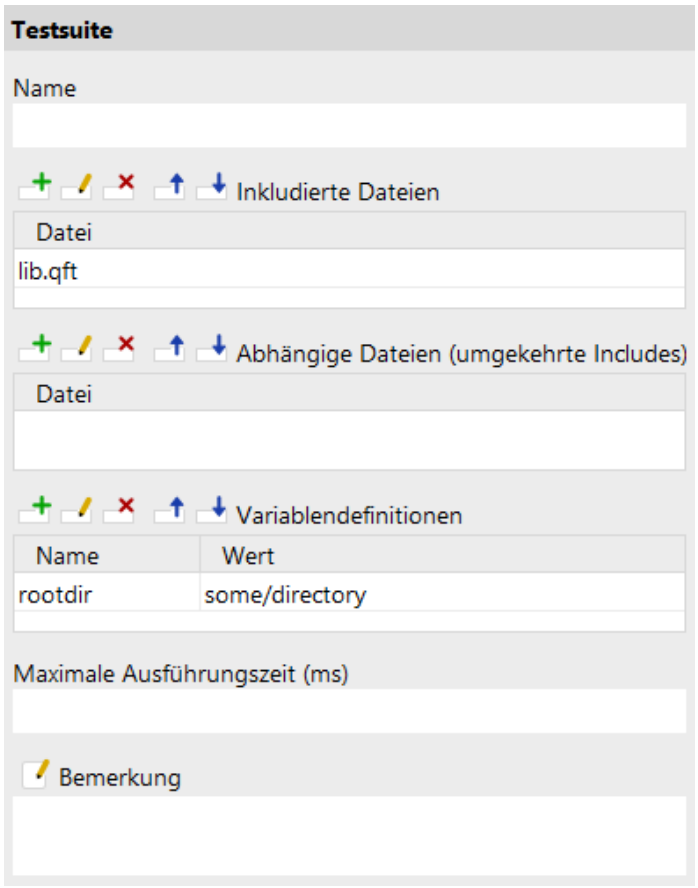


Abbildung 42.1: Testsuite Attribute

Name

Eine Art Kurzkommentar für die Testsuite. Der Name wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Testsuite aussagen.

Variabel: Nein

Einschränkungen: Keine

Inkludierte Dateien

Dies ist eine Liste von Testsuiten, auf die sich diese Testsuite bezieht. Kann eine Referenz auf eine Komponente(930) oder eine Prozedur(672) nicht in der aktuellen Suite aufgelöst werden, werden die hier angegebenen Testsuiten der Reihe nach abgesucht. Beim Aufnehmen von neuen Komponenten sucht QF-Test zunächst die Inkludierte Dateien nach einem entsprechenden Komponente Knoten ab, bevor es einen neuen anlegt.

Dateinamen können absolut oder relativ angegeben werden, wobei relative Dateinamen relativ zu dieser Testsuite oder zu Verzeichnissen des Bibliothekspfades aufgelöst werden (vgl. Option Verzeichnisse mit Testsuite-Bibliotheken[503]).

Wenn Sie an diesem Attribut Änderungen vornehmen bietet QF-Test anschließend an, alle betroffenen Knoten an die geänderten Includes anzupassen, so dass exakt die selben Knoten referenziert werden wie vorher. Dies ist z.B. dann sinnvoll, wenn Sie eine Suite zu den Includes hinzufügen oder daraus entfernen und die zugehörigen Referenzen auf Prozeduren und Komponenten implizit bzw. explizit machen wollen, ohne die Funktionalität zu verändern. Wenn Sie allerdings eine Suite umbenannt oder in ein anderes Verzeichnis verschoben haben und nun die Includes an die neuen Gegebenheiten anpassen, wählen Sie ”Nein”, so dass die vormals impliziten Referenzen auf die alte Suite nun auf die neue Suite verweisen.

9.0+ **Variabel:** Dateinamen für inkludierte Testsuiten können Umgebungsvariablen oder System-Properties referenzieren. Die Syntax dafür lautet `${env:...}` beziehungsweise `${system:...}`.

Sie können die inkludierte Testsuite während der Laufzeit ändern, indem Sie den in der Umgebungsvariablen oder System-Property hinterlegten Wert per Skript auf den neuen Wert setzen. Verwenden Sie hierzu `rc.setProperty`, das in Abschnitt 50.5[1030] beschrieben ist.

Hinweis Die obige Syntax entspricht zwar dem QF-Test Standard für Gruppen-Variablen oder Properties. Dies ist aber ein Sonderfall bei dem nur die Gruppen `env` bzw. `system` verwendet werden können.

Einschränkungen: Keine

Abhängige Dateien (umgekehrte Includes)

Diese Liste von Testsuiten ist das Gegenstück zu den Inkludierte Dateien. Sie hat keinen Einfluss auf die Ausführung von Tests, sondern dient QF-Test als Hinweis darauf, welche Testsuiten von Komponenten in dieser Testsuite abhängig sind, weil sie diese Suite über Inkludierte Dateien einbinden oder durch explizite Referenzen. Diese Information wird herangezogen, wenn QF-Test IDs von Komponenten geändert werden (z.B. durch die Funktion ”Komponenten aktualisieren”, siehe Abschnitt 5.11.2[105]). Die QF-Test ID der Komponente Attribute von abhängigen Knoten müssen in diesem Fall angepasst werden. Alle aktuell geladenen Testsuiten werden automatisch auf Abhängigkeiten geprüft. Zusätzlich lädt QF-Test die hier aufgeführten Testsuiten automatisch nach und überprüft sie ebenfalls.

Dateinamen können absolut oder relativ angegeben werden, wobei relative Dateinamen relativ zu dieser Testsuite oder zu Verzeichnissen des Bibliothekspfades aufgelöst werden (vgl. Option Verzeichnisse mit Testsuite-Bibliotheken[503]).

Hinweis Wie bei den Inkludierte Dateien werden die Abhängigen Dateien auch indirekt aufge-

löst. Hat z.B. Suite A die abhängige Suite B und diese wiederum die abhängige Suite C, so wird bei Änderungen an A sowohl B als auch C nachgeladen und überprüft.

9.0+ **Variabel:** Umgebungsvariablen und System-Properties können analog zum Attribut Inkludierte Dateien mit der Syntax `${env:...}` beziehungsweise `${system:...}` verwendet werden.

Einschränkungen: Keine

Variablendefinitionen

Diese Variablendefinitionen sind mit den Suite Variablen in den Variablen[592] Einstellungen der global Optionen identisch. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20]. Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6[116].

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximale Ausführungszeit

Zeit in Millisekunden, die der Knoten maximal ausgeführt werden soll. Nach Ablauf dieser Zeit wird der Knoten abgebrochen.

Variabel: Ja

Einschränkungen: >= 0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.2 Test- und Sequenz-Knoten

Tests und Sequenzen sind die wichtigsten Strukturelemente einer Testsuite. Testfall Knoten repräsentieren logische Testfälle und sind als spezialisierte Sequenzen implementiert. Eine Sequenz im Allgemeinen ist ein Container, dessen Childknoten einer nach dem anderen ausführt werden. Es können dabei Werte für Variablen (vgl. Kapitel 6[116]) definiert werden, die während der Ausführung der Childknoten Gültigkeit besitzen.

Darüber hinaus gibt es noch andere Arten von Sequenzen, die entweder besondere Einschränkungen bzgl. ihrer Parentknoten oder Childknoten haben, oder spezielle Eigenschaften bei der Ausführung aufweisen.

42.2.1 Testfall

Ein Testfall repräsentiert einen oder mehrere logische Testfälle. In gewissem Sinne ist er der wichtigste Knotentyp in QF-Test und alles andere dient nur dazu den Testfällen Struktur zu verleihen oder ihre Logik zu implementieren. Funktionell ist er eine besondere Sequenz[618] mit einer Reihe von wichtigen Erweiterungen und speziellen Attributen.

Ein Testfall sollte seinen Fokus auf den eigentlichen Test legen, der damit umgesetzt werden soll. Vorbereitungs- und Aufräumtätigkeiten, die notwendige Rahmenbedingungen für den Testlauf schaffen und Kollisionen mit nachfolgenden Tests vermeiden, sollten in Form von Abhängigkeiten[630], wie in Abschnitt 8.6[161] beschrieben, implementiert werden. Alternativ - oder auch zusätzlich zu Abhängigkeiten - kann ein Testfall Vorbereitung[638] und Aufräumen[641] Knoten besitzen, welche vor bzw. nach dem Testfall ausgeführt werden.

Ein Testfall kann über einen Testaufruf[614] aufgerufen werden und ist damit ähnlich einer Prozedur[672]. Sein Name[602] ist ein Pflichtattribut und er hat ein Liste von Standardwerten für Parameter[604], die im aufrufenden Knoten überschrieben werden können.

Testfälle spielen auch eine zentrale Rolle für Protokolle und Testreports. Während eines Testlaufs kann ein Testfall mehrmals in unterschiedlichen Kontexten und mit verschieden Parametern ausgeführt werden. Logisch können die Ausführungen dem gleichen oder unterschiedlichen Testfällen entsprechen. Durch die Definition eines Satzes von Charakteristische Variablen[602] lässt sich festlegen, welche Variablen verwendet werden sollen, um einzelne Testläufe zu unterscheiden - die also für die Laufzeitumgebung des Tests charakteristisch sind. Die Werte dieser Variablen zum Zeitpunkt des Eintretens in den Testfall werden im Protokoll gespeichert. Um hervorzuheben, dass jede Ausführung eines Testfall Knotens einem eigenen logischen Testfall entspricht, gibt es zusätzliche das Attribut Name für Protokoll und Report[602]. Dessen Wert kann Referenzen auf die Charakteristischen Variablen des Tests enthalten. Im Protokoll oder Report wird der Test dann

mit diesem Namen aufgeführt, inklusive eventueller Werte der Variablen zur Laufzeit.

Zu guter Letzt gibt es auch Situationen, in denen ein Test für spezifische Variablenwerte nicht ausgeführt werden kann oder soll. Falls eine Bedingung[603] für einen Testfall Knoten definiert ist, wird dieser Testfall nur ausgeführt, wenn die Auswertung der Bedingung einen wahren Wert ergibt. Wenn der Testfall wegen der Bedingung nicht zur Ausführung kommt, wird er im Report als *übersprungen* gelistet.

Enthalten in: Testsuite[595], Testfallsatz[606].

Kinder: Optional ein Abhängigkeit[630] oder Bezug auf Abhängigkeit[635] Knoten als erstes Element. Als nächstes kann eine Vorbereitung[638] folgen und ein Aufräumen[641] Knoten kann den Abschluss bilden, mit einer beliebigen Anzahl von normalen Childknoten dazwischen. Ein Testfall Knoten, der keine normalen Childknoten enthält, wird im Report als *nicht implementiert* aufgeführt.

Ausführung: Zunächst werden die Variablendefinitionen[604] des Testfall Knotens auf dem primären und die Standardwerte für Parameter[604] auf dem sekundären Variablenstapel (vgl. Kapitel 6[116]) gebunden. Anschließend wird die Bedingung[603] geprüft und der Knoten in dem Fall übersprungen, dass der Inhalt der (nicht leeren) Bedingung als *falsch* ausgewertet wird. Als nächstes folgt die Bestimmung und Durchführung der (möglicherweise auch vom Parentknoten geerbten) Abhängigkeit, wie in Abschnitt 8.6[161] beschrieben. Dann wird die optionale Vorbereitung einmalig ausgeführt, gefolgt von den normalen Childknoten und dem Aufräumen Knoten. Tritt während des Ablaufs des Testfall Knotens eine Exception auf, wird diese gefangen und an seine Abhängigkeit für die Behandlung in deren optionalen Catch[707] Knoten weitergereicht. Auch wenn keine Behandlung der Exception in der Abhängigkeit statt findet, wird sie nicht über den Testfall hinaus propagiert, um den Abbruch des gesamten Testlaufs zu vermeiden. Der Fehlerstatus wird jedoch stets korrekt im Protokoll und Report festgehalten.

Attribute:

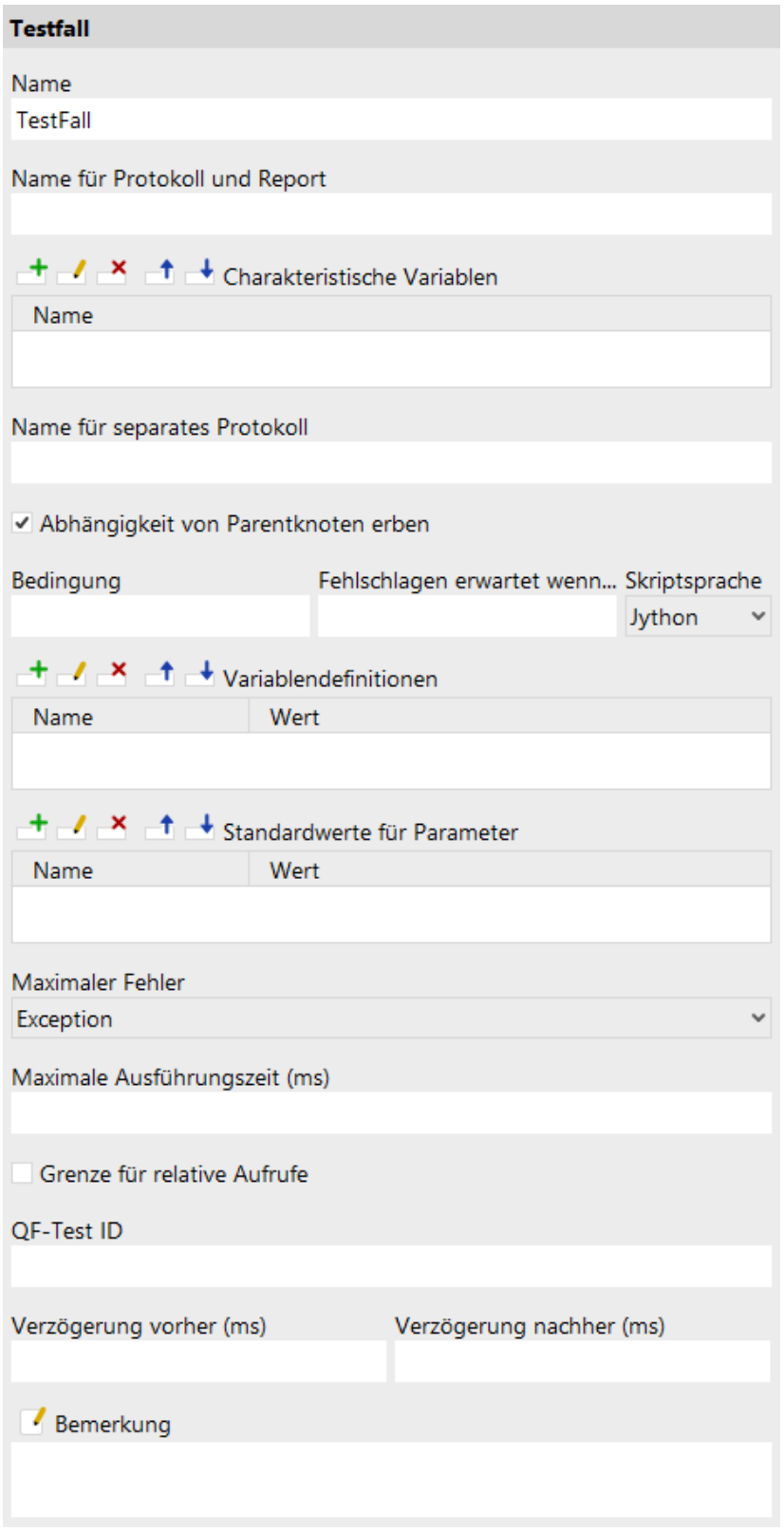


Abbildung 42.2: Testfall Attribute

Name

Ein Testfall wird durch seinen Namen und die Namen seiner Testfallsatz[606] Parentknoten identifiziert. Daher sollten Sie ”sprechende” Namen verwenden, die etwas über den jeweiligen Testfall aussagen und die Sie sich gut merken können.

Variabel: Nein

Einschränkungen: Darf nicht leer sein und keines der Zeichen '.' oder '#' enthalten.

Name für Protokoll und Report

Ein alternativer Name für Protokoll und Report. Dieser ist hilfreich, um zwischen Mehrfachausführungen mit potenziell unterschiedlichen Werten für die Charakteristischen Variablen zu unterscheiden.

Variabel: Ja

Einschränkungen: Keine

Charakteristische Variablen

Diese Variablen sind Teil der Charakteristik des Testfallsatzes oder Testfalls. Dass zwei Ausführungen eines Testfall Knotens den selben logischen Testfall repräsentieren, wird genau dann angenommen, wenn die Werte aller Charakteristische Variablen zur Laufzeit identisch sind. Die Werte der Variablen zur Laufzeit werden zudem im Protokoll gespeichert. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20].

Variabel: Nein

Einschränkungen: Keine

Name für separates Protokoll

Mit diesem Attribut kann ein Knoten als Bruchstelle zum Abteilen eines Protokolls markiert werden. Es legt den Dateinamen für das abgeteilte Protokoll fest. Nach Durchlaufen des Knotens wird das zugehörige Protokoll aus dem Hauptprotokoll entfernt und als eigenständiges Protokoll gespeichert. Diese Operation ist vollständig transparent, da das Hauptprotokoll eine Referenz auf das abgeteilte Protokoll erhält und somit vollständig navigierbar bleibt. Näheres zu geteilten Protokollen finden Sie in Abschnitt 7.1.6[143].

Dieses Attribut hat keinen Effekt, wenn die Option Geteilte Protokolle erzeugen[582] deaktiviert ist oder geteilte Protokolle durch explizite Angabe von `-splitlog`[990] im Batchmodus ausgeschaltet werden.

Es ist nicht nötig für Eindeutigkeit der Dateinamen für abgeteilte Protokolle zu sorgen. Wo nötig hängt QF-Test eine Zahl an den Dateinamen an, um Konflikte zu vermeiden. Der Dateiname darf Verzeichnisse enthalten und es können - analog

zur Angabe des Dateinamens für das Protokoll im Batchmodus - folgende Platzhalter in Kombination mit einem '%' oder '+' Zeichen verwendet werden:

Zeichen	Bedeutung
%	'%'-Zeichen.
+	'+'-Zeichen.
i	Die aktuelle Runid wie mit `-runid [<ID>]`(989) angegeben.
r	Die Fehlerstufe des abgeteilten Protokolls.
w	Die Anzahl der Warnungen im abgeteilten Protokoll.
e	Die Anzahl der Fehler im abgeteilten Protokoll.
x	Die Anzahl der Exceptions im abgeteilten Protokoll.
t	Der Threadindex zu dem das abgeteilte Protokoll gehört (für Tests mit parallelen Threads).
y	Das aktuelle Jahr (2 Ziffern).
Y	Das aktuelle Jahr (4 Ziffern).
M	Der aktuelle Monat (2 Ziffern).
d	Der aktuelle Tag (2 Ziffern).
h	Die aktuelle Stunde (2 Ziffern).
m	Die aktuelle Minute (2 Ziffern).
s	Die aktuelle Sekunde (2 Ziffern).

Tabelle 42.1: Platzhalter für das Attribut Name für separates Protokoll

Variabel: Ja

Einschränkungen: Keine, Zeichen die für Dateinamen nicht zulässig sind werden durch '_' ersetzt.

Abhängigkeit von Parentknoten erben

Diese Option erlaubt das Erben einer Abhängigkeit(630) vom Parentknoten als Ersatz oder zusätzlich zu einer für den Knoten selbst spezifizierten Abhängigkeit.

Variabel: Nein

Einschränkungen: Keine

Bedingung

Sofern die Bedingung nicht leer ist wird sie ausgewertet und, wenn das Resultat *falsch* ist, die Ausführung des aktuellen Knotens abgebrochen. In diesem Fall wird der Knoten als *übersprungen* gewertet.

Wie die Bedingung(694) eines If(693)-Knotens wird auch diese Bedingung vom Jython-Interpreter ausgewertet, so dass die selben Bemerkungen zutreffen.

Variabel: Ja

Einschränkungen: Gültige Syntax

Skriptsprache

Dieses Attribut legt den Interpreter fest, in dem das Skript ausgeführt wird, oder in anderen Worten die Skriptsprache. Mögliche Werte sind ”Jython”, ”Groovy” und ”JavaScript”.

Variabel: Nein

Einschränkungen: Keine

Fehlschlagen erwartet wenn...

Mit dieser Option kann eine Bedingung festgelegt werden, unter welcher ein Fehlschlagen des Testfalls erwartet wird. Dadurch können neue Fehler besser von bereits bekannten unterschieden werden. Natürlich sollten letztere ebenfalls beseitigt werden, aber ob und wann das geschieht entzieht sich eventuell dem Einfluss des Testers.

Meist ist es ausreichend, dieses Attribut auf ’true’ zu setzen, wenn ein Fehler für diesen Testfall erwartet wird. Falls der Testfall mehrfach ausgeführt wird, z.B. in einem Datentreiber(646) oder auf verschiedenen Systemen und nur in speziellen Fällen fehlschlägt, sollte die Bedingung so formuliert sein, dass sie für genau diese Fälle wahr ist, z.B. ${qftest:windows} für einen Testfall, der unter Windows fehlschlägt, auf anderen Systemen aber nicht.

Falls dieses Attribut ein wahres Ergebnis liefert und der Testfall mit einem Fehler scheitert, wird dieser in Protokoll, Report und Statuszeile gesondert aufgeführt. Trotzdem bedeutet dies einen Fehler in der Anwendung, so dass die Prozentangaben für erfolgreiche Tests davon unberührt bleiben.

Wird ein Scheitern erwartet und der Testfall ist dennoch erfolgreich, gilt dies als Fehler, da dies entweder bedeutet, dass der Test den Fehler nicht zuverlässig nachweist, oder der Fehler inzwischen beseitigt wurde und der Testfall entsprechend angepasst werden muss.

Variabel: Ja

Einschränkungen: Gültige Syntax

Variablendefinitionen

Diese Variablen werden auf dem Primärstapel für direkte Bindungen abgelegt (vgl. Kapitel 6(116)). Sie behalten während der Ausführung des Testfalls ihre Gültigkeit und können nicht durch einen Testaufruf(614) überschrieben werden. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20).

Variabel: Variable Namen nein, Werte ja

Einschränkungen: Keine

Standardwerte für Parameter

Hier können Sie Defaultwerte oder ”Fallback” Werte für die Parameter des Testfalls definieren. Diese werden herangezogen, wenn eine Variable an keiner anderen Stelle definiert wurde (vgl. Kapitel 6(116)). Außerdem dienen Sie als Dokumentation und bringen Zeitersparnis bei den Variablendefinitionen(616) Attributen, wenn der Dialog für die Auswahl des Testfalls im Attribut Name des Tests(615) des Testaufruf(614) Knotens verwendet wird. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

Maximale Ausführungszeit

Zeit in Millisekunden, die der Knoten maximal ausgeführt werden soll. Nach Ablauf dieser Zeit wird der Knoten abgebrochen.

Variabel: Ja

Einschränkungen: >= 0

Grenze für relative Aufrufe

Ist dieses Attribut gesetzt, ist ein relativer Prozeduraufruf, ein relativer Testaufruf oder eine relative Referenz auf eine Abhängigkeit innerhalb dieses Knotens gestattet. Relative Aufrufe bzw. Referenzen, die diese Grenze überschreiten sind nicht erlaubt. Wenn dieses Attribut in der gesamten Hierarchie nicht gesetzt ist, so können keine relativen Prozeduraufrufe einfügt werden. Relative Testaufrufe sind unterhalb des Testsuite Knotens immer möglich.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Bei der Ausführung von Tests im Batchmodus kann beim Kommandozeilenargument `-test <Index>|<ID>`(992) alternativ zum qualifizierten Namen die QF-Test ID des Knotens angegeben werden.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.2.2 Testfallsatz

Im Wesentlichen dienen Testfallsätze dazu, Testfälle[599] einer Testsuite zu strukturieren. Testfallsätze können verschachtelt werden. Der Name[609] eines Testfallsatz Knotens ist Bestandteil des vollständigen Namens eines Testfalls, auf den sich ein Testaufruf[614] bezieht. Testfallsätze sind selbst ebenfalls aufrufbar und haben deshalb einen Satz von Standardwerten für Parameter[612], die im Testaufruf überschrieben werden können.

Eine Möglichkeit, Testfälle zu strukturieren, ist es, solche mit ähnlichen Vorbereitungs- und Aufräumanforderungen in einem Testfallsatz zu sammeln. Dieser kann einen Abhängigkeit[630] oder Bezug auf Abhängigkeit[635] Knoten zur Verfügung stellen, der von den Testfällen oder verschachtelten Testfallsätzen geerbt werden kann. Wenn das Attribut Abhängigkeit von Parentknoten erben[611] im Testfallsatz aktiviert ist, werden die Abhängigkeit Knoten des oder der Parents ebenfalls vererbt. Im Abschnitt 8.6[161] finden Sie Detailinformationen zum Abhängigkeitsmechanismus in QF-Test. Alternatives oder zusätzliches Vorbereiten und Aufräumen ist in Form von Vorbereitung[638] und Aufräumen[641] Knoten möglich. Diese werden vor und nach jedem der Testfallsatz und Testfall Childknoten ausgeführt.

Ein weiteres wichtiges Feature im Testfallsatz ist datengetriebenes Testen. Dieses lässt sich durch das Hinzufügen eines Datentreiber[646] Knotens mit einem oder mehreren Daten Knoten erreichen, wie es in Kapitel 23[319] beschrieben wird.

Wie der Testfall, so spielt auch der Testfallsatz eine wichtige Rolle für den Report. Da er mehrfach mit unterschiedlichen Parametersätzen aufgerufen werden kann, verfügt er über Charakteristische Variablen[610] und einen alternativen Namen für Protokoll und Report[610], die äquivalent funktionieren wie für den Testfall. Das gleiche gilt für die Bedingung[611], die benutzt werden kann, um einen ganzen Testfallsatz, abhängig von aktuellen Variablenwerten, zu überspringen.

Enthalten in: Testsuite[595], Testfallsatz[606].

Kinder: Optional ein Abhängigkeit[630] oder Bezug auf Abhängigkeit[635] Knoten als erstes Element gefolgt von einem optionalen Datentreiber[646]. Als nächstes kann eine Vorbereitung[638] folgen und ein Aufräumen[641] Knoten kann den Abschluss bilden, mit einer beliebigen Anzahl von Testfallsatz[606], Testfall[599] und Testaufruf[614] Knoten dazwischen.

Ausführung: Zunächst werden die Standardwerte für Parameter[612] des Testfallsatz Knotens auf dem sekundären Variablenstapel (vgl. Kapitel 6[116]) gebunden. Anschließend wird die Bedingung[603] geprüft und der Knoten in dem Fall übersprungen, dass der Inhalt der (nicht leeren) Bedingung als *falsch* ausgewertet wird. Eine mögliche Abhängigkeit oder Bezug auf Abhängigkeit wird zu diesem Zeitpunkt nicht aufgelöst, es sei denn das Attribut Immer ausführen, auch in Testsuite- und Testfallsatz-Knoten[633] ist gesetzt. Ist ein Datentreiber[646] Knoten vorhanden, wird dieser ausgeführt, um einen entsprechenden Datenkontext zu erzeugen und einen oder mehrere Daten Knoten zu binden. Damit werden dann die anderen Childknoten in wiederholter Abfolge ausgeführt (vgl. Kapitel

23[319]). Für jeden Schleifendurchlauf - oder einmal im dem Fall dass kein Datentreiber vorhanden ist - bringt der Testfallsatz seine Testfallsatz oder Testfall Kindknoten zur Ausführung. Wenn optionale Vorbereitung[638] oder Aufräumen[641] Knoten vorhanden sind, werden diese vor bzw. nach jedem der Childknoten ausgeführt.

Attribute:

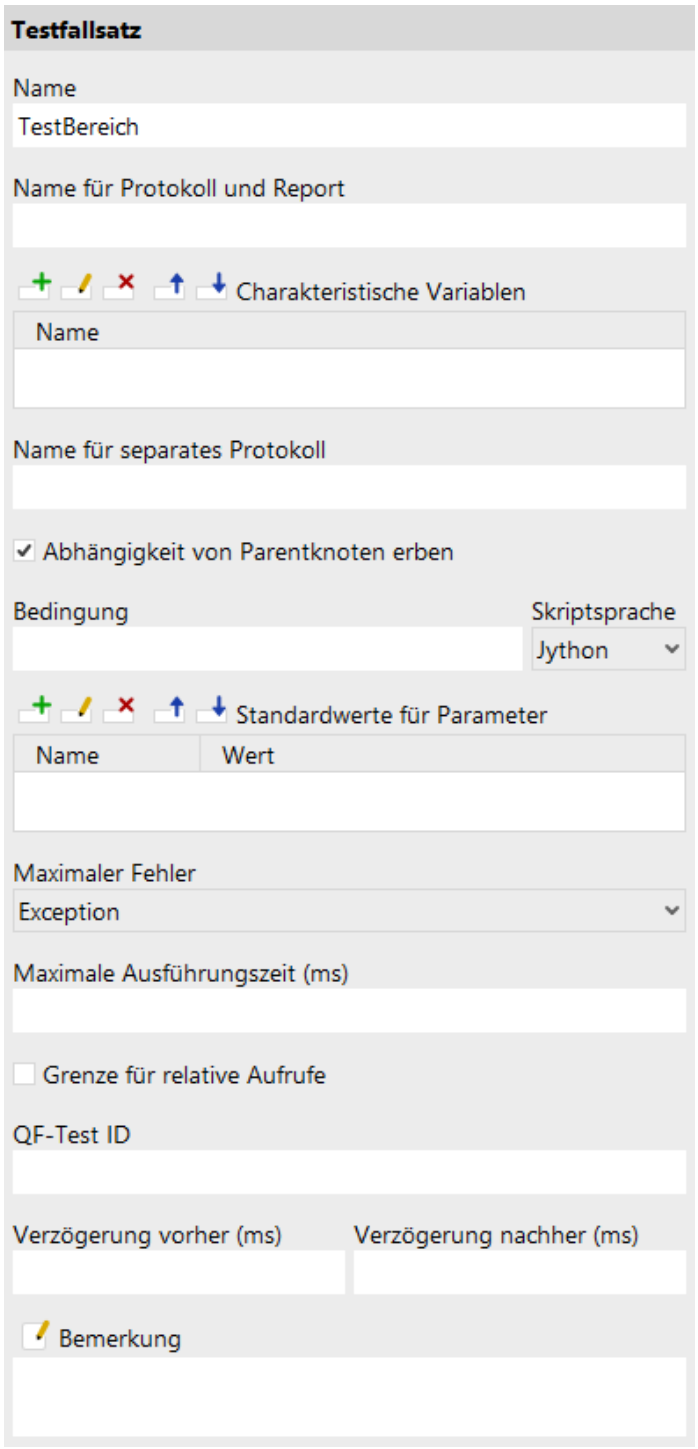


Abbildung 42.3: Testfallsatz Attribute

Name

Der Name eines Testfallsatzes ist ein Teil seiner eigenen Identifikation und der in ihm enthaltenen Testfall[599] und Testfallsatz Knoten. Verwenden Sie "sprechende" Namen, die etwas über die darin enthaltenen Tests aussagen und die Sie sich gut merken können.

Variabel: Nein

Einschränkungen: Darf nicht leer sein und keines der Zeichen '.' oder '#' enthalten.

Name für Protokoll und Report

Ein alternativer Name für Protokoll und Report. Dieser ist hilfreich, um zwischen Mehrfachausführungen mit potenziell unterschiedlichen Werten für die Charakteristischen Variablen zu unterscheiden.

Variabel: Ja

Einschränkungen: Keine

Charakteristische Variablen

Diese Variablen sind Teil der Charakteristik des Testfallsatzes oder Testfalls. Dass zwei Ausführungen eines Testfall Knotens den selben logischen Testfall repräsentieren, wird genau dann angenommen, wenn die Werte aller Charakteristische Variablen zur Laufzeit identisch sind. Die Werte der Variablen zur Laufzeit werden zudem im Protokoll gespeichert. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20].

Variabel: Nein

Einschränkungen: Keine

Name für separates Protokoll

Mit diesem Attribut kann ein Knoten als Bruchstelle zum Abteilen eines Protokolls markiert werden. Es legt den Dateinamen für das abgeteilte Protokoll fest. Nach Durchlaufen des Knotens wird das zugehörige Protokoll aus dem Hauptprotokoll entfernt und als eigenständiges Protokoll gespeichert. Diese Operation ist vollständig transparent, da das Hauptprotokoll eine Referenz auf das abgeteilte Protokoll erhält und somit vollständig navigierbar bleibt. Näheres zu geteilten Protokollen finden Sie in Abschnitt 7.1.6[143].

Dieses Attribut hat keinen Effekt, wenn die Option Geteilte Protokolle erzeugen[582] deaktiviert ist oder geteilte Protokolle durch explizite Angabe von `-splitlog`[990] im Batchmodus ausgeschaltet werden.

Es ist nicht nötig für Eindeutigkeit der Dateinamen für abgeteilte Protokolle zu sorgen. Wo nötig hängt QF-Test eine Zahl an den Dateinamen an, um Konflikte zu vermeiden. Der Dateiname darf Verzeichnisse enthalten und es können - analog

zur Angabe des Dateinamens für das Protokoll im Batchmodus - folgende Platzhalter in Kombination mit einem '%' oder '+' Zeichen verwendet werden:

Zeichen	Bedeutung
%	'%'-Zeichen.
+	'+'-Zeichen.
i	Die aktuelle Runid wie mit `-runid [<ID>]`(989) angegeben.
r	Die Fehlerstufe des abgeteilten Protokolls.
w	Die Anzahl der Warnungen im abgeteilten Protokoll.
e	Die Anzahl der Fehler im abgeteilten Protokoll.
x	Die Anzahl der Exceptions im abgeteilten Protokoll.
t	Der Threadindex zu dem das abgeteilte Protokoll gehört (für Tests mit parallelen Threads).
y	Das aktuelle Jahr (2 Ziffern).
Y	Das aktuelle Jahr (4 Ziffern).
M	Der aktuelle Monat (2 Ziffern).
d	Der aktuelle Tag (2 Ziffern).
h	Die aktuelle Stunde (2 Ziffern).
m	Die aktuelle Minute (2 Ziffern).
s	Die aktuelle Sekunde (2 Ziffern).

Tabelle 42.2: Platzhalter für das Attribut Name für separates Protokoll

Variabel: Ja

Einschränkungen: Keine, Zeichen die für Dateinamen nicht zulässig sind werden durch '_' ersetzt.

Abhängigkeit von Parentknoten erben

Diese Option erlaubt das Erben einer Abhängigkeit(630) vom Parentknoten als Ersatz oder zusätzlich zu einer für den Knoten selbst spezifizierten Abhängigkeit.

Variabel: Nein

Einschränkungen: Keine

Bedingung

Sofern die Bedingung nicht leer ist wird sie ausgewertet und, wenn das Resultat *falsch* ist, die Ausführung des aktuellen Knotens abgebrochen. In diesem Fall wird der Knoten als *übersprungen* gewertet.

Wie die Bedingung(694) eines If(693)-Knotens wird auch diese Bedingung vom Jython-Interpreter ausgewertet, so dass die selben Bemerkungen zutreffen.

Variabel: Ja

Einschränkungen: Gültige Syntax

Skriptsprache

Dieses Attribut legt den Interpreter fest, in dem das Skript ausgeführt wird, oder in anderen Worten die Skriptsprache. Mögliche Werte sind ”Jython”, ”Groovy” und ”JavaScript”.

Variabel: Nein

Einschränkungen: Keine

Standardwerte für Parameter

Hier können Sie Defaultwerte oder ”Fallback” Werte für die Parameter des Testfallsatzes definieren. Diese werden herangezogen, wenn eine Variable an keiner anderen Stelle definiert wurde (vgl. Kapitel 6[116]). Außerdem dienen Sie als Dokumentation und bringen Zeitersparnis für die Variablendefinitionen[616] Attribute wenn der Dialog für die Auswahl des Testfalls im Attribut Name des Tests[615] des Testaufruf[614] Knotens verwendet wird. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20].

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7[13]). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen[590]), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

Maximale Ausführungszeit

Zeit in Millisekunden, die der Knoten maximal ausgeführt werden soll. Nach Ablauf dieser Zeit wird der Knoten abgebrochen.

Variabel: Ja

Einschränkungen: >= 0

Grenze für relative Aufrufe

Ist dieses Attribut gesetzt, ist ein relativer Prozeduraufruf, ein relativer Testaufruf oder eine relative Referenz auf eine Abhängigkeit innerhalb dieses Knotens gestattet. Relative Aufrufe bzw. Referenzen, die diese Grenze überschreiten sind nicht erlaubt. Wenn dieses Attribut in der gesamten Hierarchie nicht gesetzt ist, so können keine relativen Prozeduraufrufe einfügt werden. Relative Testaufrufe sind unterhalb des Testsuite Knotens immer möglich.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Bei der Ausführung von Tests im Batchmodus kann beim Kommandozeilenargument `-test <Index>|<ID>`(992) alternativ zum qualifizierten Namen die QF-Test ID des Knotens angegeben werden.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.2.3 Testaufruf

Mit diesem Knoten kann der Aufruf eines anderen ausführbaren Testknotens realisiert werden. Mögliche Zielknoten sind Testsuite[595], Testfallsatz[606] und Testfall[599] in der gleichen oder einer anderen Testsuite. Die Ausführung wird im aufgerufenen Knoten fortgesetzt und kehrt nach Beendigung zum Testaufruf zurück (und damit zu seinem Parentknoten).

Der Name des aufzurufenden Testfall oder Testfallsatz Knotens wird durch dessen Namen[602] und die Namen[609] seiner Testfallsatz Parents bestimmt. Diese werden verknüpft, mit einem Punkt ('.') als Trennzeichen, beginnend mit äußersten Testfallsatz und endend mit dem Namen des Testfalls. Wollen Sie z.B. einen Testfall `nodeTest` im Testfallsatz `Tree`, der seinerseits im Testfallsatz `Main` enthalten ist, aufrufen, so müssen Sie als Name '`Main.Tree.nodeTest`' angeben. Knoten in anderen Testsuiten können durch das Voranstellen des Dateinamens der Testsuite, gefolgt von einem '#', aufgerufen werden. Ein Testsuite Knoten kann über einen einzelnen '.' adressiert werden, so dass eine gesamte Testsuite also mit einem Name des Tests Attribut der Form '`suitename.qft#.`' aufgerufen werden kann. Generell ist es am einfachsten, das Ziel interaktiv über den Knopf oberhalb des Name des Tests Attributs auszuwählen.

Weitere Informationen zu Aufrufen in anderen Testsuiten finden Sie auch in Abschnitt 26.1[359].

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Variablendefinitionen[616] des Testaufrufs werden gebunden, der Zielknoten bestimmt und die Ausführung an diesen übergeben. Wenn der Testaufruf beendet ist, werden die Variablen des Testaufrufs gelöscht.

Attribute:

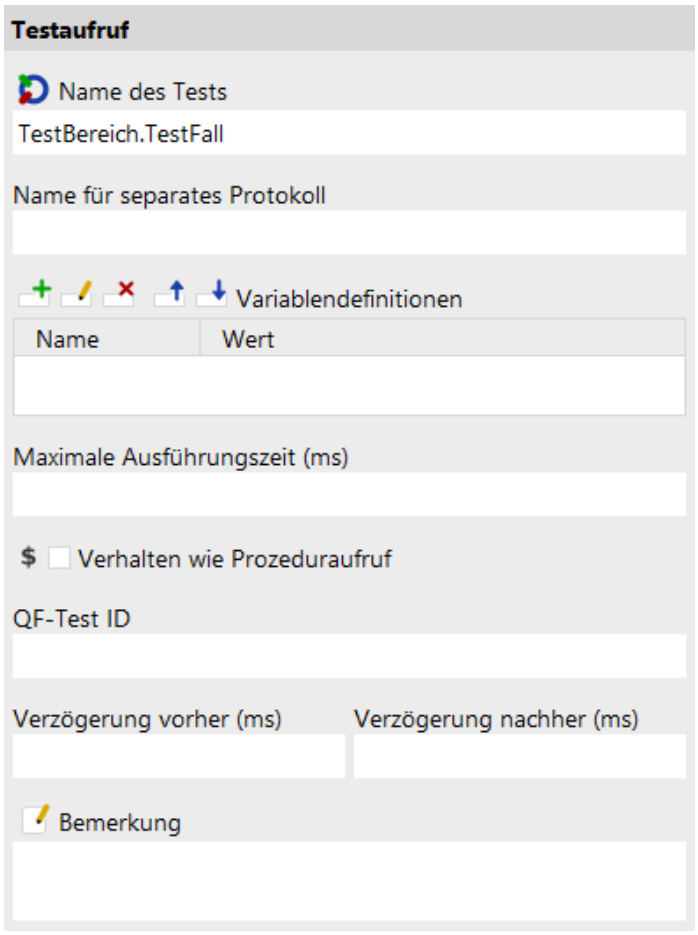


Abbildung 42.4: Testaufruf Attribute

Name

Im Falles eines Testfall oder Testfallsatz Knotens ist es der vollständige Name, welcher sich aus dem Namen(609) seiner Testfallsatz Parentknoten, gefolgt von seinem eigenen Namen(602) zusammensetzt, wobei die Einzelnamen durch einen '.' verknüpft werden. Eine Testsuite wird als einzelner '.' repräsentiert. The "Test auswählen" Button oberhalb des Attributs öffnet einen Dialog, in dem der Zielknoten interaktiv ausgewählt werden kann. Den Dialog erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Durch Setzen des "Parameter kopieren" Auswählkästchens, können die Standardwerte für Parameter des Testfall oder Testfallsatz Knotens für den Testaufruf übernommen werden, was erneutes Eintippen erspart.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Name für separates Protokoll

Mit diesem Attribut kann ein Knoten als Bruchstelle zum Abteilen eines Protokolls markiert werden. Es legt den Dateinamen für das abgeteilte Protokoll fest. Nach Durchlaufen des Knotens wird das zugehörige Protokoll aus dem Hauptprotokoll entfernt und als eigenständiges Protokoll gespeichert. Diese Operation ist vollständig transparent, da das Hauptprotokoll eine Referenz auf das abgeteilte Protokoll erhält und somit vollständig navigierbar bleibt. Näheres zu geteilten Protokollen finden Sie in Abschnitt 7.1.6(143).

Dieses Attribut hat keinen Effekt, wenn die Option Geteilte Protokolle erzeugen(582) deaktiviert ist oder geteilte Protokolle durch explizite Angabe von `-splitlog`(990) im Batchmodus ausgeschaltet werden.

Es ist nicht nötig für Eindeutigkeit der Dateinamen für abgeteilte Protokolle zu sorgen. Wo nötig hängt QF-Test eine Zahl an den Dateinamen an, um Konflikte zu vermeiden. Der Dateiname darf Verzeichnisse enthalten und es können - analog zur Angabe des Dateinamens für das Protokoll im Batchmodus - folgende Platzhalter in Kombination mit einem '%' oder '+' Zeichen verwendet werden:

Zeichen	Bedeutung
%	'%'-Zeichen.
+	'+'-Zeichen.
i	Die aktuelle Runid wie mit `-runid [<ID>]`(989) angegeben.
r	Die Fehlerstufe des abgeteilten Protokolls.
w	Die Anzahl der Warnungen im abgeteilten Protokoll.
e	Die Anzahl der Fehler im abgeteilten Protokoll.
x	Die Anzahl der Exceptions im abgeteilten Protokoll.
t	Der Threadindex zu dem das abgeteilte Protokoll gehört (für Tests mit parallelen Threads).
y	Das aktuelle Jahr (2 Ziffern).
Y	Das aktuelle Jahr (4 Ziffern).
M	Der aktuelle Monat (2 Ziffern).
d	Der aktuelle Tag (2 Ziffern).
h	Die aktuelle Stunde (2 Ziffern).
m	Die aktuelle Minute (2 Ziffern).
s	Die aktuelle Sekunde (2 Ziffern).

Tabelle 42.3: Platzhalter für das Attribut Name für separates Protokoll

Variabel: Ja

Einschränkungen: Keine, Zeichen die für Dateinamen nicht zulässig sind werden durch '_' ersetzt.

Variablendefinitionen

Hier können Sie Parameterwerte für den Zielknoten definieren. Diese werden auf dem primären Variablenstapel gebunden (vgl. Kapitel 6[116]), so dass sie Standardwerte für Parameter überschreiben. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20].

4.2+ Mittels Rechtsklick und Auswahl von Parameterordnung zurücksetzen, können Sie die Sortierung der Parameter, wie diese im aufgerufenen Testfall bzw. Testfallsatz sortiert sind, wieder herstellen.

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Verhalten wie Prozeduraufruf

Falls der Testaufruf Knoten innerhalb eines Testfalls ausgeführt wird, steuert dieses Attribut, wie Exceptions in den aufgerufenen Knoten behandelt werden. Ist es gesetzt, beendet eine Exception den gesamten Aufruf unabhängig von der Verschachtelung von Testfallsatz und Testfall Knoten, analog zu einem Prozeduraufruf. Ist dieses Attribut nicht gesetzt, bleibt die besondere Rolle Testfallsatz und Testfall Knoten mit ihrer lokalen Behandlung von Exceptions erhalten.

Variabel: Ja

Einschränkungen: Keine

Maximale Ausführungszeit

Zeit in Millisekunden, die der Knoten maximal ausgeführt werden soll. Nach Ablauf dieser Zeit wird der Knoten abgebrochen.

Variabel: Ja

Einschränkungen: >= 0

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.2.4 Sequenz

Diese einfachste Form aller Sequenzen hat beliebig viele Childknoten, die sie der Reihe nach abarbeitet.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Beliebig

Ausführung: Die Variablen der Sequenz werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

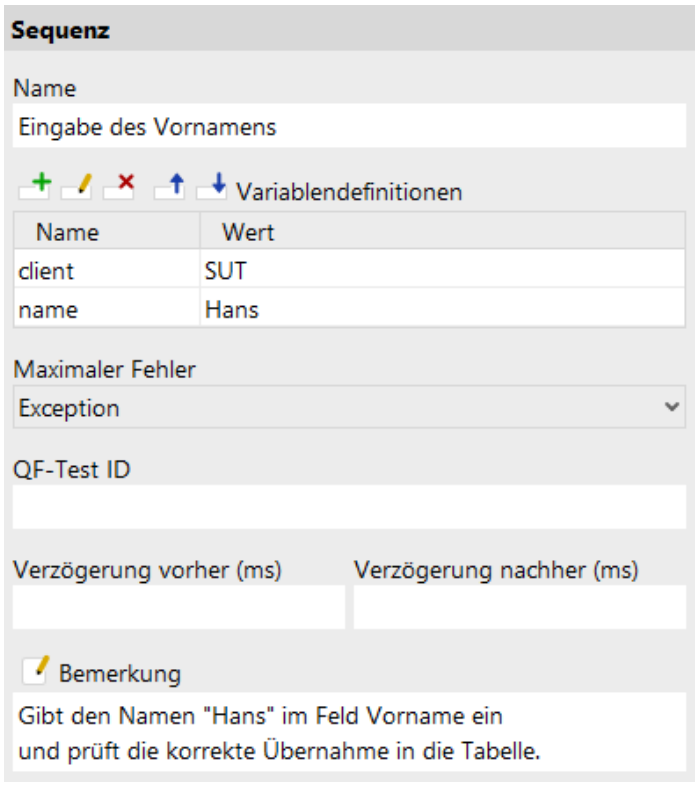


Abbildung 42.5: Sequenz Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7[13]). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen[590]), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.2.5 Testschritt

Ein Testschritt ist eine spezielle Sequenz zur Unterteilung eines Testfalls in mehrere Schritte, welche individuell dokumentiert werden können und im Report und der testdoc Dokumentation wiedergegeben werden. Im Gegensatz zu Testfällen, die nicht verschachtelt werden dürfen, können Testschritte beliebig tief verschachtelt werden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Beliebig

Ausführung: Die Variablen des Testschritts werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen des Testschritts wieder gelöscht.

Attribute:

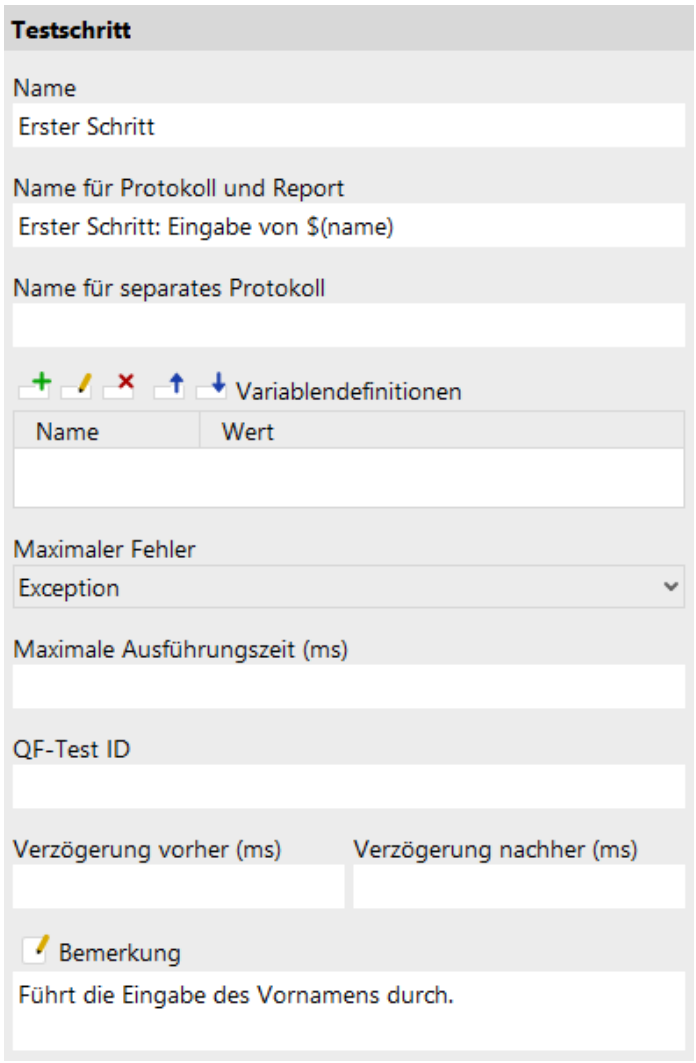


Abbildung 42.6: Testschritt Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Name für Protokoll und Report

Ein alternativer Name für Protokoll und Report. Dieser ist hilfreich, um zwischen

Mehrfachausführungen des umgebenden Testfalls mit potenziell unterschiedlichen Werten für dessen Charakteristischen Variablen zu unterscheiden.

Variabel: Ja

Einschränkungen: Keine

Name für separates Protokoll

Mit diesem Attribut kann ein Knoten als Bruchstelle zum Abteilen eines Protokolls markiert werden. Es legt den Dateinamen für das abgeteilte Protokoll fest. Nach Durchlaufen des Knotens wird das zugehörige Protokoll aus dem Hauptprotokoll entfernt und als eigenständiges Protokoll gespeichert. Diese Operation ist vollständig transparent, da das Hauptprotokoll eine Referenz auf das abgeteilte Protokoll erhält und somit vollständig navigierbar bleibt. Näheres zu geteilten Protokollen finden Sie in Abschnitt 7.1.6(143).

Dieses Attribut hat keinen Effekt, wenn die Option Geteilte Protokolle erzeugen(582) deaktiviert ist oder geteilte Protokolle durch explizite Angabe von `-splitlog`(990) im Batchmodus ausgeschaltet werden.

Es ist nicht nötig für Eindeutigkeit der Dateinamen für abgeteilte Protokolle zu sorgen. Wo nötig hängt QF-Test eine Zahl an den Dateinamen an, um Konflikte zu vermeiden. Der Dateiname darf Verzeichnisse enthalten und es können - analog zur Angabe des Dateinamens für das Protokoll im Batchmodus - folgende Platzhalter in Kombination mit einem '%' oder '+' Zeichen verwendet werden:

Zeichen	Bedeutung
%	'%'-Zeichen.
+	'+'-Zeichen.
i	Die aktuelle Runid wie mit `-runid [<ID>]`(989) angegeben.
r	Die Fehlerstufe des abgeteilten Protokolls.
w	Die Anzahl der Warnungen im abgeteilten Protokoll.
e	Die Anzahl der Fehler im abgeteilten Protokoll.
x	Die Anzahl der Exceptions im abgeteilten Protokoll.
t	Der Threadindex zu dem das abgeteilte Protokoll gehört (für Tests mit parallelen Threads).
y	Das aktuelle Jahr (2 Ziffern).
Y	Das aktuelle Jahr (4 Ziffern).
M	Der aktuelle Monat (2 Ziffern).
d	Der aktuelle Tag (2 Ziffern).
h	Die aktuelle Stunde (2 Ziffern).
m	Die aktuelle Minute (2 Ziffern).
s	Die aktuelle Sekunde (2 Ziffern).

Tabelle 42.4: Platzhalter für das Attribut Name für separates Protokoll

Variabel: Ja

Einschränkungen: Keine, Zeichen die für Dateinamen nicht zulässig sind werden durch '_' ersetzt.

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf "Keinen Fehler" zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

Maximale Ausführungszeit

Zeit in Millisekunden, die der Knoten maximal ausgeführt werden soll. Nach Ablauf dieser Zeit wird der Knoten abgebrochen.

Variabel: Ja

Einschränkungen: >= 0

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.2.6 Sequenz mit Zeitlimit

Dieser Knoten erweitert den gewöhnlichen Sequenz(618) Knoten, um die Überprüfung der Laufzeit. Seine Childknoten werden ganz normal ausgeführt, aber nach Beendigung der Sequenz wird die verstrichene Zeit mit dem vorgegebenen Zeitlimit verglichen. Wird das Zeitlimit überschritten, führt das zu einer Warnung, einem Fehler oder einer Exception, je nach Einstellung des Attributs Fehlerstufe bei Zeitüberschreitung(627). Explizite Verzögerungen wie Verzögerung vorher/nachher(628) oder Unterbrechungen durch den Anwender werden vor der Prüfung von der Laufzeit abgezogen.

Für die Reportgenerierung werden Zeitlimits wie Checks behandelt. Beginnt die Bemerkung(628) mit einem '!', wird das Ergebnis im Report als Check dargestellt.

Hinweis

Dieser Knoten dient zur Überprüfung von Zeitlimits im Bereich der Anwenderakzeptanz,

d.h. zwischen ein paar hundert Millisekunden und einigen Sekunden. Checks im Echtzeitbereich von einigen Millisekunden und weniger sind damit nicht möglich.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Beliebig

Ausführung: Die Variablen der Sequenz werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht. Die bei der Ausführung verstrichene Zeit wird auf Einhaltung des Zeitlimits überprüft.

Attribute:

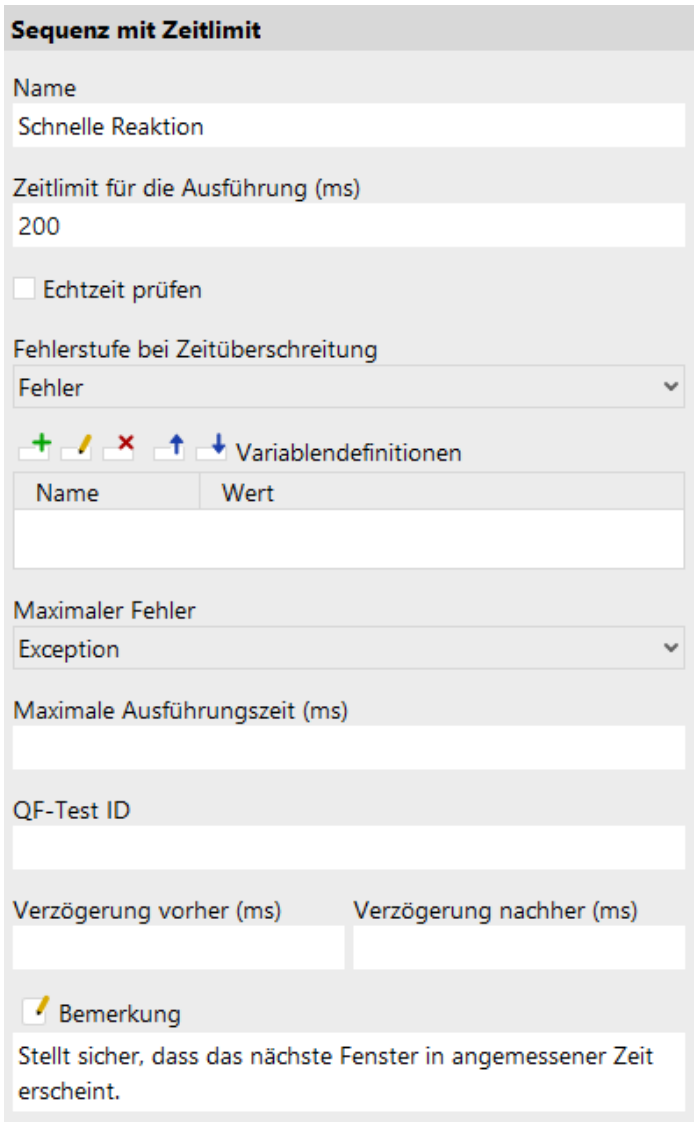


Abbildung 42.7: Sequenz mit Zeitlimit Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Zeitlimit für die Ausführung

Die erlaubte Zeit (in Millisekunden) für die Ausführung der Sequenz.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

Echtzeit prüfen

Normalerweise werden explizite Verzögerungen wie Verzögerung vorher/nachher(628) oder Unterbrechungen durch den Anwender vor der Prüfung von der Laufzeit abgezogen. Soll dies umgangen und somit die Echtzeit geprüft werden, ist dieses Attribut zu aktivieren.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe bei Zeitüberschreitung

Dieses Attribut legt fest, was bei Überschreitung des Zeitlimits passiert. Ist der Wert ”Exception”, wird eine `CheckFailedException`(963) geworfen. Andernfalls wird eine Meldung mit der entsprechenden Fehlerstufe in das Protokoll geschrieben.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die

übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

Maximale Ausführungszeit

Zeit in Millisekunden, die der Knoten maximal ausgeführt werden soll. Nach Ablauf dieser Zeit wird der Knoten abgebrochen.

Variabel: Ja

Einschränkungen: >= 0

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.2.7 Extrasequenzen

Dieser Knoten ist eine Art Spielwiese zum Ausprobieren und Zusammenstellen von Tests. Hier können Sie beliebige Knoten ablegen, unbehindert von den normalen Einschränkungen. Auch als Zwischenablage sind die Extrasequenzen nützlich.

Enthalten in: Wurzelknoten

Kinder: Beliebig

Ausführung: Kann nicht ausgeführt werden

Attribute:

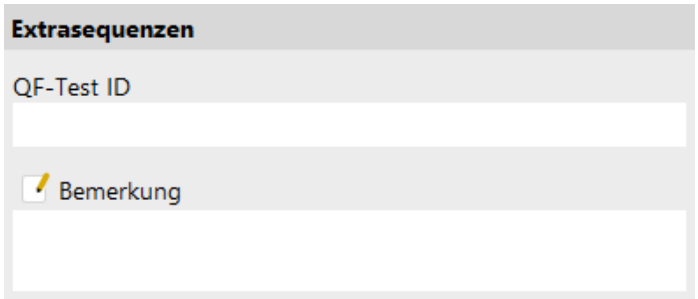


Abbildung 42.8: Extrasequenzen Attribute

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.3 Abhängigkeiten

Abhängigkeiten sind ein sehr mächtiges Feature für die automatische Behandlung von Anforderungen für die Vorbereitung und das Aufräumen bei Testfällen. Das Ziel ist es, Testfälle so zu isolieren, so dass sie unabhängig von einander lauffähig sind und sich nicht gegenseitig beeinflussen. Dies ist eine sehr wichtige Voraussetzung unter anderem für das Ausführen einer beliebigen Untermenge von Testfällen, z.B. um nur fehlerhafte Test nochmal anzustoßen, oder während der Testentwicklung, wo es möglich sein soll, jeden beliebigen Testfall schnell auszuführen und zu debuggen.

42.3.1 Abhängigkeit

Abhängigkeit Knoten werden benutzt, um eine fortgeschrittene, halbautomatische Behandlung von Anforderungen des Vorbereitens und Aufräumens für Testfallsätze[606] und Testfälle[599] zu implementieren. Eine detaillierte Beschreibung des Abhängigkeitsmechanismus in QF-Test finden Sie in Abschnitt 8.6[161]. Dieser Abschnitt konzentriert sich auf die formale Beschreibung der Abhängigkeit Knoten und ihrer Attribute.

Da Abhängigkeiten durchaus komplex sind, sollten sie so oft wie möglich wiederverwendet werden. Dies kann in der Weise geschehen, dass Testfälle mit gleichen Abhängigkeiten in einem Testfallsatz gruppiert werden und sie die Abhängigkeit von diesem Testfallsatz erben. Jedoch ist dieser Mechanismus alleine nicht flexibel genug, weshalb Abhängigkeiten zusätzlich wie Prozedur(672) Knoten unter den Prozeduren(682) einer Testsuite abgelegt werden können. Dort können sie mittels eines Bezug auf Abhängigkeit Knotens referenziert zu werden. Damit dies funktioniert, ist der Name(602) ein Pflichtattribut und es gibt auch eine Liste mit Standardwerten für Parameter(633), die im aufrufenden Knoten überschrieben werden können.

Die Charakteristische Variablen(633) einer Abhängigkeit sind Teil ihrer Identität und spielen eine wichtige Rolle Aufbau des Stapels der Abhängigkeiten.

Enthalten in: Testsuite(595), Testfallsatz(606), Testfall(599), Prozeduren(682), Package(680).

Kinder: Null oder mehrere Bezüge auf Abhängigkeiten(635), auf welchen die Abhängigkeit basiert, optionale Vorbereitung(638) und Aufräumen(641) Knoten und eine optionale Fehlerbehandlung(643), gefolgt von null oder mehreren Catch(707) Knoten.

Ausführung: In der Regel werden Abhängigkeiten nur indirekt während der Vorbereitungsphase eines Testfallsatzes oder Testfalls ausgeführt. Wenn ein Abhängigkeiten Knoten interaktiv aufgerufen wird, führt dies zu einem Aufbau des Stapels von Abhängigkeiten, wie in Abschnitt 8.6(161) beschrieben.

Attribute:

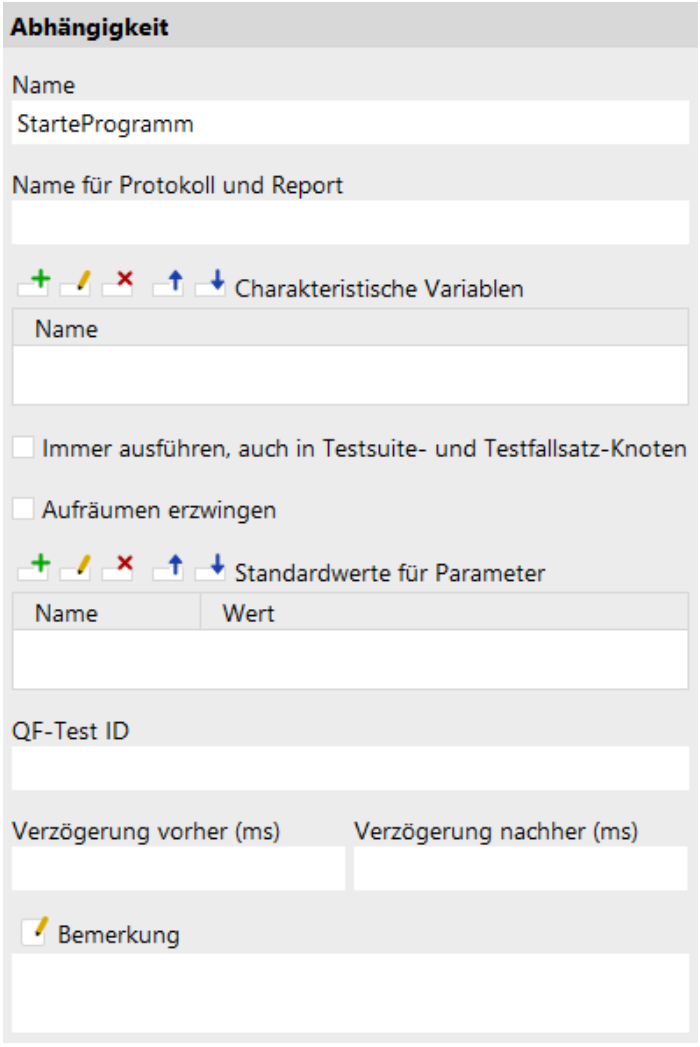


Abbildung 42.9: Abhängigkeit Attribute

Name

Eine Abhängigkeit wird durch ihren Namen identifiziert. Daher sollten Sie ”sprechende” Namen verwenden, die etwas über die jeweilige Abhängigkeit aussagen und die Sie sich gut merken können.

Variabel: Nein

Einschränkungen: Darf nicht leer sein und keines der Zeichen ’.’ oder ’#’ enthalten.

Name für Protokoll und Report

Ein alternativer Name für Protokoll und Report. Dieser ist hilfreich, um zwischen

Mehrfachausführungen mit potenziell unterschiedlichen Werten für die Charakteristischen Variablen zu unterscheiden.

Variabel: Ja

Einschränkungen: Keine

Charakteristische Variablen

Diese Variablen sind Teil der Charakteristik einer Abhängigkeit. Beim Aufbau des Stapels der Abhängigkeiten, wie in Abschnitt 8.6[161] beschrieben, werden zwei Abhängigkeiten als gleich betrachtet, wenn es sich um ein und den selben Knoten handelt und wenn die Werte aller Charakteristische Variablen identisch sind.

Bei der Ausführung des Aufräumen Knotens einer Abhängigkeit besitzt eine charakteristische Variable den gleichen Wert wie bei der Ausführung des zugehörigen Vorbereitung Knotens - unabhängig vom Wert der Variablen im aktuellen Testfall.

Variabel: Nein

Einschränkungen: Keine

Immer ausführen, auch in Testsuite- und Testfallsatz-Knoten

In der Regel wird eine Abhängigkeit nur ausgeführt, wenn sie zu einem Testfall Knoten gehört. Abhängigkeiten in Testsuite oder Testfallsatz Knoten werden einfach von den Testfall Childknoten dieses Knotens geerbt. In einigen Fällen kann es jedoch hilfreich sein, die Abhängigkeit vorzeitig aufzulösen, z.B. wenn von der Abhängigkeit Parameter für den Testlauf bereitgestellt werden, die nötig sind, um die Bedingung[603] eines nachfolgenden Testfalls auszuwerten. Durch Aktivieren dieser Option lässt sich genau das erreichen.

Variabel: Nein

Einschränkungen: Keine

Aufräumen erzwingen

In der Regel werden Abhängigkeiten nur zurück gerollt und ihre Aufräumsequenz ausgeführt, wenn der Auflösungsmechanismus für Abhängigkeiten, wie in Abschnitt 8.6[161] beschrieben, dies erfordert. In einigen Fällen macht es Sinn ein teilweises Aufräumen des Stapels von Abhängigkeiten zu erzwingen, gleich nachdem ein Testfall beendet wird. Dies kann über das Aufräumen erzwingen Attribut erreicht werden. Ist diese Option aktiviert, wird der Stapel von Abhängigkeiten mindestens bis zu und inklusive der zugehörigen Abhängigkeit aufgelöst.

Variabel: Nein

Einschränkungen: Keine

Standardwerte für Parameter

Hier können Sie Defaultwerte oder ”Fallback” Werte für die Parameter der Abhängigkeit definieren. Diese werden herangezogen, wenn eine Variable an keiner anderen Stelle definiert wurde (vgl. Kapitel 6[116]). Außerdem dienen Sie als Dokumentation und bringen Zeitersparnis bei den Variablendefinitionen[637] Attributen, wenn der Dialog für die Auswahl der Abhängigkeit im Bezug auf Abhängigkeit[635] Knoten verwendet wird. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20].

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.3.2 Bezug auf Abhängigkeit

Ein Bezug auf Abhängigkeit Knoten steht einfach für eine Abhängigkeit[630], die an einer anderen Stelle in einem Package[680] definiert ist. Der Name der referenzierten Abhängigkeit wird durch ihren Namen[632] und den ihrer möglichen Package Parentknoten bestimmt. Diese werden verknüpft, mit einem Punkt ('.') als Trennzeichen, beginnend mit dem äußersten Package und endend mit dem Namen der Abhängigkeit. Wollen Sie z.B. eine Abhängigkeit `demoStarted` im Package `Demo`, das seinerseits im Package `Main` enthalten ist, referenzieren, so müssen Sie als Name '`Main.Demo.demoStarted`' angeben.

Informationen zu Referenzen in andere Testsuiten finden Sie in Abschnitt 26.1[359].

Enthalten in: Testsuite[595], Testfallsatz[606], Testfall[599] und Abhängigkeit[630]

Kinder: Keine

Ausführung: In der Regel werden Bezüge auf Abhängigkeiten nur indirekt während der Vorbereitungsphase eines Testfallsatzes oder Testfalls ausgeführt. Wenn ein Bezug auf Abhängigkeit Knoten interaktiv aufgerufen wird, führt dies zur Bestimmung der referenzierten Abhängigkeit und dem Aufbau des Stapels von Abhängigkeiten, wie in Abschnitt 8.6[161] beschrieben.

Attribute:

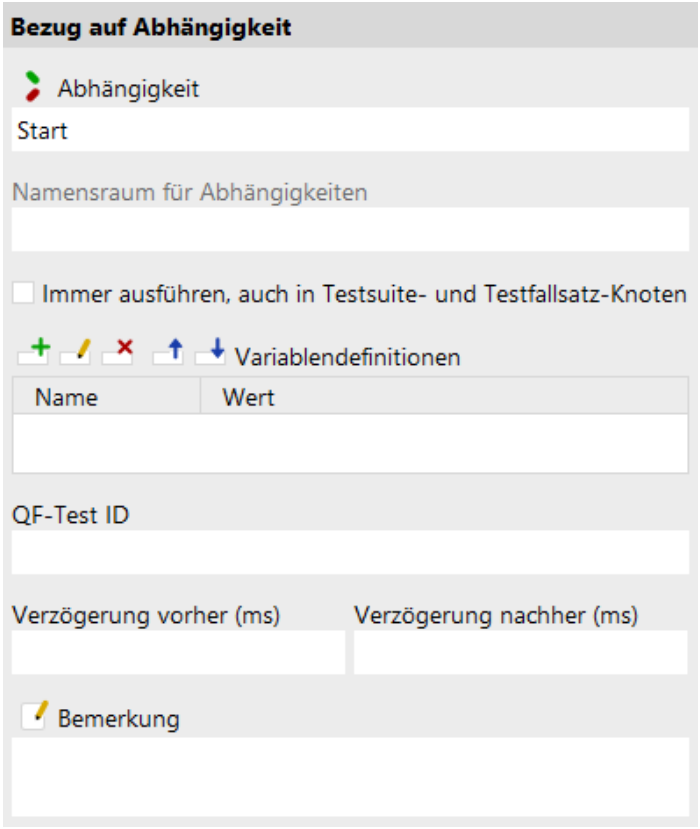


Abbildung 42.10: Bezug auf Abhängigkeit Attribute

Abhängigkeit

Der vollständige Name der Abhängigkeit(630), welcher sich aus dem Namen(681) seiner Package Parentknoten, gefolgt von seinem eigenen Namen zusammensetzt, wobei die Einzelnamen durch einen '.' verknüpft werden. Der "Abhängigkeit auswählen" Button oberhalb des Attributs öffnet einen Dialog, in dem der Zielknoten interaktiv ausgewählt werden kann. Durch Setzen des "Parameter kopieren" Auswählkästchens, können die Standardwerte für Parameter der Abhängigkeit als Parameter für den Bezug auf Abhängigkeit übernommen werden, was erneutes Eintippen erspart.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Namensraum für Abhängigkeiten

Normalerweise gibt es nur einen Stapel von Abhängigkeiten. In manchen Fällen, speziell wenn Tests für voneinander unabhängige SUT-Client gemischt werden,

kann es aber sehr nützlich sein, die Abhängigkeiten für verschiedene Teile von Tests voneinander getrennt zu halten, so dass das Auflösen von Abhängigkeiten für einen Teil nicht notwendigerweise alle Abhängigkeiten für einen anderen Teil abbaut.

Indem Sie das Attribut Namensraum für Abhängigkeiten eines Bezug auf Abhängigkeit Knotens setzen, teilen Sie QF-Test mit, dass die Abhängigkeit in dem entsprechenden Namensraum aufgelöst werden soll. Der normale Stapel von Abhängigkeiten wird in diesem Fall komplett ignoriert. Sollten bereits Abhängigkeiten in diesem Namensraum aufgelöst worden sein, wird der entsprechende Stapel als Vergleich herangezogen. Andernfalls wird ein neuer Stapel erzeugt. Ein Beispiel finden Sie in Namensräume für Abhängigkeiten(175).

Variabel: Ja

Einschränkungen: Keine

Immer ausführen, auch in Testsuite- und Testfallsatz-Knoten

In der Regel wird eine Abhängigkeit nur ausgeführt, wenn sie zu einem Testfall Knoten gehört. Abhängigkeiten in Testsuite oder Testfallsatz Knoten werden einfach von den Testfall Childknoten dieses Knotens geerbt. In einigen Fällen kann es jedoch hilfreich sein, die Abhängigkeit vorzeitig aufzulösen, z.B. wenn von der Abhängigkeit Parameter für den Testlauf bereitgestellt werden, die nötig sind, um die Bedingung(603) eines nachfolgenden Testfalls auszuwerten. Durch Aktivieren dieser Option lässt sich genau das erreichen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Diese Variablen überschreiben die Standardwerte für Parameter(633) der Abhängigkeit, die von diesem Bezug auf Abhängigkeit referenziert werden. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20).

4.2+ Mittels Rechtsklick und Auswahl von Parameterordnung zurücksetzen, können Sie die Sortierung der Parameter, wie diese in der Abhängigkeit sortiert sind, wieder herstellen.

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.3.3 Vorbereitung

Dieser Knoten verhält sich exakt wie eine Sequenz[618]. Seine Besonderheit liegt in seiner Position in einem Abhängigkeit[630], Testfallsatz[606] oder Testfall[599] Knoten.

Enthalten in: Abhängigkeit[630], Testfallsatz[606] oder Testfall[599]

Kinder: Beliebig

Ausführung: Die Variablendefinitionen[639] der Vorbereitung werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

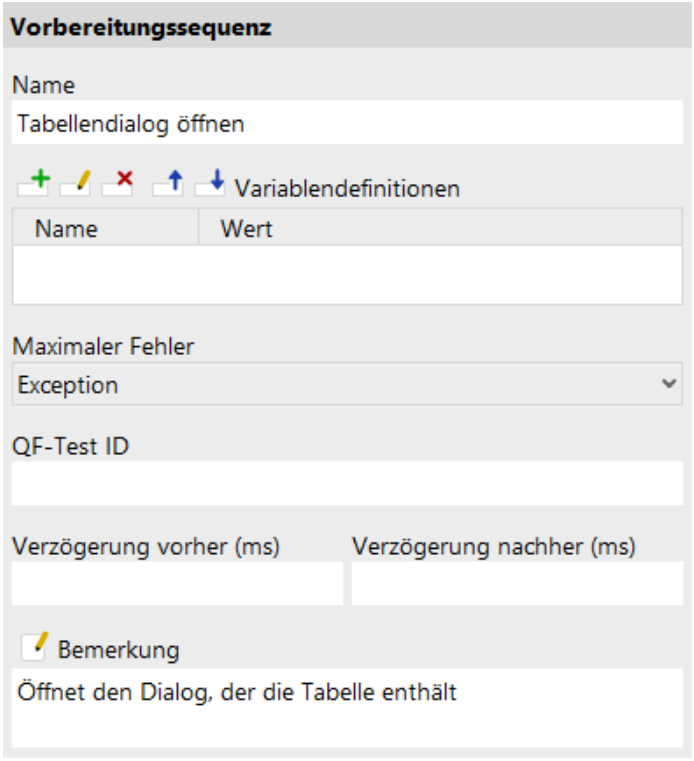


Abbildung 42.11: Vorbereitung Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20]. Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6[116].

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7[13]). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen[590]), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.3.4 Aufräumen

Dieser Knoten verhält sich exakt wie eine Sequenz[618]. Seine Besonderheit liegt in seiner Position in einem Abhängigkeit[630], Testfallsatz[606] oder Testfall[599] Knoten.

Enthalten in: Abhängigkeit[630], Testfallsatz[606] oder Testfall[599]

Kinder: Beliebig

Ausführung: Die Variablendefinitionen[642] der Aufräumen werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

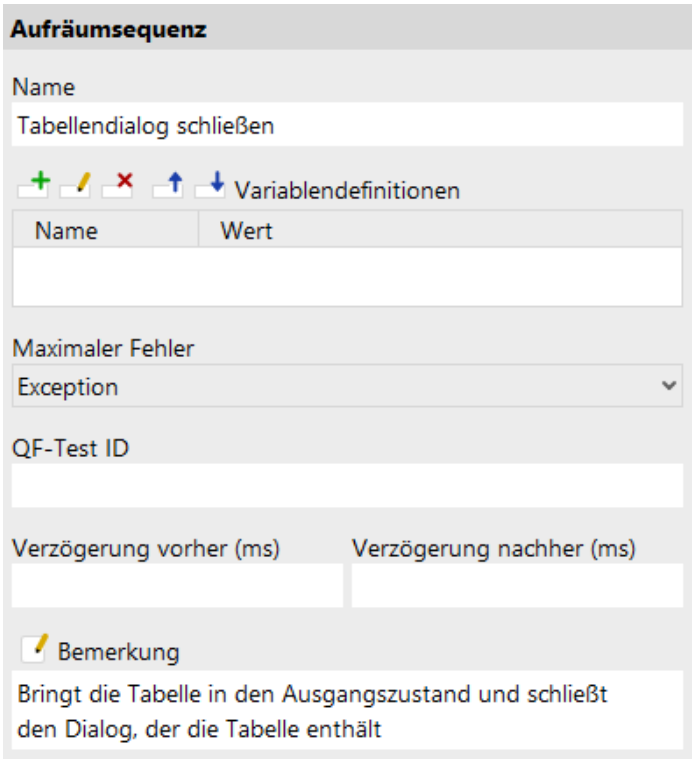


Abbildung 42.12: Aufräumen Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung

bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.3.5 Fehlerbehandlung

Dieser Knoten ist gleich zu sehen wie eine Sequenz(618) mit Ausnahme seiner besonderen Platzierung in einer Abhängigkeit(630) (vgl. Abschnitt 8.6(161)).

Enthalten in: Abhängigkeit(630)

Kinder: Beliebig

Ausführung: Die Variablendefinitionen(644) des Fehlerbehandlung Knotens werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

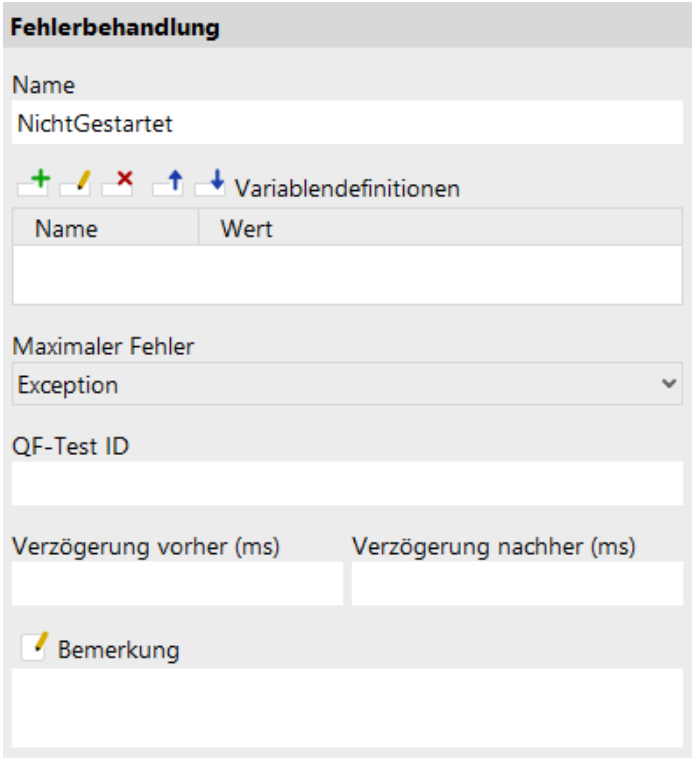


Abbildung 42.13: Fehlerbehandlung Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7[13]). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen[590]), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.4 Datentreiber

Der Ausdruck *datengetriebenes Testen* bezieht sich auf eine verbreitete Methode bei automatisierten Tests, bei der Testfälle mehrfach mit unterschiedlichen Datensätzen ausgeführt werden. Dank des flexiblen Variablenkonzepts in QF-Test gibt es keine Einschränkungen, wie diese Daten genutzt werden können. Die häufigste Verwendung ist sicherlich für Eingabewerte in Event und erwartete Ergebnisse in Check Knoten.

QF-Test's Mechanismus für datengetriebenes Testen besteht aus dem Datentreiber(646) Knoten, der benutzt wird um einen Datenkontext zur Verfügung zu stellen, und verschiedenen Arten von Daten Knoten. Aktuell gibt es den Datentabelle(650) Knoten, der Daten in Form einer Tabelle in QF-Test selbst speichert, den CSV-Datei(664) Knoten, der Daten aus einer CSV-Datei, sowie den Datenbank(653) Knoten, der Daten aus einer Datenbank und den Excel-Datei(659) Knoten, der Daten aus einer Excel-Datei liest. Es wird auch ein Erweiterungs-API für den Zugriff auf beliebige externe Daten zur Verfügung gestellt.

Weitere Informationen über die Zusammenarbeit der verschiedenen Teile des Datentreiber-Mechanismus finden Sie in Kapitel 23(319).

42.4.1 Datentreiber

Außer seiner besonderen Position im Testfallsatz(606) oder Testschritt(621) verhält sich ein Datentreiber wie eine normale Sequenz(618). Er stellt den Kontext für einen oder mehrere Daten Knoten bereit, für den sich diese während der Ausführung des Datentreibers registrieren können. Der Testfallsatz iteriert dann über die Datensätze, die von den registrierten Daten Knoten zur Verfügung gestellt werden und führt seine Childknoten aus, wie in Kapitel 23(319) beschrieben. Für diesen Zweck muss ein Datentreiber innerhalb eines Testfallsatz(606) Knotens platziert werden, zwischen die optionalen Abhängigkeit(630) und Vorbereitung(638) Knoten. Datentreiber Knoten können auch innerhalb von Testschritt(621) Knoten an erster Stelle platziert werden.

Enthalten in: Testfallsatz(606), Testschritt(621).

Kinder: Beliebig

Ausführung: Wenn ein Testfallsatz oder Testschritt ausgeführt wird, überprüft dieser, ob ein Datentreiber vorhanden ist und führt ihn aus. Der Inhalt des Datentreiber Knotens ist

nicht auf Daten Knoten beschränkt, sondern kann beliebige ausführbare Knoten enthalten, so dass jede Form von Vorbereitung realisiert werden kann, die für das Bereitstellen der Daten notwendig ist. Damit ist es auch möglich, Daten Knoten wieder zu verwenden, indem man sie in einer Prozedur(672) platziert und diese aus dem Datentreiber heraus aufruft. Jeder Daten Knoten, der in einem Datentreiber Kontext registriert ist, wird vom Testfallsatz oder Testschritt auf seine Daten hin abgefragt.

Attribute:

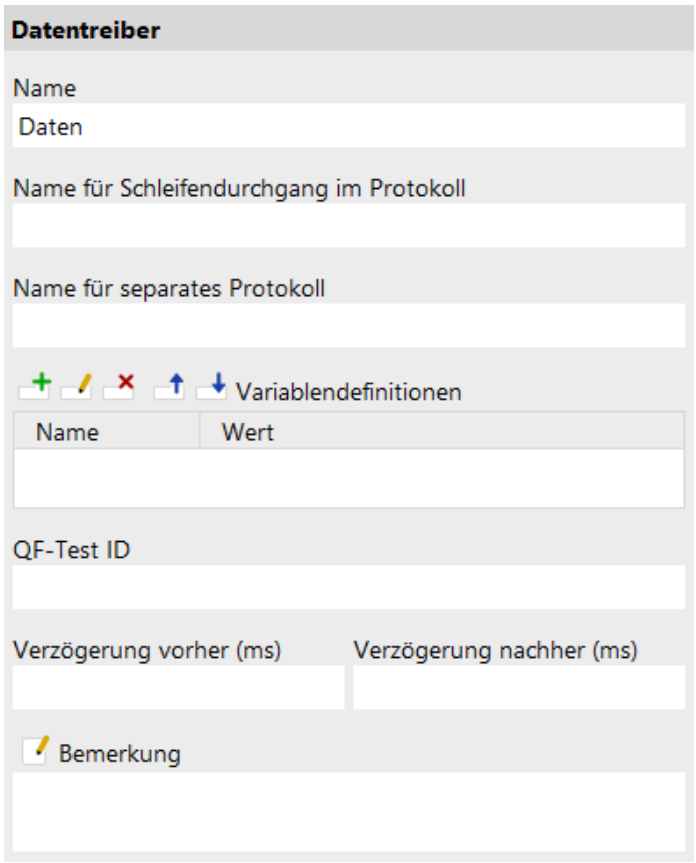


Abbildung 42.14: Datentreiber Attribute

Name

Der Name eines Datentreibers ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Art der bereitgestellten Daten aussagen.

Variabel: Nein

Einschränkungen: Keine

Name für Schleifendurchgang im Protokoll

Hier kann ein eigener Name für die Iterationsschritte definiert werden, der dann im Protokoll verwendet wird. Dieser kann Variablen der gebundenen Daten enthalten um damit das Auffinden eines spezifischen Iterationsschrittes zu erleichtern.

Variabel: Ja

Einschränkungen: Nein

Name für separates Protokoll

Mit diesem Attribut kann ein Knoten als Bruchstelle zum Abteilen eines Protokolls markiert werden. Es legt den Dateinamen für das abgeteilte Protokoll fest. Nach jeder Iteration des Datentreibers wird das zugehörige Protokoll aus dem Hauptprotokoll entfernt und als eigenständiges Protokoll gespeichert. Diese Operation ist vollständig transparent, da das Hauptprotokoll eine Referenz auf das abgeteilte Protokoll erhält und somit vollständig navigierbar bleibt. Näheres zu geteilten Protokollen finden Sie in Abschnitt 7.1.6(143).

Dieses Attribut hat keinen Effekt, wenn die Option Geteilte Protokolle erzeugen(582) deaktiviert ist oder geteilte Protokolle durch explizite Angabe von `-splitlog`(990) im Batchmodus ausgeschaltet werden.

Es ist nicht nötig für Eindeutigkeit der Dateinamen für abgeteilte Protokolle zu sorgen. Wo nötig hängt QF-Test eine Zahl an den Dateinamen an, um Konflikte zu vermeiden. Der Dateiname darf Verzeichnisse enthalten und es können - analog zur Angabe des Dateinamens für das Protokoll im Batchmodus - folgende Platzhalter in Kombination mit einem '%' oder '+' Zeichen verwendet werden:

Zeichen	Bedeutung
%	'%'-Zeichen.
+	'+'-Zeichen.
i	Die aktuelle Runid wie mit `-runid [<ID>]`(989) angegeben.
r	Die Fehlerstufe des abgeteilten Protokolls.
w	Die Anzahl der Warnungen im abgeteilten Protokoll.
e	Die Anzahl der Fehler im abgeteilten Protokoll.
x	Die Anzahl der Exceptions im abgeteilten Protokoll.
t	Der Threadindex zu dem das abgeteilte Protokoll gehört (für Tests mit parallelen Threads).
y	Das aktuelle Jahr (2 Ziffern).
Y	Das aktuelle Jahr (4 Ziffern).
M	Der aktuelle Monat (2 Ziffern).
d	Der aktuelle Tag (2 Ziffern).
h	Die aktuelle Stunde (2 Ziffern).
m	Die aktuelle Minute (2 Ziffern).
s	Die aktuelle Sekunde (2 Ziffern).

Tabelle 42.5: Platzhalter für das Attribut Name für separates Protokoll

Variabel: Ja

Einschränkungen: Keine, Zeichen die für Dateinamen nicht zulässig sind werden durch '_' ersetzt.

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung

bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.4.2 Datentabelle

Eine Datentabelle bietet eine komfortable Schnittstelle um Testdaten in Tabellen direkt in QF-Test zu speichern. Details zum Mechanismus des datengetriebenen Testens in QF-Test finden Sie in Kapitel 23[319].

Enthalten in: Beliebig

Kinder: Keine

Ausführung: Die Datentabelle expandiert Variablen in der Tabelle entsprechend der Option Werte von Variablen beim Binden sofort expandieren[592]. Jede Zeile wird individuell von links nach rechts expandiert, so dass eine Zelle - innerhalb derselben Zeile - auf Variablen verweisen kann, die in einer Spalte weiter links gebunden wurden. Dann registriert sich die Datentabelle im Kontext eines Datentreibers. Zusätzlich wird eine Propertygruppe mit dem Namen des Knotens und den Variablen `size` und `totalsize` angelegt. `size` beinhaltet die Anzahl der Datensätze mit Rücksicht auf Iterationsintervalle. `totalsize` beinhaltet die gesamte Anzahl an Datensätzen der Datenquelle ohne Rücksicht auf Iterationsintervalle. Nachdem alle Daten Knoten angemeldet wurden,

fragt der zugehörige Testfallsatz(606) die Datensätze der Datentabelle ab, um darüber zu iterieren. Wenn kein Datentreiber Kontext verfügbar ist, wird die Propertygruppe um alle Variablen des Knotens erweitert.

Attribute:

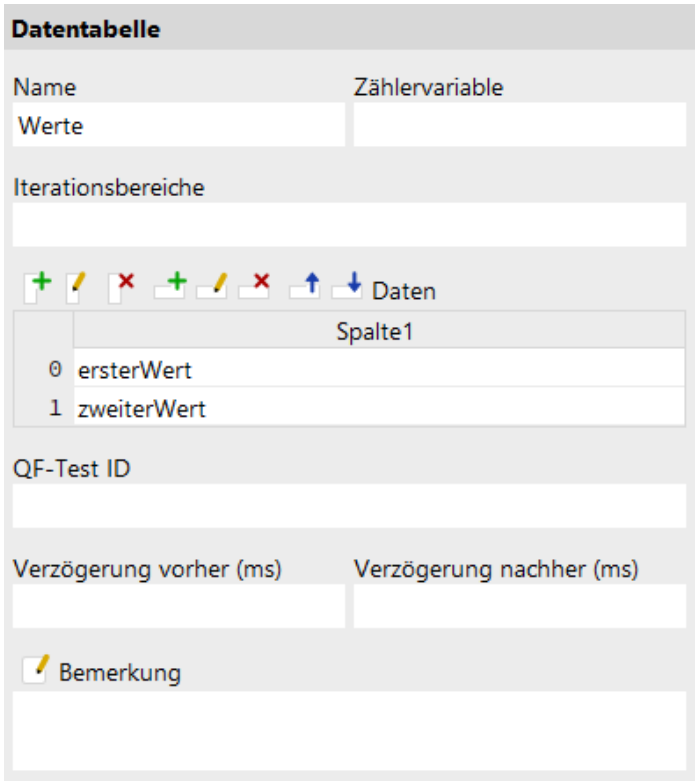


Abbildung 42.15: Datentabelle Attribute

Name

Der Name eines Daten Knotens ist ein Pflichtattribut. Er wird benutzt, um Daten Knoten im selben Datentreiber Kontext zu unterscheiden. Ein Break(691) Knoten, der während des datengetriebenen Tests ausgeführt wird, kann durch Angabe dieses Namens in seinem Attribut QF-Test ID der Schleife(692) gezielt die Schleife dieser Datentabelle abbrechen.

Variabel: Ja

Einschränkungen: Keine

Zählervariable

Der Name der Variablen, an die der Index des Schleifendurchgangs gebunden wird.

Variabel: Ja

Einschränkungen: Keine

Iterationsbereiche

Ein Optionaler Satz von zu nutzenden Indizes oder Bereichen für die Daten. Dies ist besonders während der Testentwicklung hilfreich, um nur über einen einzelnen Index oder einen Teil der Daten zu iterieren.

Bereiche werden durch ',' getrennt. Jeder Bereich ist entweder in einzelner Index, oder ein geschlossenes Intervall der Form 'von-bis' oder 'von:bis' wobei die Angabe von 'bis' optional ist. Indizes oder Bereiche können mehrfach, überlappend oder in absteigender Richtung angegeben werden. Indizes sind 0-basiert, ein negativer Index wird vom Ende her gezählt, wobei -1 dem letzten Eintrag entspricht. Eine ungültige Syntax oder ein Index außerhalb des vorhandenen Datenbereichs führen zu einer `BadRangeException`(962).

Die folgende Tabelle zeigt einige Beispiele für Iterationsbereiche und die daraus resultierenden Indizes.

Iterationsbereiche	Daraus resultierende Indizes
0	[0]
-2, -1	[18, 19]
1-2,4:5	[1, 2, 4, 5]
18:,-3-	[18, 19, 17, 18, 19]
3-2,16:15	[3, 2, 16, 15]

Tabelle 42.6: Beispiele für Iterationsbereiche

Hinweis

Der für die Zählervariable gebunde Wert entspricht dem angegebenen Index im Iterationsbereich, nicht der Zahl der tatsächlich durchgeführten Iterationen. Wenn Sie z.B. '2' angeben, führt dies zu einer einzigen Iteration mit der Zählervariable '2', nicht '0'.

Variabel: Ja

Einschränkungen: Gültige Syntax und Indexwerte

Daten

Hier werden die aktuellen Testdaten definiert. Jede Tabellenspalte steht hierbei für eine Variable, deren Name im Spaltenkopf spezifiziert wird. Jede Zeile ist ein Datensatz, ein Wert je Variable. So bestimmt die Zeilenzahl die Anzahl der Durchläufe in der datengetriebenen Schleife. Um Ihre Daten eingeben zu können, müssen Sie als erstes Spalten zur Tabelle hinzufügen und somit die

Variablen definieren. Dann können mittels neuer Zeilen die Werte festgelegt werden. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20).

Variabel: Ja, sogar die Spaltentitel

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.4.3 Datenbank

Ein Datenbank Knoten wird benutzt, um externe Daten aus einer Datenbank zu laden. Damit QF-Test auf die Datenbank zugreifen kann, muss die jar-Datei mit den Klassen des Datenbanktreibers im QF-Test Klassenpfad zu finden sein. Üblicherweise wird die Datei ins `qftest` Pluginverzeichnis (Abschnitt 50.2[1029]) kopiert. Um mehr Informationen über den Aufbau einer Datenbankverbindung zu erlangen, fragen Sie einen Entwickler oder besuchen Sie www.connectionstrings.com.

Weitere Details zum Datentreiber-Mechanismus finden Sie in Kapitel 23[319].

Enthalten in: Beliebig

Kinder: Keine

Ausführung: Der Datenbank Knoten lädt die Daten aus der Datenbank und expandiert Variablen entsprechend der Option Werte von Variablen beim Binden sofort expandieren[592]. Jede Zeile wird individuell von links nach rechts expandiert, so dass eine Zelle - innerhalb derselben Zeile - auf Variablen verweisen kann, die in einer Spalte weiter links gebunden wurden. Dann registriert sich der Datenbank im Kontext eines Datentreibers. Für jedes Feld im select-Ausdruck wird eine Variable gleichen Namens definiert. Die Schreibweise (Groß/Klein) kann jedoch vom Datenbanktreiber abhängen, z.B. nur Großbuchstaben bei Oracle. Zusätzlich wird eine Propertygruppe mit dem Namen des Knotens und den Variablen `size` und `totalsize` angelegt. `size` beinhaltet die Anzahl der Datensätze mit Rücksicht auf Iterationsintervalle. `totalsize` beinhaltet die gesamte Anzahl an Datensätzen der Datenquelle ohne Rücksicht auf Iterationsintervalle. Nachdem alle Daten Knoten angemeldet wurden, fragt der zugehörige Testfallsatz[606] die Datensätze der Datenbank ab, um darüber zu iterieren. Wenn kein Datentreiber Kontext verfügbar ist, wird die Propertygruppe um alle Variablen des Knotens erweitert.

Attribute:

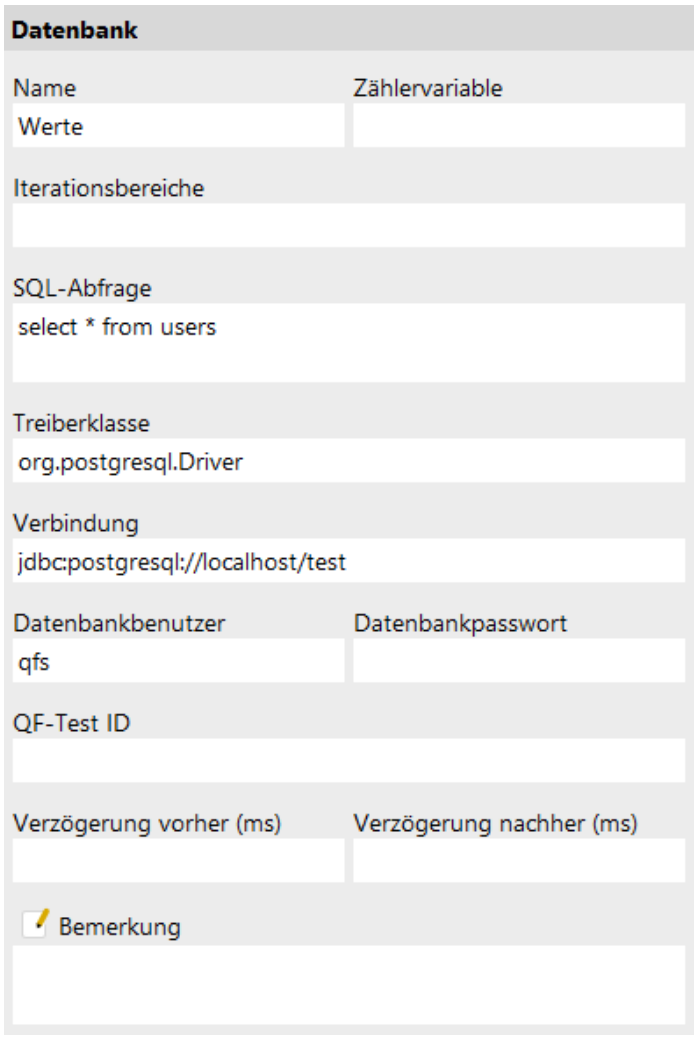


Abbildung 42.16: Datenbank Attribute

Name

Der Name eines Daten Knotens ist ein Pflichtattribut. Er wird benutzt, um Daten Knoten im selben Datentreiber Kontext zu unterscheiden. Ein Break(691) Knoten, der während des datengetriebenen Tests ausgeführt wird, kann durch Angabe dieses Namens in seinem Attribut QF-Test ID der Schleife(692) gezielt die Schleife dieser Datentabelle abbrechen.

Variabel: Ja

Einschränkungen: Keine

Zählervariable

Der Name der Variablen, an die der Index des Schleifendurchgangs gebunden wird.

Variabel: Ja

Einschränkungen: Keine

Iterationsbereiche

Ein Optionaler Satz von zu nutzenden Indizes oder Bereichen für die Daten. Dies ist besonders während der Testentwicklung hilfreich, um nur über einen einzelnen Index oder einen Teil der Daten zu iterieren.

Bereiche werden durch ',' getrennt. Jeder Bereich ist entweder in einzelner Index, oder ein geschlossenes Intervall der Form 'von-bis' oder 'von:bis' wobei die Angabe von 'bis' optional ist. Indizes oder Bereiche können mehrfach, überlappend oder in absteigender Richtung angegeben werden. Indizes sind 0-basiert, ein negativer Index wird vom Ende her gezählt, wobei -1 dem letzten Eintrag entspricht. Eine ungültige Syntax oder ein Index außerhalb des vorhandenen Datenbereichs führen zu einer `BadRangeException`(962).

Die folgende Tabelle zeigt einige Beispiele für Iterationsbereiche und die daraus resultierenden Indizes.

Iterationsbereiche	Daraus resultierende Indizes
0	[0]
-2, -1	[18, 19]
1-2,4:5	[1, 2, 4, 5]
18:,-3-	[18, 19, 17, 18, 19]
3-2,16:15	[3, 2, 16, 15]

Tabelle 42.7: Beispiele für Iterationsbereiche

Hinweis

Der für die Zählervariable gebunde Wert entspricht dem angegebenen Index im Iterationsbereich, nicht der Zahl der tatsächlich durchgeführten Iterationen. Wenn Sie z.B. '2' angeben, führt dies zu einer einzigen Iteration mit der Zählervariable '2', nicht '0'.

Variabel: Ja

Einschränkungen: Gültige Syntax und Indexwerte

SQL-Abfrage

Die SQL-Abfrage, die ausgeführt wird, um die gewünschten Testdaten zu erhalten. Dies sollte ein *select* Ausdruck sein. Für jedes Feld im select-Ausdruck wird eine Variable gleichen Namens definiert. Die Schreibweise (Groß/Klein)

kann jedoch vom Datenbanktreiber abhängen, z.B. nur Großbuchstaben bei Oracle.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Treiberklasse

Die Klasse des Datenbanktreibers.

Hinweis

Die jar-Datei mit den Klassen des Datenbanktreibers muss vor dem Start von QF-Test ins `qftest` Pluginverzeichnis kopiert werden.

Hier eine Liste der gebräuchlichsten Datenbanktreiber:

Datenbank	Klassenname für den JDBC-Treiber
Borland Interbase	`interbase.interclient.Driver`
DB2	`com.ibm.db2.jcc.DB2Driver`
Informix	`com.informix.jdbc.IfxDriver`
IDS Server	`ids.sql.IDSDriver`
MS SQL Server 2000	`com.microsoft.jdbc.sqlserver.SQLServerDriver`
MS SQL Server 2005	`com.microsoft.sqlserver.jdbc.SQLServerDriver`
mSQL	`COM.imaginary.sql.msql.MsqlDriver`
MySQL	`com.mysql.jdbc.Driver`
Oracle	`oracle.jdbc.driver.OracleDriver`
Pointbase	`com.pointbase.jdbc.jdbcUniversalDriver`
PostgreSQL	`org.postgresql.Driver`
Standard Treiber	`sun.jdbc.odbc.JdbcOdbcDriver`
Sybase	`com.sybase.jdbc2.jdbc.SybDriver`
SQLite	`org.sqlite.JDBC`

Tabelle 42.8: JDBC Treiberklassen

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Verbindung

Der *Connection String* für die Datenbank. Dieser ist meist von der Form `jdbc:`*`datenbankname`*`://`*`datenbankserver`*`/`*`datenbankname`*.

Hier eine Liste der gebräuchlichsten Connection Strings:

Datenbank	Beispiel
Derby	`jdbc:derby:net://`*`datenbankserver`*`:`*`port`*`/`
IBM DB2	`jdbc:db2://`*`datenbank`*
HSQLB	`jdbc:hsqldb:file:`*`datenbank`*
Interbase	`jdbc:interbase://`*`datenbankserver`*`/`*`datenbank`*`.gdb`
MS SQL Server 2000	`jdbc:microsoft:sqlserver://`*`databaseserver`*`:`*`port`*`;DatabaseName=`*`database`*`;`
MS SQL Server 2005	`jdbc:sqlserver://`*`databaseserver`*`:`*`port`*`;DatabaseName=`*`database`*`;`
MySQL	`jdbc:mysql://`*`datenbankserver`*`/`*`datenbank`*
PostgreSQL	`jdbc:postgresql://`*`datenbankserver`*`/`*`datenbank`*
ODBC-Datenquellen	`jdbc:odbc:`*`datenbank`*
Oracle Thin	`jdbc:oracle:thin:@`*`datenbankserver`*`:`*`port`*`:`*`datenbank`*
Sybase	`jdbc:sybase:tds:`*`datenbankserver`*`:`*`port`*`/`*`datenbank`*
SQLite	`jdbc:sqlite:`*`sqlite_database_file_path`*

Tabelle 42.9: Datenbank Verbindungen

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Datenbankbenutzer

Der Benutzer für die Datenbank. Sollte Ihre Datenbank keinen Benutzer benötigen, lassen Sie dieses Feld einfach leer.

Variabel: Ja

Einschränkungen: Keine

Datenbankpasswort

Das Passwort für den Benutzer der Datenbank. Sollte Ihre Datenbank keinen Benutzer oder kein Passwort benötigen, lassen Sie dieses Feld einfach leer. Um das Passwort nach Eingabe in dieses Feld zu verschlüsseln, wählen Sie nach einem Rechts-Klick Text verschlüsseln aus dem resultierenden Popupmenü. Geben Sie auf jeden Fall vor der Verschlüsselung einen Salt in der Option Salt für Verschlüsselung von Kennwörtern[532] an.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.4.4 Excel-Datei

Ein Excel-Datei Knoten wird benutzt, um externe Daten aus einer Excel-Datei zu laden. In der ersten Zeile müssen die Namen der Variablen definiert werden und in den weiteren Zeilen die zu bindenden Werte, wobei jede Zeile einem Testdatensatz entspricht.

Hinweis

Die korrekte Darstellung der Zelleninhalte vom Typ "Datum", "Zeit" oder "Währung" kann nicht immer sicher gestellt werden, da Excel den Wert und das Format der Zelle getrennt voneinander speichert und die verwendete Java-Bibliothek nicht alle Möglichkeiten unterstützt. Dennoch sollte der Excel-Datei Knoten weitestgehend korrekt arbeiten. Sollte es zu Problemen mit diesen Formaten kommen, ändern Sie bitte in der Excel-Datei den

Typ der betroffenen Zellen auf ”Text”. In QF-Test werden nach dem Einlesen sowieso alle Werte als Strings behandelt.

Weitere Details zum Datentreiber-Mechanismus finden Sie in Kapitel 23(319).

Enthalten in: Beliebig

Kinder: Keine

Ausführung: Der Excel-Datei Knoten liest die Daten aus der Excel-Datei und expandiert Variablen entsprechend der Option Werte von Variablen beim Binden sofort expandieren(592). Jede Zeile wird individuell von links nach rechts expandiert, so dass eine Zelle - innerhalb derselben Zeile - auf Variablen verweisen kann, die in einer Spalte weiter links gebunden wurden. Dann registriert sich der Excel-Datei im Kontext eines Datentreibers. Zusätzlich wird eine Propertygruppe mit dem Namen des Knotens und den Variablen `size` und `totalsize` angelegt. `size` beinhaltet die Anzahl der Datensätze mit Rücksicht auf Iterationsintervalle. `totalsize` beinhaltet die gesamte Anzahl an Datensätzen der Datenquelle ohne Rücksicht auf Iterationsintervalle. Nachdem alle Daten Knoten angemeldet wurden, fragt der zugehörige Testfallsatz(606) die Datensätze der Excel-Datei ab, um darüber zu iterieren. Wenn kein Datentreiber Kontext verfügbar ist, wird die Propertygruppe um alle Variablen des Knotens erweitert.

Attribute:

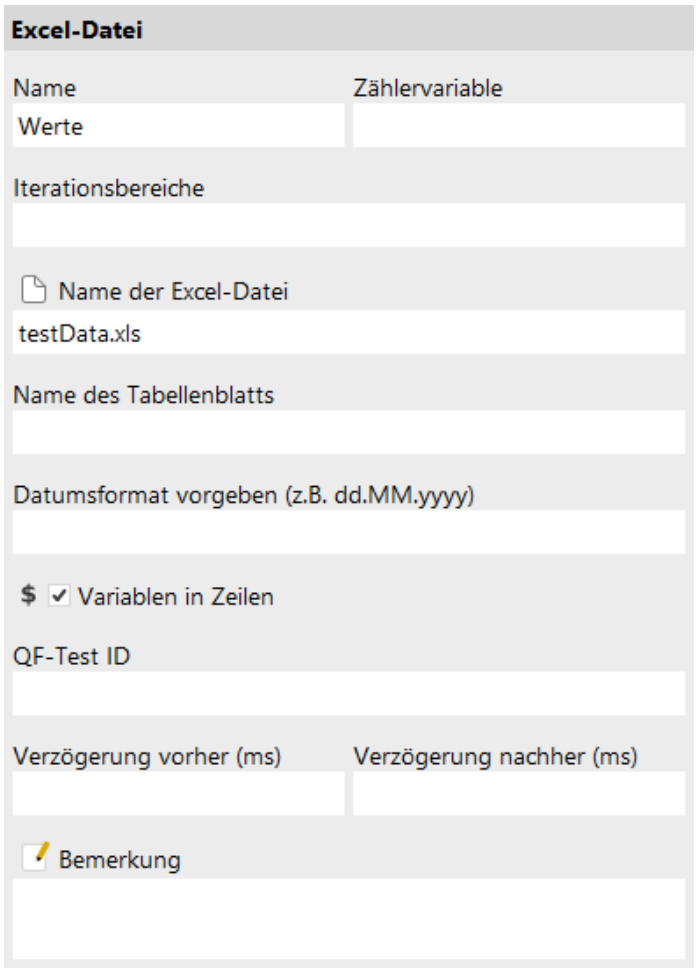


Abbildung 42.17: Excel-Datei Attribute

Name

Der Name eines Daten Knotens ist ein Pflichtattribut. Er wird benutzt, um Daten Knoten im selben Datentreiber Kontext zu unterscheiden. Ein Break(691) Knoten, der während des datengetriebenen Tests ausgeführt wird, kann durch Angabe dieses Namens in seinem Attribut QF-Test ID der Schleife(692) gezielt die Schleife dieser Datentabelle abbrechen.

Variabel: Ja

Einschränkungen: Keine

Zählervariable

Der Name der Variablen, an die der Index des Schleifendurchgangs gebunden wird.

Variabel: Ja

Einschränkungen: Keine

Iterationsbereiche

Ein Optionaler Satz von zu nutzenden Indizes oder Bereichen für die Daten. Dies ist besonders während der Testentwicklung hilfreich, um nur über einen einzelnen Index oder einen Teil der Daten zu iterieren.

Bereiche werden durch ',' getrennt. Jeder Bereich ist entweder in einzelner Index, oder ein geschlossenes Intervall der Form 'von-bis' oder 'von:bis' wobei die Angabe von 'bis' optional ist. Indizes oder Bereiche können mehrfach, überlappend oder in absteigender Richtung angegeben werden. Indizes sind 0-basiert, ein negativer Index wird vom Ende her gezählt, wobei -1 dem letzten Eintrag entspricht. Eine ungültige Syntax oder ein Index außerhalb des vorhandenen Datenbereichs führen zu einer `BadRangeException`(962).

Die folgende Tabelle zeigt einige Beispiele für Iterationsbereiche und die daraus resultierenden Indizes.

Iterationsbereiche	Daraus resultierende Indizes
0	[0]
-2, -1	[18, 19]
1-2,4:5	[1, 2, 4, 5]
18:,-3-	[18, 19, 17, 18, 19]
3-2,16:15	[3, 2, 16, 15]

Tabelle 42.10: Beispiele für Iterationsbereiche

Hinweis

Der für die Zählervariable gebunde Wert entspricht dem angegebenen Index im Iterationsbereich, nicht der Zahl der tatsächlich durchgeführten Iterationen. Wenn Sie z.B. '2' angeben, führt dies zu einer einzigen Iteration mit der Zählervariable '2', nicht '0'.

Variabel: Ja

Einschränkungen: Gültige Syntax und Indexwerte

Name der Excel-Datei

Der Name der Excel-Datei aus der die Testdaten gelesen werden sollen. Relative Pfadangaben werden relativ zum Verzeichnis der Testsuite aufgelöst.

Der Knopf über dem Attribut öffnet einen Dialog in dem die Excel-Datei interaktiv ausgewählt werden kann. Den Dialog erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Name des Tabellenblatts

Der Name des Tabellenblatts aus dem die Testdaten gelesen werden sollen.

Variabel: Ja

Einschränkungen: Keine

Datumsformat vorgeben (z.B. dd.MM.yyyy)

In diesem Textfeld können Sie ein Datumsformat vorgeben, welches für alle aus der Excel-Datei geladenen Zellen vom Typ ”Datum” verwendet werden soll. Dieses Format muss wie für die Java-Klasse `SimpleDateFormat` spezifiziert sein, d.h. ’d’ für Tag, ’M’ für Monat und ’y’ für Jahr. Mit ’dd’ werden zwei Stellen für den Tag reserviert, analog dazu gibt es ’MM’, ’yy’ oder ’yyyy’.

Variabel: Ja

Einschränkungen: Keine

Variablen in Zeilen

Wenn dieses Attribute angehakt ist, werden die Namen der Variablen aus der ersten Zeile der Excel-Datei ausgelesen, sonst aus der ersten Spalte.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder

Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.4.5 CSV-Datei

Eine CSV-Datei wird benutzt, um externe Daten aus einer Datei zu laden und für datengetriebene Tests bereit zu stellen. *CSV* steht für *Comma-separated Values* - also für *durch Komma getrennte Werte*, welches ein Standardformat für Textdateien darstellt. Hierbei steht jede Zeile in der Datei für einen Datensatz, dessen Werte durch ein Trennzeichen separiert werden. Als Trennzeichen wird oft, aber nicht ausschließlich, ein Komma (',') verwendet. Bei der Benutzung in einem CSV-Datei Knoten muss die erste Zeile der CSV-Datei die Namen der Variablen enthalten, die gebunden werden sollen. In den weiteren Zeilen folgen dann die Werte für die einzelnen Iterationsschritte.

Leider gibt es bei CSV-Dateien uneinheitliche Definitionen für Spezialfälle wie das Schützen von Sonderzeichen (quoting), die Behandlung von Leerzeichen oder Trennzeichen in Werten oder mehrzeilige Werte. Aktuell existieren zwei de-facto Standards. Einen benutzt Microsoft Excel und einen der Rest der Welt. QF-Test unterstützt beide. Weitere Details zum Datentreiber-Mechanismus finden Sie in Kapitel 23[319].

Enthalten in: Beliebig

Kinder: Keine

Ausführung: Der CSV-Datei Knoten liest die Daten aus der CSV-Datei und expandiert Variablen entsprechend der Option Werte von Variablen beim Binden sofort expandieren[592]. Jede Zeile wird individuell von links nach rechts expandiert, so dass eine Zelle - innerhalb derselben Zeile - auf Variablen verweisen kann, die in einer Spalte weiter links gebunden wurden. Dann registriert sich der CSV-Datei im Kontext eines Datentreibers. Zusätzlich wird eine Propertygruppe mit dem Namen des Knotens

und den Variablen `size` und `totalsize` angelegt. `size` beinhaltet die Anzahl der Datensätze mit Rücksicht auf Iterationsintervalle. `totalsize` beinhaltet die gesamte Anzahl an Datensätzen der Datenquelle ohne Rücksicht auf Iterationsintervalle. Nachdem alle Daten Knoten angemeldet wurden, fragt der zugehörige Testfallsatz(606) die Datensätze der CSV-Datei ab, um darüber zu iterieren. Wenn kein Datentreiber Kontext verfügbar ist, wird die Propertygruppe um alle Variablen des Knotens erweitert.

Attribute:

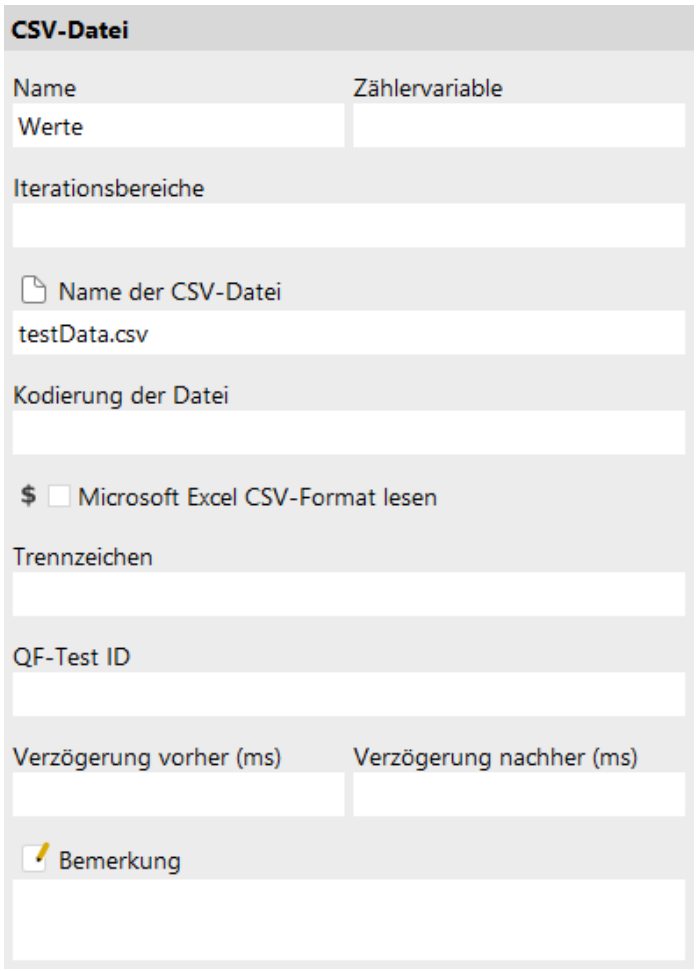


Abbildung 42.18: CSV-Datei Attribute

Name

Der Name eines Daten Knotens ist ein Pflichtattribut. Er wird benutzt, um Daten Knoten im selben Datentreiber Kontext zu unterscheiden. Ein Break(691) Knoten, der während des datengetriebenen Tests ausgeführt wird, kann durch Angabe dieses

Namens in seinem Attribut QF-Test ID der Schleife[692] gezielt die Schleife dieser Datentabelle abbrechen.

Variabel: Ja

Einschränkungen: Keine

Zählervariable

Der Name der Variablen, an die der Index des Schleifendurchgangs gebunden wird.

Variabel: Ja

Einschränkungen: Keine

Iterationsbereiche

Ein Optionaler Satz von zu nutzenden Indizes oder Bereichen für die Daten. Dies ist besonders während der Testentwicklung hilfreich, um nur über einen einzelnen Index oder einen Teil der Daten zu iterieren.

Bereiche werden durch ',' getrennt. Jeder Bereich ist entweder in einzelner Index, oder ein geschlossenes Intervall der Form 'von-bis' oder 'von:bis' wobei die Angabe von 'bis' optional ist. Indizes oder Bereiche können mehrfach, überlappend oder in absteigender Richtung angegeben werden. Indizes sind 0-basiert, ein negativer Index wird vom Ende her gezählt, wobei -1 dem letzten Eintrag entspricht. Eine ungültige Syntax oder ein Index außerhalb des vorhandenen Datenbereichs führen zu einer `BadRangeException`[962].

Die folgende Tabelle zeigt einige Beispiele für Iterationsbereiche und die daraus resultierenden Indizes.

Iterationsbereiche	Daraus resultierende Indizes
0	[0]
-2, -1	[18, 19]
1-2,4:5	[1, 2, 4, 5]
18:,-3-	[18, 19, 17, 18, 19]
3-2,16:15	[3, 2, 16, 15]

Tabelle 42.11: Beispiele für Iterationsbereiche

Hinweis

Der für die Zählervariable gebunde Wert entspricht dem angegebenen Index im Iterationsbereich, nicht der Zahl der tatsächlich durchgeführten Iterationen. Wenn Sie z.B. '2' angeben, führt dies zu einer einzigen Iteration mit der Zählervariable '2', nicht '0'.

Variabel: Ja

Einschränkungen: Gültige Syntax und Indexwerte

Name der CSV-Datei

Der Name der CSV-Datei aus der die Testdaten gelesen werden sollen. Relative Pfadangaben werden relativ zum Verzeichnis der Testsuite aufgelöst.

Der Knopf über dem Attribut öffnet einen Dialog in dem die CSV-Datei interaktiv ausgewählt werden kann. Den Dialog erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Kodierung der Datei

Eine optionale Kodierung für die CSV-Datei, z.B "UTF-8". Ist keine Kodierung angeben, wird die Datei mit der Standard-Kodierung der Java-VM gelesen.

Variabel: Ja

Einschränkungen: Die Kodierung muss von der Java-VM unterstützt werden.

Microsoft Excel CSV-Format lesen

Wenn diese Option gesetzt ist, versucht QF-Test die CSV-Datei gemäß dem von Microsoft Excel verwendeten Format einzulesen.

Variabel: Ja

Einschränkungen: Keine

Trennzeichen

Mit diesem Attribut können sie das Zeichen festlegen, welches zur Trennung der Datenwerte in der CSV-Datei verwendet wird. Wenn kein Trennzeichen definiert ist, wird das Komma (',') als Standardwert genutzt. Wenn Microsoft Excel CSV-Format lesen(667) gesetzt ist, wird dieses Attribut ignoriert.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.4.6 Datenschleife

Bei einem Datenschleife Knoten handelt es sich um eine einfache Schleife, bei der eine einzelne Variable als Schleifenzähler gebunden wird. So können Testfälle[599] mehrfach ausgeführt werden, indem sie in einen Testfallsatz[606] mit einem Datentreiber[646] und einer Datenschleife gelegt werden.

Enthalten in: Beliebig

Kinder: Keine

Ausführung: Bei ihrer Ausführung registriert sich die Datenschleife lediglich im Kontext eines Datentreibers. Die Zählvariable wird als Variable gesetzt. Zusätzlich wird eine Propertygruppe mit dem Namen des Knotens und den Variablen `size` und `totalsize` angelegt. `size` beinhaltet die Anzahl der Datensätze mit Rücksicht auf Iterationsintervalle. `totalsize` beinhaltet die gesamte Anzahl an Datensätzen der Datenquelle ohne Rücksicht auf Iterationsintervalle. Nachdem alle Daten Knoten angemeldet wurden, fragt der zugehörige Testfallsatz[606] die Datensätze der Datenschleife ab, um darüber zu iterieren. Wenn kein Datentreiber Kontext verfügbar ist, wird die Propertygruppe um alle Variablen des Knotens erweitert.

Attribute:

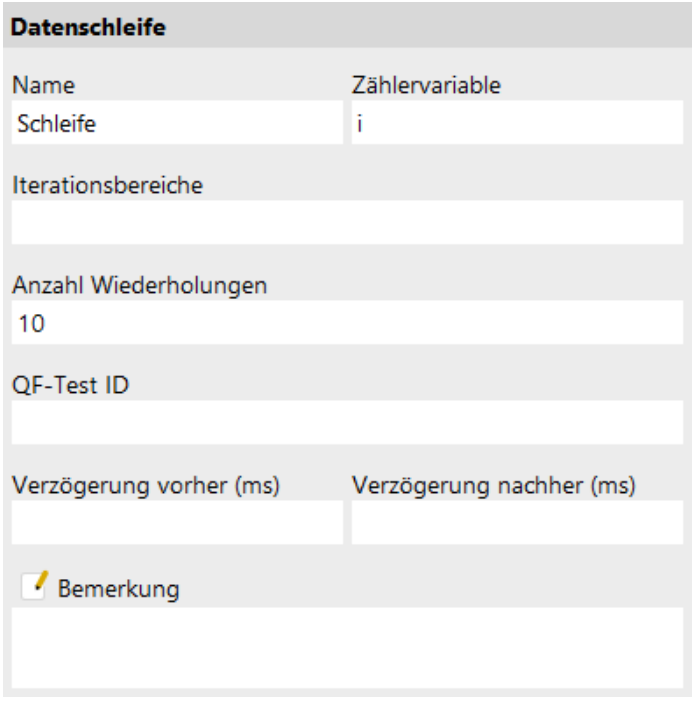


Abbildung 42.19: Datenschleife Attribute

Name

Der Name eines Daten Knotens ist ein Pflichtattribut. Er wird benutzt, um Daten Knoten im selben Datentreiber Kontext zu unterscheiden. Ein Break(691) Knoten, der während des datengetriebenen Tests ausgeführt wird, kann durch Angabe dieses Namens in seinem Attribut QF-Test ID der Schleife(692) gezielt die Schleife dieser Datentabelle abbrechen.

Variabel: Ja

Einschränkungen: Keine

Zählervariable

Der Name der Variablen, an die der Index des Schleifendurchgangs gebunden wird.

Variabel: Ja

Einschränkungen: Keine

Iterationsbereiche

Ein Optionaler Satz von zu nutzenden Indizes oder Bereichen für die Daten. Dies ist besonders während der Testentwicklung hilfreich, um nur über einen einzelnen Index oder einen Teil der Daten zu iterieren.

Bereiche werden durch ',' getrennt. Jeder Bereich ist entweder in einzelner Index, oder ein geschlossenes Intervall der Form 'von-bis' oder 'von:bis' wobei die Angabe von 'bis' optional ist. Indizes oder Bereiche können mehrfach, überlappend oder in absteigender Richtung angegeben werden. Indizes sind 0-basiert, ein negativer Index wird vom Ende her gezählt, wobei -1 dem letzten Eintrag entspricht. Eine ungültige Syntax oder ein Index außerhalb des vorhandenen Datenbereichs führen zu einer `BadRangeException`(962).

Die folgende Tabelle zeigt einige Beispiele für Iterationsbereiche und die daraus resultierenden Indizes.

Iterationsbereiche	Daraus resultierende Indizes
0	[0]
-2, -1	[18, 19]
1-2,4:5	[1, 2, 4, 5]
18:,-3-	[18, 19, 17, 18, 19]
3-2,16:15	[3, 2, 16, 15]

Tabelle 42.12: Beispiele für Iterationsbereiche

Hinweis

Der für die Zählervariable gebunde Wert entspricht dem angegebenen Index im Iterationsbereich, nicht der Zahl der tatsächlich durchgeführten Iterationen. Wenn Sie z.B. '2' angeben, führt dies zu einer einzigen Iteration mit der Zählervariable '2', nicht '0'.

Variabel: Ja

Einschränkungen: Gültige Syntax und Indexwerte

Anzahl Wiederholungen

Dieser Wert legt die Anzahl der Durchgänge für die Ausführung fest.

Dieses Attribut ist prädestiniert für den Einsatz von Variablen. Wenn Sie die Anzahl Wiederholungen z.B. auf `$(count)` setzen, können Sie den Wert für die Variable `count` in den Variablen der Suite (vgl. Kapitel 6(116)) auf 1 oder 2 setzen. Das vereinfacht das Erstellen und Ausprobieren der Testsuite.

Wenn Sie den Test dann ernsthaft ausführen, z.B. automatisch mittels `qftest -batch`, können Sie durch Angabe von `-variable count=100` auf der Kommandozeile den Wert für `count` nach Belieben variieren.

Variabel: Ja

Einschränkungen: >0

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.5 Prozeduren

Bei automatischen GUI Tests müssen immer wieder ähnliche Abläufe mit kleinen Variationen ausgeführt werden. Um die Komplexität einer Testsuite zu reduzieren und besser auf Änderungen reagieren zu können, ist es sinnvoll, wiederverwendbare Teile in Prozeduren[672] auszulagern.

Eine Prozedur können Sie von jedem anderen Punkt der Testsuite und sogar aus anderen Testsuiten heraus aufrufen (vgl. Abschnitt 26.1[359]). Dabei können Sie Parameter in Form von Variablendefinitionen übergeben (vgl. Kapitel 6[116]).

42.5.1 Prozedur

Eine Prozedur ist eine Sequenz[618], die mittels eines Prozeduraufrufs[675] von einer beliebigen anderen Stelle aus aufgerufen werden kann.

In einer Prozedur lassen sich häufig wiederkehrende Bestandteile einer Testsuite kapseln und über die Argumente des Prozeduraufrufs in Grenzen variieren. Dadurch müssen diese Teile nur an einer Stelle gepflegt und an mögliche Änderungen des SUT angepasst werden.

Die Parameter, die die Prozedur erwartet, werden nicht explizit angegeben, sondern ergeben sich durch die Variablenreferenzen der Kinder der Prozedur. Um den Überblick zu behalten, sollten Sie es sich zur Gewohnheit machen, die Parameter in der Bemerkung[674] aufzuführen.

Eine Prozedur kann mit Hilfe eines Return[678] Knotens einen Wert an den aufrufenden Knoten zurückgeben. Ohne einen solchen Return Knoten wird implizit der leere String zurück geliefert.

Enthalten in: Package[680], Prozeduren[682]

Kinder: Beliebig

Ausführung: Die Variablen der Prozedur werden als ”Fallback” gebunden, d.h. diese Werte werden verwendet, falls eine Variable keine andere Definition hat. Nachdem die Childknoten der Prozedur ausgeführt wurden, werden die Fallbacks wieder gelöscht.

Attribute:

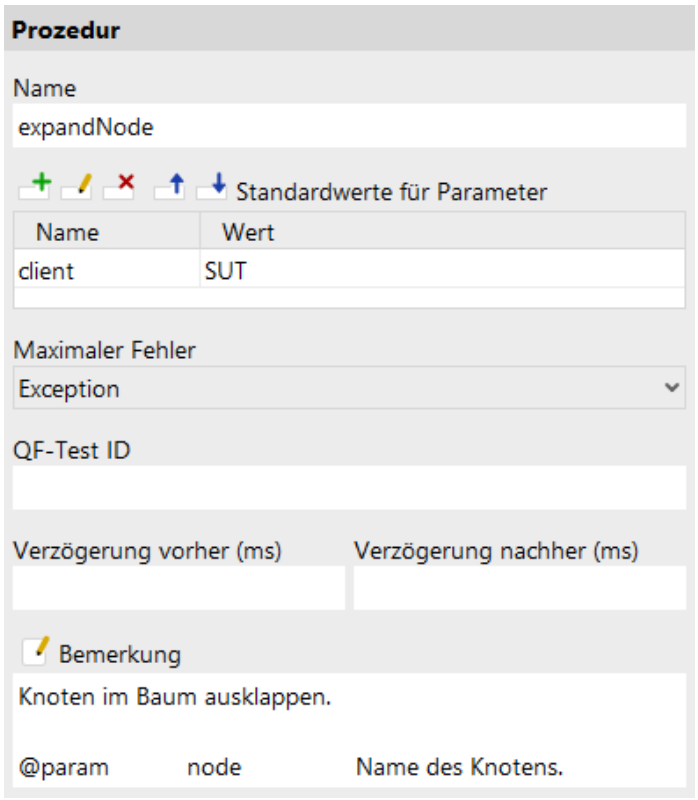


Abbildung 42.20: Prozedur Attribute

Name

Die Prozedur wird über ihren Namen, sowie die Namen ihrer Parents identifiziert. Daher sollten Sie einen Namen pro Package(680) nur einmal vergeben und ”sprechende” Namen verwenden, die etwas über die Prozedur aussagen und die Sie sich gut merken können.

Variabel: Nein

Einschränkungen: Darf nicht leer sein und keines der Zeichen ’.’ oder ’#’ enthalten.

Variablendefinitionen

Hier können Sie Defaultwerte oder ”Fallback” Werte für die Parameter der Prozedur definieren. Diese werden herangezogen, wenn eine Variable an keiner anderen Stelle definiert wurde (vgl. Kapitel 6(116)). Außerdem dienen Sie als Dokumentation und bringen Zeitersparnis bei den Variablendefinitionen(677) Attributen, wenn der Dialog für die Auswahl der Prozedur im Prozeduraufruf(675) Knoten verwendet wird. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20).

4.2+ Mittels Rechtsklick und Auswahl von Parameterordnung zurücksetzen, können Sie die Sortierung der Parameter, wie diese in der Prozedur sortiert sind, wieder herstellen.

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt

hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.5.2 Prozeduraufruf

Mit diesem Knoten lässt sich der Testlauf in einer Prozedur[672] fortsetzen. Dieser können beliebige Parameter übergeben werden, z.B. der Name des Clients, Koordinaten für Mausklicks, Namen für eine Dateiauswahl etc. Nach dem Ende der Prozedur wird der im Attribut Variable für Rückgabewert[676] definierten Variable der Rückgabewert der Prozedur zugewiesen und die Ablaufkontrolle kehrt wieder an die ursprüngliche Stelle zurück.

Die aufzurufende Prozedur wird über ihren Namen und die Namen der Packages[680] identifiziert. Die Namen werden analog zu Java-Klassennamen zusammengesetzt, vom äußersten Package bis zur Prozedur, verbunden mit einem Punkt ('.'). Wollen Sie z.B. die Prozedur `expandNode` im Package `tree`, das seinerseits im Package `main` enthalten ist, aufrufen, so müssen Sie als Name **`main.tree.expandNode`** angeben.

Sie können auch Prozeduren in anderen Testsuiten aufrufen. Näheres hierzu entnehmen Sie bitte Abschnitt 26.1[359].

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Variablen des Aufrufs werden gebunden, die Prozedur an Hand ihres Namens ermittelt und die Ausführung mit der Prozedur fortgesetzt. Nach dem Ende der Prozedur werden die Variablen des Aufrufs gelöscht.

Attribute:

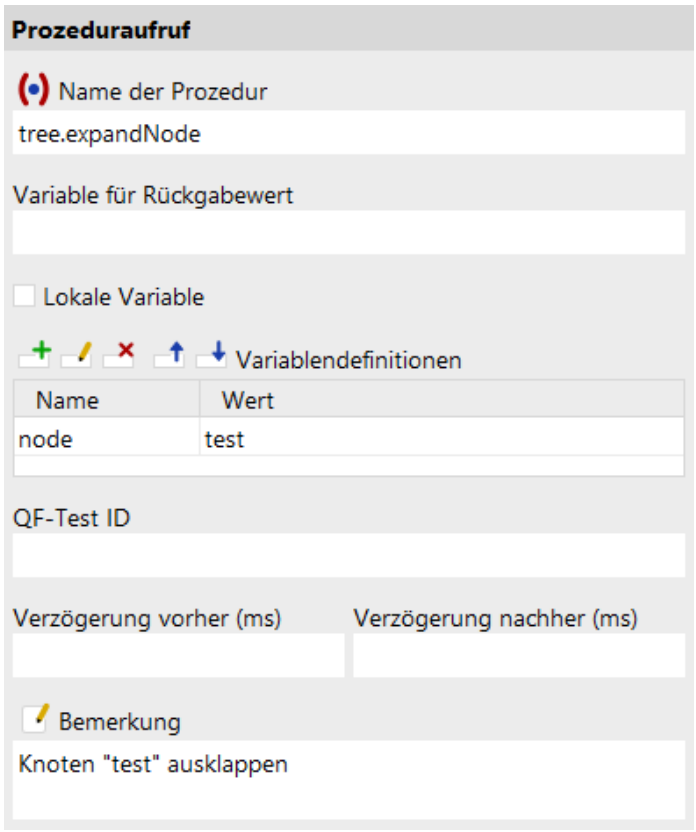


Abbildung 42.21: Prozeduraufruf Attribute

Name der Prozedur

Der volle Name der Prozedur(672), zusammengesetzt aus den Namen der übergeordneten Packages(680) und der Prozedur selbst, verbunden mit '.'.

Der "Prozedur auswählen" Button oberhalb des Attributs öffnet einen Dialog, in dem Sie die Prozedur direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Prozedurknoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Variable für Rückgabewert

Der Rückgabewert der Prozedur wird der in diesem Attribut definierten Variable

zugewiesen. Diese ist optional. Zusätzlich steht der letzte Rückgabewert einer Prozedur auch immer über ${qftest:return} zur Verfügung.

6.1+ Der Name der Variablen wird im Testsuitebaum angezeigt. Wenn es sich um eine globale Variable handelt, in Schwarz, bei einer lokalen in Blau.

Variabel: Ja

Einschränkungen: Kein

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur(672), Abhängigkeit(630) oder Testfall(599) Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6(116).

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte der Parameter für die Prozedur definieren. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.5.3 Return

Mit Hilfe dieses Knotens kann eine Prozedur[672] vorzeitig beendet und zusätzlich ein Wert an den aufrufenden Knoten zurückgegeben werden.

Aus einem Skript kann der selbe Effekt durch werfen einer `ReturnException` erzielt werden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Es wird eine `ReturnException` geworfen die zum Abbruch der innersten Prozedur führt. Wird dieser Knoten außerhalb einer Prozedur aufgerufen wird ein Fehler ausgelöst.

Attribute:

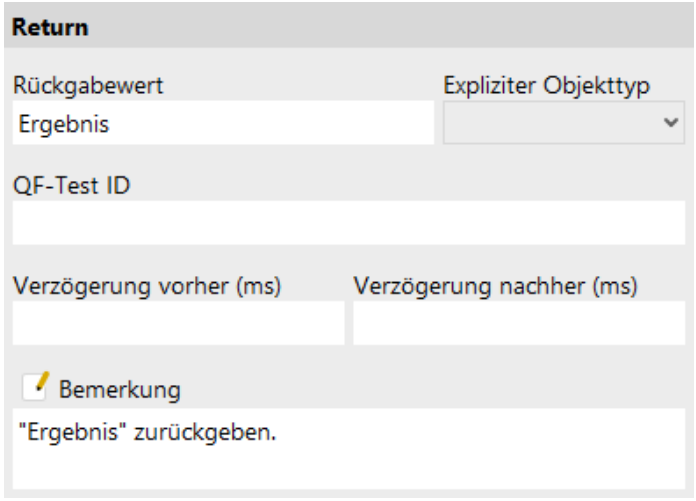


Abbildung 42.22: Return Attribute

Rückgabewert

Der Wert, der von der Prozedur zurückgegeben werden soll. Ist nichts angegeben, wird der leere String zurückgegeben.

Variabel: Ja

Einschränkungen: Keine

Expliziter Objekttyp

9.0+ In QF-Test Variablen können neben Zeichenketten auch beliebige andere Objekte gespeichert werden. Im Textfeld für den Wert kann zwar nur eine Zeichenkette angegeben werden, doch lässt sich hiermit festlegen, wie die Eingabe von QF-Test interpretiert werden soll:

- Keine Auswahl: Die Eingabe wird nicht weiter interpretiert. In den meisten Fällen wird das gespeicherte Objekt eine Zeichenkette sein, es sei denn, die Eingabe wurde durch Variablen-Expansion vollständig durch eine andere Variable ersetzt, wodurch deren Objekt unverändert übernommen wird.
- String: Die Eingabe wird in eine Zeichenkette umgewandelt.
- Boolean: Die Eingabe wird in einen boolschen Wahrheitswert umgewandelt, wobei `0`, leere Strings sowie die Zeichenketten `false`, `no` und `nein` als `false`, andere Werte als `true` ausgewertet werden.
- Number: Die Eingabe wird in eine Zahl umgewandelt. Je nach Eingabewert kann dies ein Integer, Long, BigInteger, Double oder ein BigDecimal sein. Schlägt die Umwandlung fehl, so wird eine `ValueCastException`(966) geworfen.

- Object aus JSON: Die Eingabe wird als JSON-String interpretiert und in verschachtelte Maps und Lists mit Strings, Numbers und Booleans umgewandelt. Schlägt die Umwandlung fehl, so wird eine `ValueCastException`(966) geworfen.

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.5.4 Package

Packages dienen dazu, die Prozeduren[672] einer Testsuite zu strukturieren, um eine bessere Übersicht zu erhalten.

Enthalten in: Package[680], Prozeduren[682]

Kinder: Package[680], Prozedur[672]

Ausführung: Kann nicht ausgeführt werden.

Attribute:

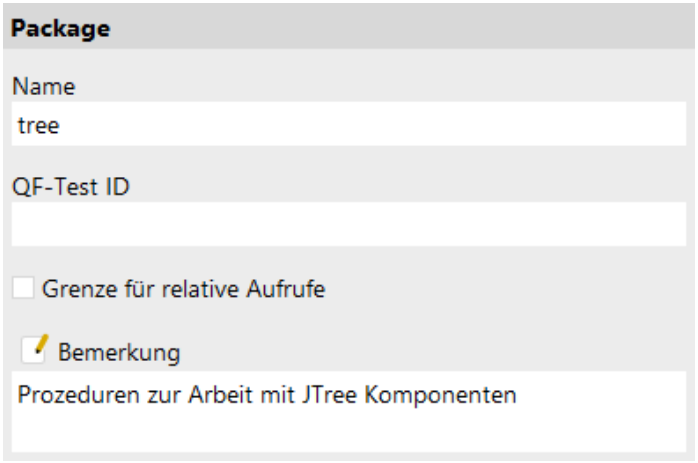


Abbildung 42.23: Package Attribute

Name

Der Name eines Packages ist ein Teil der Identifikation der in ihm enthaltenen Prozeduren. Damit die Zuordnung eindeutig ist, sollte der Name des Packages innerhalb des Parents eindeutig sein. Verwenden Sie auch für die Packages ”sprechende” Namen, die etwas über die darin enthaltenen Prozeduren aussagen und die Sie sich gut merken können.

Variabel: Nein

Einschränkungen: Darf nicht leer sein und keines der Zeichen ’.’ oder ’#’ enthalten.

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Grenze für relative Aufrufe

Ist dieses Attribut gesetzt, ist ein relativer Prozeduraufruf, ein relativer Testaufruf oder eine relative Referenz auf eine Abhängigkeit innerhalb dieses Knotens gestattet. Relative Aufrufe bzw. Referenzen, die diese Grenze überschreiten sind nicht erlaubt. Wenn dieses Attribut in der gesamten Hierarchie nicht gesetzt ist, so können keine relativen Prozeduraufrufe einfügt werden. Relative Testaufrufe sind unterhalb des Testsuite Knotens immer möglich.

Variabel: Nein

Einschränkungen: Keine

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.5.5 Prozeduren

Dies ist die Wurzel aller Packages(680) und Prozeduren(672).

Enthalten in: Wurzelknoten

Kinder: Package(680), Prozedur(672)

Ausführung: Kann nicht ausgeführt werden.

Attribute:

Abbildung 42.24: Prozeduren Attribute

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Grenze für relative Aufrufe

Ist dieses Attribut gesetzt, ist ein relativer Prozeduraufruf, ein relativer Testaufruf oder eine relative Referenz auf eine Abhängigkeit innerhalb dieses Knotens gestattet. Relative Aufrufe bzw. Referenzen, die diese Grenze überschreiten sind nicht erlaubt. Wenn dieses Attribut in der gesamten Hierarchie nicht gesetzt ist, so können keine relativen Prozeduraufrufe einfügt werden. Relative Testaufrufe sind unterhalb des Testsuite Knotens immer möglich.

Variabel: Nein

Einschränkungen: Keine

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6 Ablaufsteuerung

Neben den normalen Sequenzen[599] verfügt QF-Test über eine Reihe von speziellen Kontrollstrukturen, die der Steuerung des Ablaufs eines Tests dienen. Diese orientieren sich an den gängigen Kontrollstrukturen, wie sie auch in Java vorhanden sind.

Außer Schleifen und bedingter Ausführung erbt QF-Test von Java auch die Art der Fehlerbehandlung mittels *Exceptions*, die im Fehlerfall bei der Ausführung eines Knotens ”geworfen” werden. Eine Exception bricht die Ausführung eines Testlaufs schrittweise von unten nach oben ab, so lange bis sie durch ein Catch[707] abgefangen wird, oder bis der gesamte Test beendet ist.

Für Java-Entwickler ist dieses Konzept Teil der täglichen Arbeit. Wenn es Ihnen als Tester verwirrend erscheint, werden Ihre Entwickler es Ihnen sicher gerne erläutern.

Die Klassenhierarchie der Exceptions mit denen QF-Test arbeitet, hat ihre Wurzel in der Klasse `TestException`[958]. Andere Exceptions wie `ComponentNotFoundException`[958] sind Spezialisierungen davon. In Kapitel 43[958] erfahren Sie mehr zu diesem Thema.

Außerdem wird Skripting in Form von Jython (früher *JPython*), Groovy und JavaScript unterstützt. Mehr dazu in Kapitel 11[186].

42.6.1 Schleife

Hierbei handelt es sich um eine Sequenz die mehrfach ausgeführt werden kann. Neben der Anzahl der Wiederholungen kann auch der Name einer Variable angegeben werden, die die Nummer des jeweiligen Durchgangs zugewiesen bekommt.

Dieser Knoten dient vor allem dazu, die Aussage eines Tests über die Zuverlässigkeit des SUT und der Testvorschriften zu verbessern. Ein Test der 100 mal funktioniert sagt mehr aus, als ein Test der einmal funktioniert.

Das Verhalten von Schleifen kann durch Zuweisen eines Wertes an die Zählervariable[686] innerhalb der Schleife beeinflusst werden.

Die Ausführung der Schleife kann mittels eines Break(691) Knotens vorzeitig abgebrochen werden. Ein optionaler Else(701) Knoten am Ende der Schleife wird ausgeführt, falls alle Iterationen der Schleife komplett durchlaufen wurden, ohne auf ein Break zu treffen.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Beliebig

Ausführung: Die Variablen der Sequenz werden gebunden. Die Laufvariable erhält den Wert 0 und die Childknoten werden einer nach dem anderen ausgeführt. Für jede Wiederholung wird der Wert der Laufvariable um 1 erhöht und die Children erneut ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

Abbildung 42.25: Schleife Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Anzahl Wiederholungen

Dieser Wert legt die Anzahl der Durchgänge für die Ausführung fest.

Dieses Attribut ist prädestiniert für den Einsatz von Variablen. Wenn Sie die Anzahl Wiederholungen z.B. auf `$(count)` setzen, können Sie den Wert für die Variable `count` in den Variablen der Suite (vgl. Kapitel 6(116)) auf 1 oder 2 setzen. Das vereinfacht das Erstellen und Ausprobieren der Testsuite.

Wenn Sie den Test dann ernsthaft ausführen, z.B. automatisch mittels `qftest -batch`, können Sie durch Angabe von `-variable count=100` auf der Kommandozeile den Wert für `count` nach Belieben variieren.

Variabel: Ja

Einschränkungen: >0

Zählervariable

Hier geben Sie den Namen einer Variablen an, die während der Ausführung die Nummer des aktuellen Durchgangs enthält, beginnend bei 0. Bei verschachtelten Schleifen sollten Sie unterschiedliche Namen für die Zählervariable verwenden.

Variabel: Ja

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID der Schleife kann in einem Break(691) Knoten angegeben werden, um bei verschachtelten Schleifen gezielt die Schleife mit dieser QF-Test ID abzubrechen.

Variabel: Nein

Einschränkungen: Darf nicht leer sein, keines der Zeichen ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.2 While

Hierbei handelt es sich um eine Sequenz die immer wieder ausgeführt wird, so lange eine Bedingung erfüllt ist.

Die Ausführung des While Knotens kann mittels eines Break[691] Knotens vorzeitig abgebrochen werden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Beliebig

Ausführung: Die Variablen der Sequenz werden gebunden. Die Bedingung wird überprüft und wenn sie erfüllt ist, werden die Childknoten einer nach dem anderen ausgeführt. Dies wiederholt sich so lange, bis die Bedingung nicht mehr erfüllt ist oder der Vorgang durch eine Exception oder einen Break[691] Knoten abgebrochen wird. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

Abbildung 42.26: While Attribute

Bedingung

Eine Bedingung ist ein Ausdruck, dessen Wert entweder wahr oder falsch ist. QF-Test unterscheidet hierbei zwischen einfachen Ausdrücken, die es selbst auswertet und komplexen Ausdrücken, die es an die Skriptsprache Jython weiterreicht.

Leeren Text, sowie den Text `false` (unabhängig von Groß- oder Kleinschreibung) interpretiert QF-Test als falsch, den Text `true` als wahr. Ganze Zahlen sind genau dann wahr, wenn sie nicht 0 sind.

Durch die Auswertung komplexer Ausdrücke mittels Jython stehen hier sehr weitreichende Möglichkeiten zur Verfügung, die noch größer werden, wenn Skripting vollständig integriert ist. Für den Moment muss die folgende Kurzzusammenfassung genügen: Jython unterstützt die üblichen Operatoren `==`, `!=`, `>`, `>=`, `<` und `<=`. Sie können Ausdrücke mit `and` und `or` verknüpfen und die Priorität durch Klammerung festlegen.

Hinweis

Der Zugriff auf QF-Test Variablen in Bedingungen folgt denselben Regeln wie in Jython-Skripten (vgl. Abschnitt 11.3.3(191)). Die standard QF-Test Syntax `$(...)` und `${...:...}` kann für numerische und boolesche Werte verwendet werden. Auf Zeichenketten sollte mittels `rc.getStr` zugegriffen werden.

Wichtig: Wenn Sie Zeichenketten (im Gegensatz zu Zahlen) vergleichen wollen, müssen Sie diese für Jython mit einem einfachen oder doppelten Anführungszeichen schützen. Andernfalls interpretiert Jython diese als Jython-Variablen, die natürlich nicht definiert sind und damit zu einem Syntaxfehler führen.

Einige Beispiele zum besseren Verständnis:

Ausdruck	**Wert**
Leerer Text	Falsch
0	Falsch
21	Wahr
False	Falsch
True	Wahr
abc abc	Syntaxfehler
25 > 0	Wahr
${qftest:batch}	True if QF-Test is run in batch mode
not ${qftest:batch}	True if QF-Test is run in interactive mode
rc.getStr(”system”, ”java.version”) == ”1.3.1”	Wahr falls JDK Version 1.3.1 ist
rc.getStr(”system”, ”java.version”)[0] == ”1”	Wahr falls JDK Version mit 1 beginnt
(1 > 0 and 0 == 0) or 2 < 1	Wahr

Tabelle 42.13: Beispiele für Bedingungen

Variabel: Ja

Einschränkungen: Gültige Syntax

Skriptsprache

Dieses Attribut legt den Interpreter fest, in dem das Skript ausgeführt wird, oder in anderen Worten die Skriptsprache. Mögliche Werte sind ”Jython”, ”Groovy” und ”JavaScript”.

Variabel: Nein

Einschränkungen: Keine

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID des While Knotens kann in einem Break[691] Knoten angegeben werden, um bei verschachtelten Schleifen gezielt die Schleife mit dieser QF-Test ID abzubrechen.

Variabel: Nein

Einschränkungen: Darf nicht leer sein, keines der Zeichen '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.3 Break

Dieser Knoten dient dazu, Schleifen[684] und While[688] Knoten vorzeitig zu beenden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Es wird eine `BreakException` geworfen. Wird diese von einer passenden Schleife gefangen, so wird die Schleife beendet. Andernfalls führt sie zu einem Fehler.

Attribute:

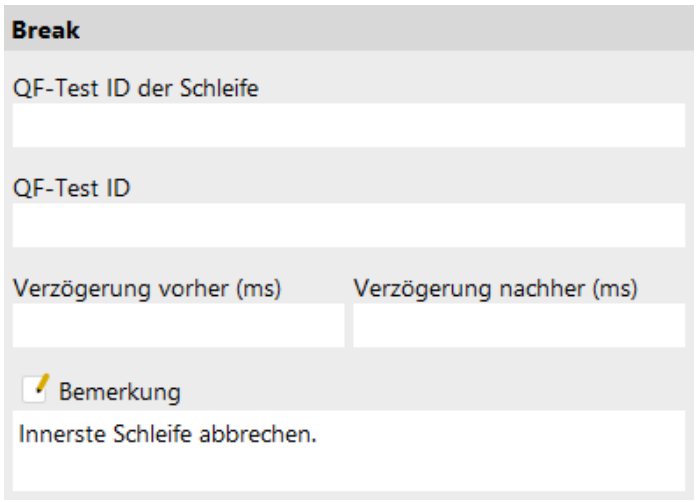


Abbildung 42.27: Break Attribute

QF-Test ID der Schleife

Hiermit können Sie bei verschachtelten Schleifen gezielt die Schleife auswählen, die abgebrochen werden soll, indem Sie im Schleife(684) Knoten das QF-Test ID(687) setzen und hier referenzieren. Das gleiche gilt für While(688) Knoten mit dem QF-Test ID(690) Attribut. Ist dieses Feld leer, wird die innerste Schleife abgebrochen. Falls Sie eine Schleife des Datentreiber(646) Knotens abbrechen wollen, so geben Sie hier den Wert des Attributs 'Name' des jeweiligen Datentreibers(646) an.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung

bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.4 If

Analog zu Java werden die Childknoten dieser Sequenz nur dann ausgeführt, wenn eine Bedingung erfüllt ist. Allerdings unterscheidet sich QF-Test von gängigen Programmiersprachen in der Art in der die Alternativzweige angeordnet sind.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Beliebige Knoten, gefolgt von beliebig vielen Elseif(697) Knoten mit einem optionalen Else(701) Knoten am Ende.

Ausführung: Die Variablen der Sequenz werden gebunden. Die Bedingung wird überprüft und wenn sie erfüllt ist, werden die normalen Childknoten einer nach dem anderen ausgeführt. Ist die Bedingung nicht erfüllt, werden die Bedingungen der Elseif(697) Childknoten eine nach der anderen geprüft, bis eine Bedingung erfüllt ist und der entsprechende Elseif(697) Knoten ausgeführt wird. Ist kein Elseif(697) Childknoten vorhanden oder keine der Bedingungen erfüllt, wird der Else(701) Knoten ausgeführt, sofern vorhanden. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

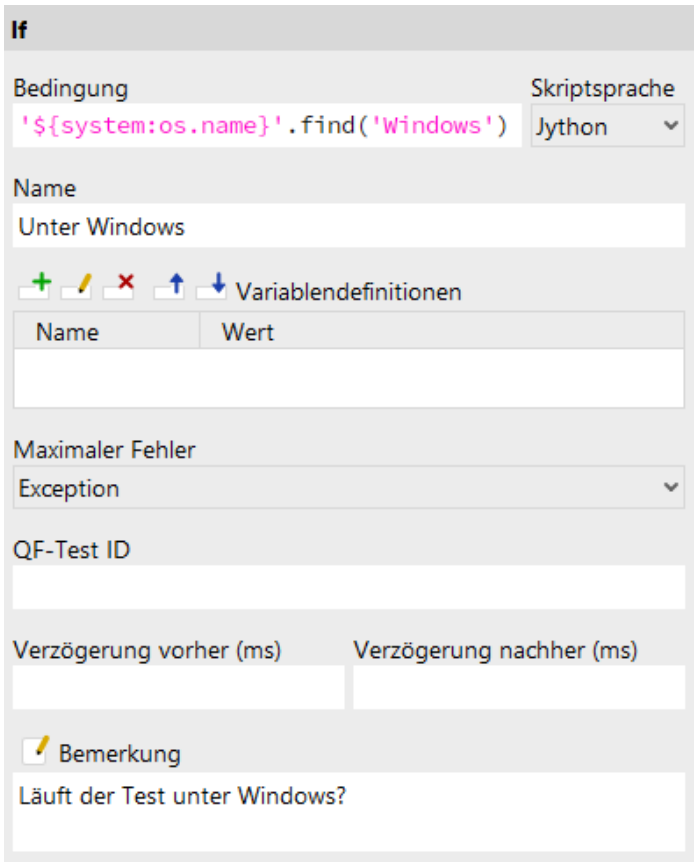


Abbildung 42.28: If Attribute

Bedingung

Eine Bedingung ist ein Ausdruck, dessen Wert entweder wahr oder falsch ist. QF-Test unterscheidet hierbei zwischen einfachen Ausdrücken, die es selbst auswertet und komplexen Ausdrücken, die es an die Skriptsprache Jython weiterreicht.

Leeren Text, sowie den Text `false` (unabhängig von Groß- oder Kleinschreibung) interpretiert QF-Test als falsch, den Text `true` als wahr. Ganze Zahlen sind genau dann wahr, wenn sie nicht 0 sind.

Durch die Auswertung komplexer Ausdrücke mittels Jython stehen hier sehr weitreichende Möglichkeiten zur Verfügung, die noch größer werden, wenn Skripting vollständig integriert ist. Für den Moment muss die folgende Kurzzusammenfassung genügen: Jython unterstützt die üblichen Operatoren `==`, `!=`, `>`, `>=`, `<` und

<=. Sie können Ausdrücke mit `and` und `or` verknüpfen und die Priorität durch Klammerung festlegen.

Hinweis

Der Zugriff auf QF-Test Variablen in Bedingungen folgt denselben Regeln wie in Jython-Skripten (vgl. Abschnitt 11.3.3(191)). Die standard QF-Test Syntax `$(...)` und `${...:...}` kann für numerische und boolesche Werte verwendet werden. Auf Zeichenketten sollte mittels `rc.getStr` zugegriffen werden.

Wichtig: Wenn Sie Zeichenketten (im Gegensatz zu Zahlen) vergleichen wollen, müssen Sie diese für Jython mit einem einfachen oder doppelten Anführungszeichen schützen. Andernfalls interpretiert Jython diese als Jython-Variablen, die natürlich nicht definiert sind und damit zu einem Syntaxfehler führen.

Einige Beispiele zum besseren Verständnis:

Ausdruck	Wert
Leerer Text	Falsch
0	Falsch
21	Wahr
False	Falsch
True	Wahr
abc abc	Syntaxfehler
25 > 0	Wahr
${qftest:batch}	True if QF-Test is run in batch mode
not ${qftest:batch}	True if QF-Test is run in interactive mode
rc.getStr(”system”, ”java.version”) == ”1.3.1”	Wahr falls JDK Version 1.3.1 ist
rc.getStr(”system”, ”java.version”)[0] == ”1”	Wahr falls JDK Version mit 1 beginnt
(1 > 0 and 0 == 0) or 2 < 1	Wahr

Tabelle 42.14: Beispiele für Bedingungen

Variabel: Ja

Einschränkungen: Gültige Syntax

Skriptsprache

Dieses Attribut legt den Interpreter fest, in dem das Skript ausgeführt wird, oder in anderen Worten die Skriptsprache. Mögliche Werte sind ”Jython”, ”Groovy” und ”JavaScript”.

Variabel: Nein

Einschränkungen: Keine

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung

bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.5 Elseif

Dieser Knoten stellt einen Alternativzweig in einem If(693) Knoten dar. Ist die Bedingung des If Knotens nicht erfüllt, wird der erste Elseif Knoten ausgeführt, dessen Bedingung erfüllt ist.

Enthalten in: If(693)

Kinder: Beliebig

Ausführung: Die Variablen der Sequenz werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

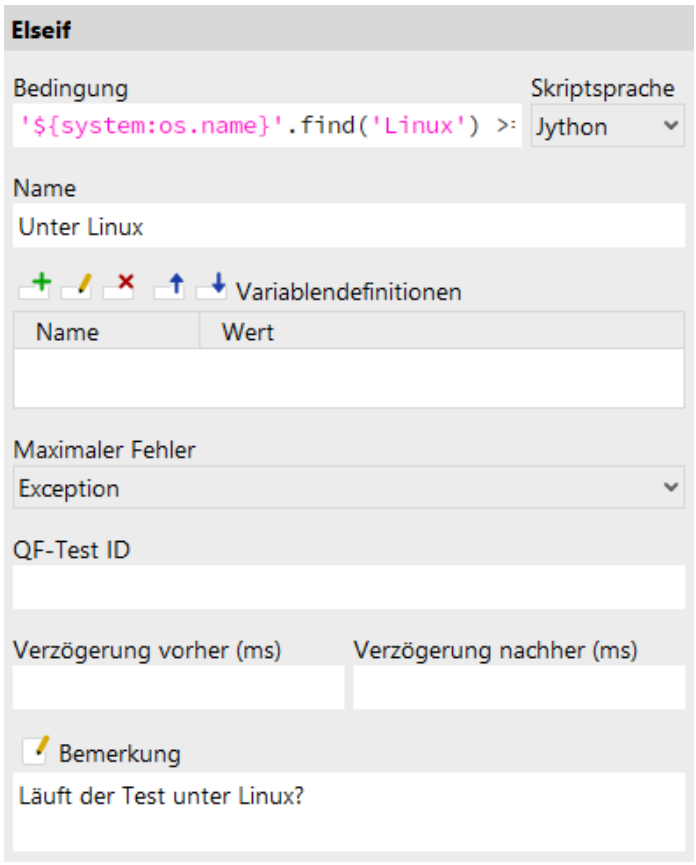


Abbildung 42.29: Elseif Attribute

Bedingung

Eine Bedingung ist ein Ausdruck, dessen Wert entweder wahr oder falsch ist. QF-Test unterscheidet hierbei zwischen einfachen Ausdrücken, die es selbst auswertet und komplexen Ausdrücken, die es an die Skriptsprache Jython weiterreicht.

Leeren Text, sowie den Text `false` (unabhängig von Groß- oder Kleinschreibung) interpretiert QF-Test als falsch, den Text `true` als wahr. Ganze Zahlen sind genau dann wahr, wenn sie nicht 0 sind.

Durch die Auswertung komplexer Ausdrücke mittels Jython stehen hier sehr weitreichende Möglichkeiten zur Verfügung, die noch größer werden, wenn Skripting vollständig integriert ist. Für den Moment muss die folgende Kurzzusammenfassung genügen: Jython unterstützt die üblichen Operatoren `==`, `!=`, `>`, `>=`, `<` und `<=`. Sie können Ausdrücke mit `and` und `or` verknüpfen und die Priorität durch Klammerung festlegen.

Hinweis

Der Zugriff auf QF-Test Variablen in Bedingungen folgt denselben Regeln wie in

Jython-Skripten (vgl. Abschnitt 11.3.3[191]). Die standard QF-Test Syntax `$(...)` und `${...:...}` kann für numerische und boolesche Werte verwendet werden. Auf Zeichenketten sollte mittels `rc.getStr` zugegriffen werden.

Wichtig: Wenn Sie Zeichenketten (im Gegensatz zu Zahlen) vergleichen wollen, müssen Sie diese für Jython mit einem einfachen oder doppelten Anführungszeichen schützen. Andernfalls interpretiert Jython diese als Jython-Variablen, die natürlich nicht definiert sind und damit zu einem Syntaxfehler führen.

Einige Beispiele zum besseren Verständnis:

Ausdruck	**Wert**
Leerer Text	Falsch
0	Falsch
21	Wahr
False	Falsch
True	Wahr
abc abc	Syntaxfehler
25 > 0	Wahr
${qftest:batch}	True if QF-Test is run in batch mode
not ${qftest:batch}	True if QF-Test is run in interactive mode
rc.getStr(”system”, ”java.version”) == ”1.3.1”	Wahr falls JDK Version 1.3.1 ist
rc.getStr(”system”, ”java.version”)[0] == ”1”	Wahr falls JDK Version mit 1 beginnt
(1 > 0 and 0 == 0) or 2 < 1	Wahr

Tabelle 42.15: Beispiele für Bedingungen

Variabel: Ja

Einschränkungen: Gültige Syntax

Skriptsprache

Dieses Attribut legt den Interpreter fest, in dem das Skript ausgeführt wird, oder in anderen Worten die Skriptsprache. Mögliche Werte sind ”Jython”, ”Groovy” und ”JavaScript”.

Variabel: Nein

Einschränkungen: Keine

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.6 Else

Ein Else Knoten wird ausgeführt, wenn weder die Bedingung seines If(693) Parents, noch die Bedingungen der Elseif(697) Knoten erfüllt sind.

Enthalten in: If(693), Schleife(684), Try(704)

Kinder: Beliebig

Ausführung: Die Variablen der Sequenz werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

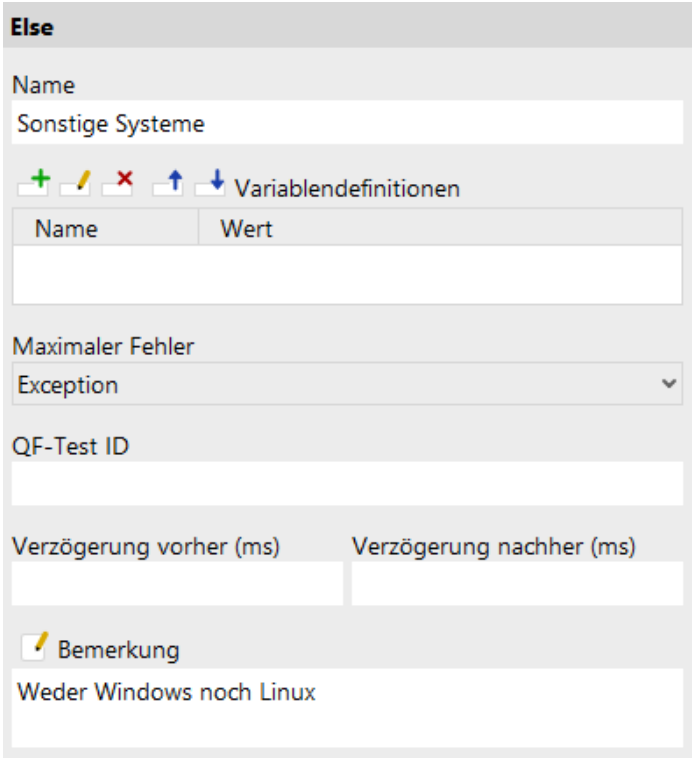


Abbildung 42.30: Else Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.7 Try

Ein Try ist eine besondere Sequenz, deren Semantik dem Java-Konstrukt `try-catch-finally` entspricht. Dieses Konstrukt wurde, wie bei Python, erweitert, sodass optionale `else` Blöcke im Try möglich sind. Sie verhält sich wie eine normale Sequenz[618], die zusätzlich über die Möglichkeit verfügt, Fehler zu erkennen und abzufangen. Dies kann z.B. verhindern, dass der Ablauf einer ganzen Testsuite nur wegen einem relativ unbedeutenden Fehler abgebrochen wird. Außerdem kann es wichtig sein, einzelne Aufräumarbeiten auch im Fehlerfall durchzuführen.

Um einen Fehler, d.h. eine Exception, abzufangen, können nach den normalen Children des Try ein oder mehrere Catch[707] Knoten eingefügt werden. Diese speziellen Sequenzen werden nur ausgeführt, wenn die Exception zu ihrem Exception-Klasse[708] Attribut passt. Es wird immer höchstens ein Catch ausgeführt, und zwar der erste, der auf die Exception passt.

Ein möglicher Else[701] Knoten am Ende wird genau dann ausgeführt, wenn kein Catch[707] Knoten ausgeführt wurde. Das heißt, wenn kein Fehler im Try Block aufgetreten ist.

Ganz am Ende des Try darf außerdem ein Finally[711] Knoten stehen. Dabei handelt es sich um eine Sequenz, die in jedem Fall abgearbeitet wird, unabhängig davon, ob eine Exception auftritt und unabhängig davon, ob Sie von einem Catch behandelt wird.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Beliebige Knoten, gefolgt von beliebig vielen Catch[707] Knoten mit einem optionalen Else[701] Knoten und/oder Finally[711] Knoten am Ende.

Ausführung: Die Variablen der Sequenz werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Wird dabei eine Exception ausgelöst, wird deren Ausführung beendet und unter den Catch Knoten der Sequenz die erste ermittelt, die diese Exception abfängt. Ist ein solcher Catch Knoten vorhanden, wird er ausgeführt. Die Exception ist damit ”entschärft”, der Test wird nach dem Try normal fortgesetzt. Ist kein passender Catch vorhanden, wird die Exception nach oben weitergereicht. Ist ein Finally Knoten vorhanden, wird er in jedem Fall als letztes ausgeführt, auch wenn keine Exception auftritt oder eine Exception nicht behandelt wurde. Zuletzt werden die Variablen der Sequenz wieder gelöscht. Tritt eine Exception während der Ausführung eines Catch auf, so wird der Try - nach Ausführung des Finally - mit dieser neuen Excepti-

on beendet. Eine Exception innerhalb des Finally Knotens beendet diesen und den Try sofort.

Attribute:

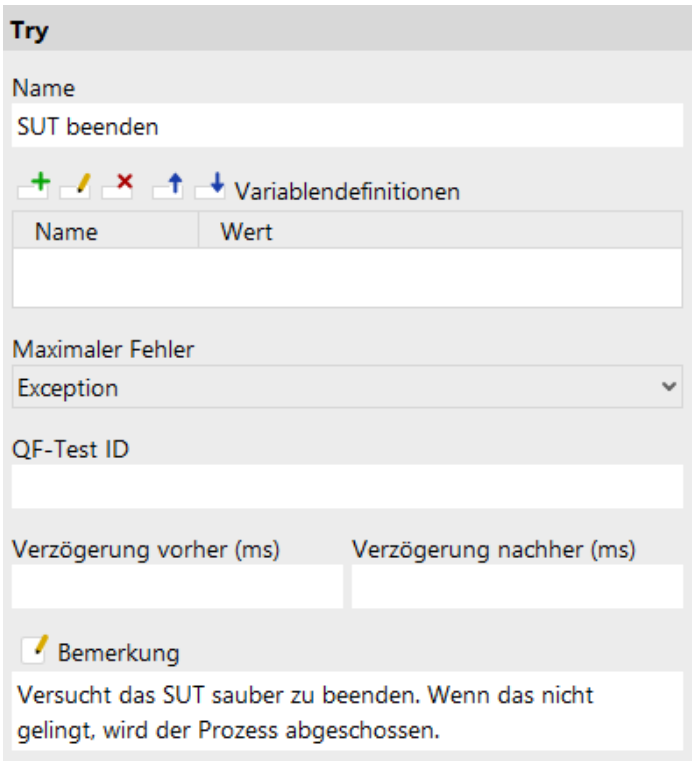


Abbildung 42.31: Try Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Bei einem Try Knoten wird der Status des Protokolls zusätzlich durch den Maximalen Fehler(710) des Catch(707) Knotens beschränkt, der eine Exception abfängt.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.8 Catch

Ein Catch Knoten ist eine spezielle Sequenz[618] und kann nur innerhalb eines Try[704] oder eines Abhängigkeit[630] Knotens stehen. Er dient dazu, Exceptions abzufangen, die während der Ausführung eines Try Knotens oder eines Testfall[599] Knotens mit einer Abhängigkeit auftreten können.

Ein Catch Knoten kann eine Exception fangen, wenn die Klasse der auftretenden Exception entweder die im Exception-Klasse[708] Attribut des Catch definierten Klasse, oder eine ihrer Basisklassen ist. Das entspricht dem Verhalten in Java.

Enthalten in: Try[704], Abhängigkeit[630]

Kinder: Beliebig

Ausführung: Die Variablendefinitionen[709] der Catch Sequenz werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

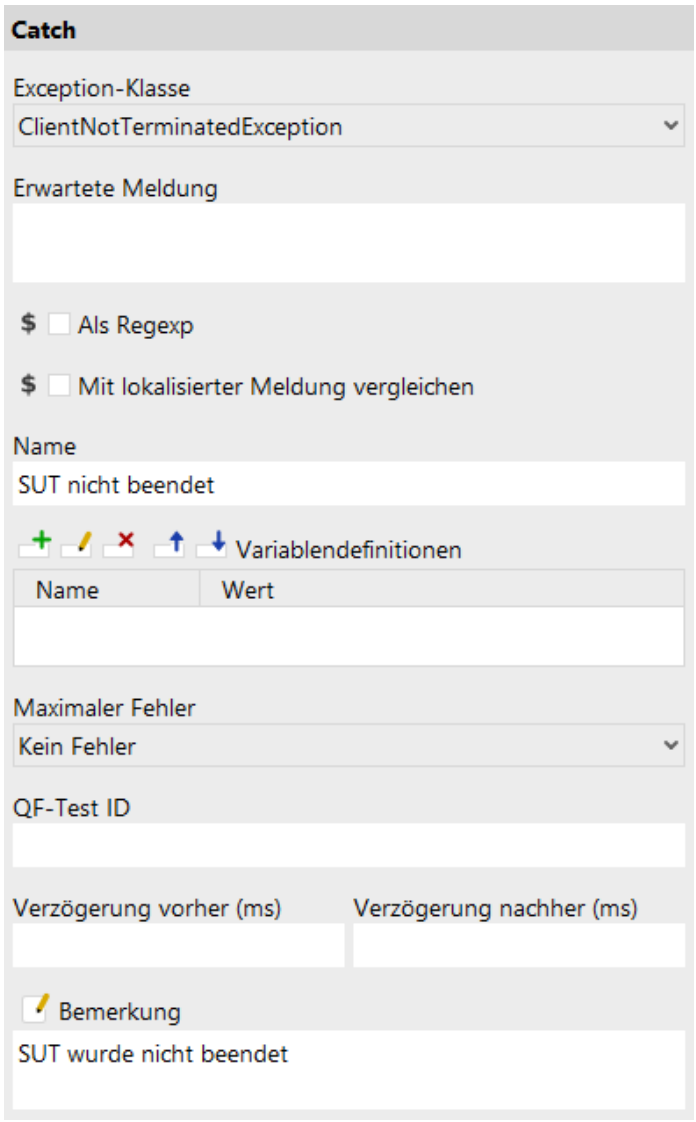


Abbildung 42.32: Catch Attribute

Exception-Klasse

In dieser *ComboBox* wählen Sie die Klasse der Exception aus, die von diesem Knoten abgefangen werden soll. Alle Exceptions in QF-Test sind von der Klasse `TestException`(958) abgeleitet. Eine Aufstellung aller möglichen Exceptions finden Sie in Kapitel 43(958).

Variabel: Nein

Einschränkungen: Keine

Erwartete Meldung

Sie können die zu fangende Exception genauer eingrenzen, indem Sie hier die zu erwartende Meldung festlegen. Ist dieses Attribut leer, werden alle Exceptions der angegebenen Klasse gefangen. Andernfalls wird es mit dem Rückgabewert von `exception.getMessage()` verglichen und die Exception nur bei Übereinstimmung gefangen.

Variabel: Ja

Einschränkungen: Gültige Regexp, falls Als Regexp gesetzt ist.

Als Regexp

Ist dieses Attribut gesetzt, findet der Vergleich mit der Meldung der Exception mittels eines regulären Ausdrucks statt (vgl. Abschnitt 49.3(1023)), andernfalls als 1:1 Textvergleich.

Variabel: Ja

Einschränkungen: Keine

Mit lokalisierter Meldung vergleichen

Die meisten Exceptions haben zwei Arten von Fehlermeldung: Die Rohmeldung ist typischerweise ein kurzer, englischer Text, während die lokalisierte Meldung mehr Details enthält, allerdings auch abhängig von den QF-Test Spracheinstellungen entweder in Deutsch oder Englisch gehalten ist. Ist dieses Attribut gesetzt, wird mit der lokalisierten Meldung verglichen, andernfalls mit der Rohmeldung. Letzteres ist üblicherweise vorzuziehen, da man sich nicht von der Spracheinstellung abhängig macht bzw. beide Varianten per regulärem Ausdruck abdecken muss.

Variabel: Ja

Einschränkungen: Keine

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Im Gegensatz zu dem Maximalen Fehler[619] der anderen Sequenzen[599], bestimmt dieses Attribut nicht, welchen Fehlerstatus das Protokoll für den Catch Knoten selbst weitergibt, sondern legt den maximalen Fehler für seinen Try[704] Parentknoten fest, sofern dieser Catch Knoten zur Behandlung einer Exception ausgeführt wird.

Fehler die innerhalb dieses Catch Knotens auftreten, werden mit **unverändertem** Status weitergegeben, damit Probleme, die während der Behandlung einer Exception auftreten, nicht übersehen werden.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.9 Finally

Ein besondere Bedeutung kommt dem Finally am Ende eines Try[704] Knotens zu, da es in jedem Fall ausgeführt wird. Damit kann sichergestellt werden, dass Aufräumarbeiten, wie z.B. das Löschen einer temporären Datei oder das Schließen eines Fensters, tatsächlich durchgeführt werden.

Enthalten in: Try[704]

Kinder: Beliebig

Ausführung: Die Variablen der Sequenz werden gebunden und die Childknoten einer nach dem anderen ausgeführt. Anschließend werden die Variablen der Sequenz wieder gelöscht.

Attribute:

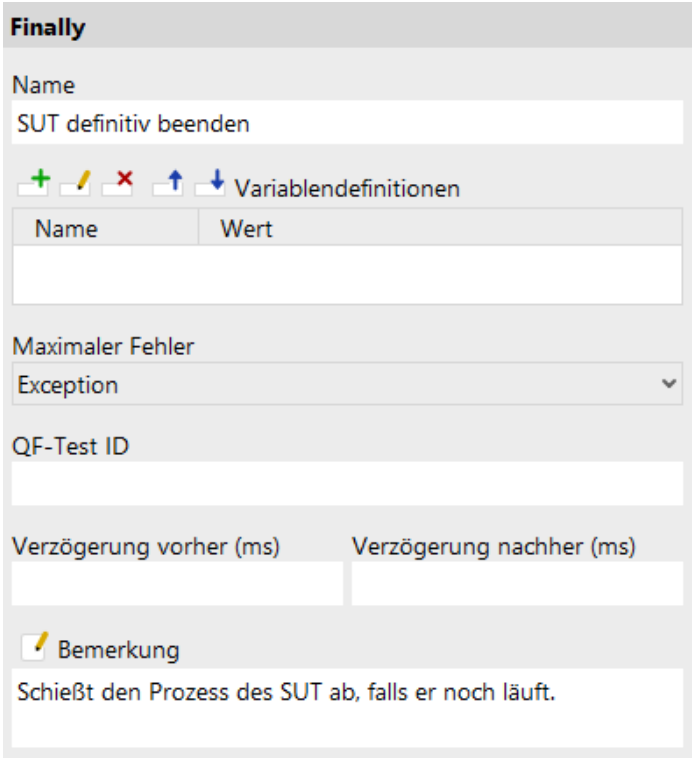


Abbildung 42.33: Finally Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5(20). Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6(116).

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.10 Throw

Dieser Knoten dient dazu, explizit eine Exception zu werfen, um auf eine besondere Ausnahmesituation zu reagieren.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Eine `UserException`(966) mit der Fehlermeldung aus dem Attribut Meldung für die Exception(714) wird ausgelöst.

Attribute:

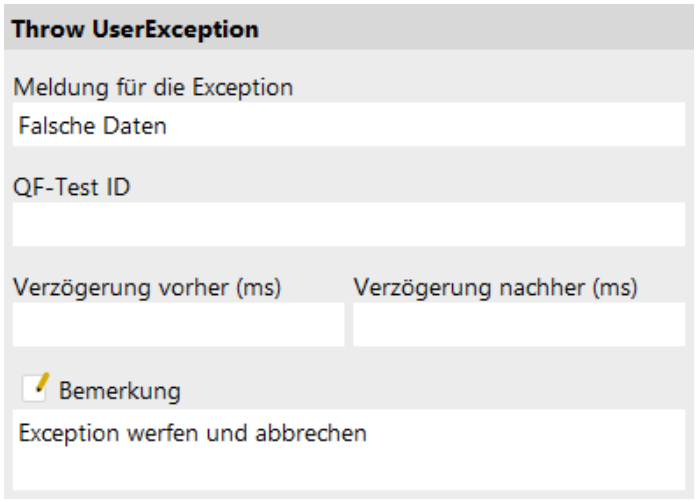


Abbildung 42.34: Throw Attribute

Meldung für die Exception

Hier können Sie eine beliebige Fehlermeldung für die zu werfende `UserException`(966) angeben.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.11 Rethrow

Eine Exception, die von einem Catch[707] Knoten abgefangen wurde, kann mit Hilfe dieses Knotens erneut geworfen werden. Dies ist besonders in Situationen nützlich, wo alle Exceptions außer einer bestimmten abgefangen werden sollen. Dies erreichen Sie, indem Sie unterhalb eines Try[704] Knotens zunächst einen Catch für die spezielle Exception anlegen, gefolgt von einem Catch für eine `TestException`[958]. In den ersten Catch fügen Sie dann einen Rethrow Knoten ein.

Enthalten in: Alle Arten von Sequenzen(599). Der Rethrow Knoten muss sich nicht direkt unterhalb des entsprechenden Catch Knotens befinden. Er kann z.B. auch in einer Prozedur(672) untergebracht werden, die zur Behandlung der Exception aufgerufen wird.

Kinder: Keine

Ausführung: Die letzte Exception, die von einem Catch(707) Knoten abgefangen wurde, wird erneut geworfen. Gibt es keine solche Exception, wird eine `CannotRethrowException`(967) ausgelöst.

Attribute:

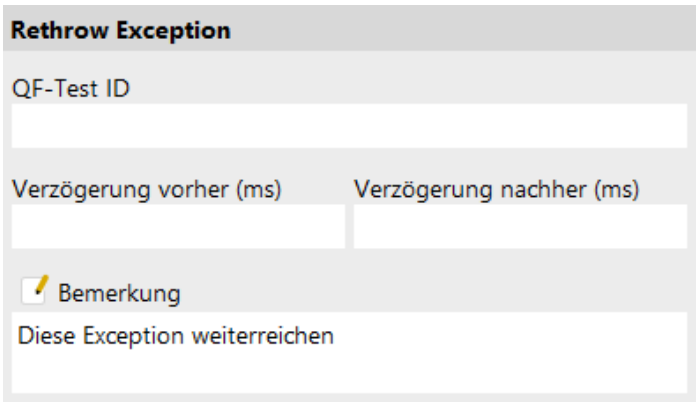


Abbildung 42.35: Rethrow Attribute

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder

Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.12 Server-Skript

Serverseitige Skripte werden von einem in QF-Test eingebetteten Interpreter (Jython, Groovy oder JavaScript) ausgeführt. Skripte werden ausführlich in Kapitel 11[186] und Kapitel 50[1028] erläutert. Nachdem serverseitige Skripte im QF-Test Interpreter laufen, können diese nicht auf das SUT zugreifen. Sie sollten diese Skripte für Aktionen ohne SUT bzw. zeitintensive Aktionen verwenden, wie Zugriffen auf eine Datenbank oder Dateioperationen.

Enthalten in: Alle Arten von von Sequenzen[599].

Kinder: Keine

Ausführung: Das Skript wird von einem Interpreter ausgeführt.

Attribute:

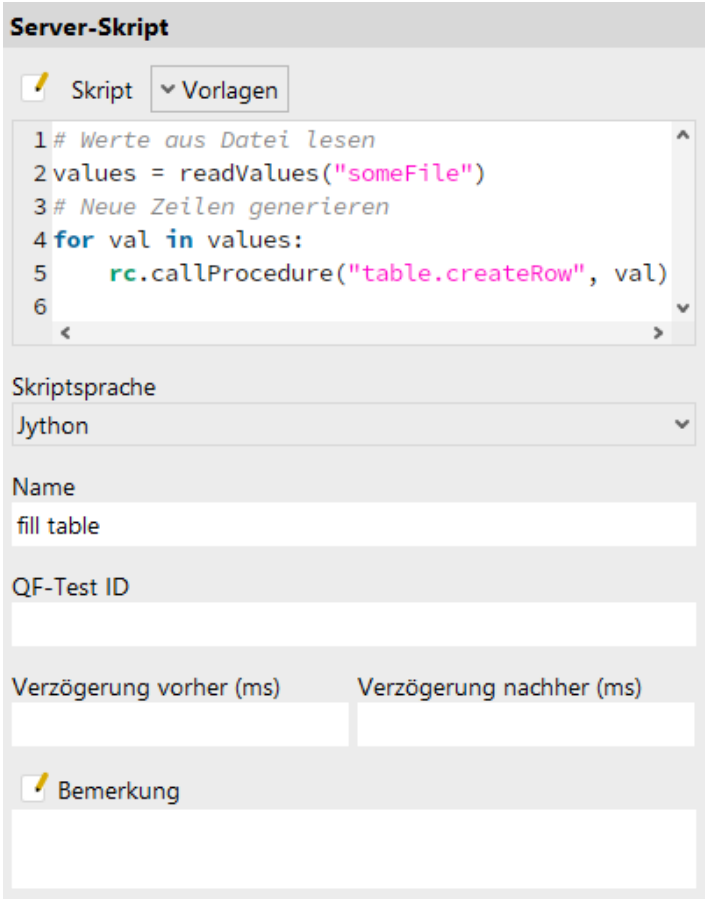


Abbildung 42.36: Server-Skript Attribute

Skript

Das Skript, das ausgeführt werden soll.

Hinweis In Jython-Skripten können QF-Test Variablen der Form $(var) oder ${Gruppe:Name} verwendet werden. Diese werden expandiert bevor das Skript an den Interpreter übergeben wird. Dies kann zu unerwünschten Effekten führen. Stattdessen sollte dafür die Methode `rc.getStr` verwendet werden, die in allen Skriptsprachen zur Verfügung steht (vgl. Abschnitt 11.3.3(192)).

Hinweis Trotz Syntax-Highlighting und automatischer Einrückung ist dieses Textfeld womöglich nicht der geeignete Ort, um komplexe Skripte zu schreiben. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons das Skript komfortabel bearbeitet werden kann. Außerdem können komplexe Skripte in sepa-

rate Module ausgelagert werden, die dann in QF-Test Skripte importiert werden. (vgl. Kapitel 50(1028)).

Variabel: Ja

Einschränkungen: Gültige Syntax

Vorlagen

Dieses Dropdown-Menü enthält eine Liste nützlicher Vorlagenskripte. Die verfügbaren Vorlagen unterscheiden sich je nach gewähltem Skripttyp und Skriptsprache.

Wenn Sie eine dieser Vorlagen auswählen, wird der aktuelle Inhalt Ihres Skripts ersetzt.

Sie können Ihre eigenen Vorlagen zu diesem Menü hinzufügen, indem Sie "Benutzervorlagen-Verzeichnis öffnen" wählen und Ihre Vorlagendateien dort ablegen. Die folgenden Dateitypen sind gültig:

- **[Verzeichnis]:** Wird als Untermenü angezeigt.
- **.py:** Eine Jython-Vorlage.
- **.groovy:** Eine Groovy-Vorlage.
- **.js:** Eine JavaScript-Vorlage.

Skriptsprache

Dieses Attribut legt den Interpreter fest, in dem das Skript ausgeführt wird, oder in anderen Worten die Skriptsprache. Mögliche Werte sind "Jython", "Groovy" und "JavaScript".

Variabel: Nein

Einschränkungen: Keine

Name

Der Name eines Skripts ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion des Skripts aussagen.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.6.13 SUT-Skript

Client-seitige Skripte werden von einem Interpreter (Jython, Groovy oder JavaScript) ausgeführt, den QF-Test in das SUT einbettet. Skripte werden ausführlich in Kapitel 11[186] und Kapitel 50[1028] erläutert. Nachdem client-seitige Skripte innerhalb des SUTs laufen, können diese auf die Komponenten zugreifen.

Enthalten in: Alle Arten von von Sequenzen[599].

Kinder: Keine

Ausführung: Das Skript wird von einem Interpreter im SUT ausgeführt.

Attribute:

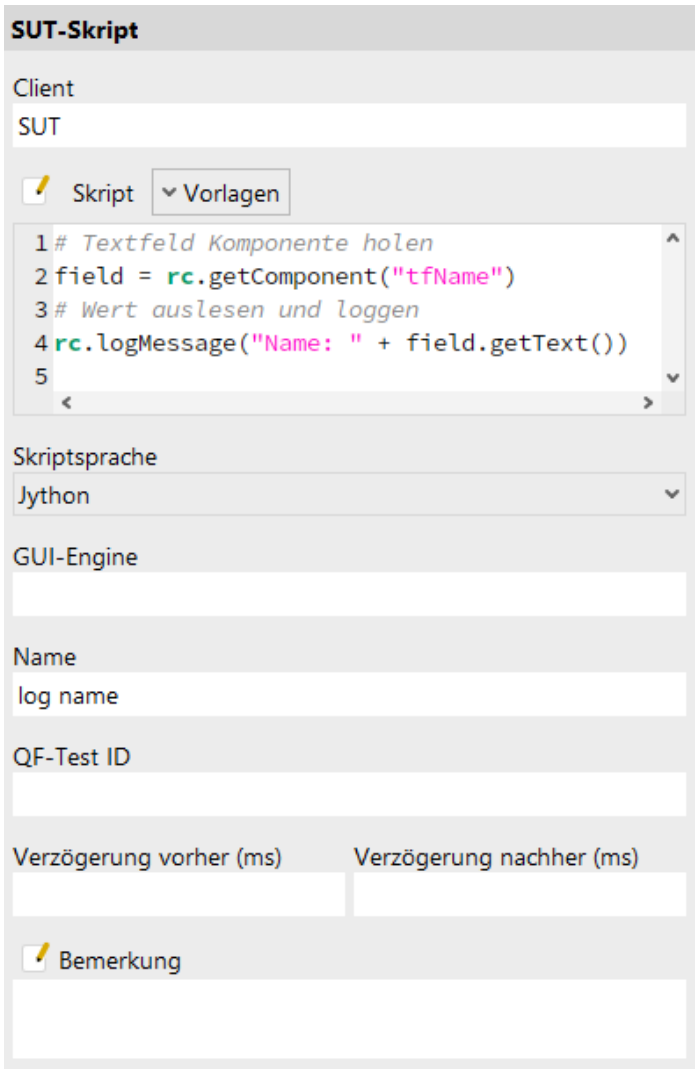


Abbildung 42.37: SUT-Skript Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, in dem das Skript ausgeführt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Skript

Das Skript, das ausgeführt werden soll.

Hinweis

In Jython-Skripten können QF-Test Variablen der Form $(var) oder

${Gruppe:Name} verwendet werden. Diese werden expandiert bevor das Skript an den Interpreter übergeben wird. Dies kann zu unerwünschten Effekten führen. Stattdessen sollte dafür die Methode `rc.getStr` verwendet werden, die in allen Skriptsprachen zur Verfügung steht (vgl. Abschnitt 11.3.3(192)).

Hinweis

Trotz Syntax-Highlighting und automatischer Einrückung ist dieses Textfeld womöglich nicht der geeignete Ort, um komplexe Skripte zu schreiben. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons das Skript komfortabel bearbeitet werden kann. Außerdem können komplexe Skripte in separate Module ausgelagert werden, die dann in QF-Test Skripte importiert werden. (vgl. Kapitel 50(1028)).

Variabel: Ja

Einschränkungen: Gültige Syntax

Vorlagen

Dieses Dropdown-Menü enthält eine Liste nützlicher Vorlagenskripte. Die verfügbaren Vorlagen unterscheiden sich je nach gewähltem Skripttyp und Skriptsprache.

Wenn Sie eine dieser Vorlagen auswählen, wird der aktuelle Inhalt Ihres Skripts ersetzt.

Sie können Ihre eigenen Vorlagen zu diesem Menü hinzufügen, indem Sie "Benutzervorlagen-Verzeichnis öffnen" wählen und Ihre Vorlagendateien dort ablegen. Die folgenden Dateitypen sind gültig:

- **[Verzeichnis]:** Wird als Untermenü angezeigt.
- **.py:** Eine Jython-Vorlage.
- **.groovy:** Eine Groovy-Vorlage.
- **.js:** Eine JavaScript-Vorlage.

Skriptsprache

Dieses Attribut legt den Interpreter fest, in dem das Skript ausgeführt wird, oder in anderen Worten die Skriptsprache. Mögliche Werte sind "Jython", "Groovy" und "JavaScript".

Variabel: Nein

Einschränkungen: Keine

GUI-Engine

Die GUI-Engine in der das Skript ausgeführt werden soll. Nur relevant für SUTs mit mehr als einer GUI-Engine wie in Kapitel 45(998) beschrieben.

Variabel: Ja

Einschränkungen: Siehe Kapitel 45(998)

Name

Der Name eines Skripts ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion des Skripts aussagen.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7 Prozesse

Um die Kommunikation zwischen dem SUT und QF-Test herzustellen, müssen der oder die Prozesse des SUT aus QF-Test heraus gestartet werden. Details zu den Hintergründen beim Starten einer Applikation und Hinweise darauf, welche Methode wann am geeignetsten ist, finden Sie in Kapitel 46[1000].

Mit Hilfe der folgenden Knotentypen können Sie Programme starten, auf die Verbindung mit einem SUT Client warten, auf das Ende eines Programms warten und dessen Exitcode auswerten, sowie Prozesse ”abschießen”, vergleichbar mit einem `kill` unter Linux. Die Zahl der Prozesse, die gleichzeitig von QF-Test gestartet werden können, ist nur durch das zu Grunde liegende System begrenzt. Allerdings muss jedem aktiven Prozess ein eindeutiger Name zugewiesen werden über den ihn andere Knoten ansprechen können.

Im Weiteren werden wir von QF-Test gestartete Prozesse als Clients bezeichnen. QF-Test unterscheidet zwischen zwei Arten von Clients: Beliebigen Programmen, die dazu dienen, korrekte Randbedingungen für einen Test sicher zu stellen und SUT Clients, den eigentlichen Java Applikationen, deren GUI von QF-Test ferngesteuert wird.

Die Standardein- und -ausgabekanäle eines Clients werden in ein Terminal umgeleitet, das über das Client Menü zugänglich ist. Die Ausgabe des Client's wird auch im Protokoll des Testlaufs gespeichert.

42.7.1 Java-SUT-Client starten

Dieser Knoten stellt die direkteste und flexibelste Möglichkeit dar, einen SUT Client zu starten. Hierzu muss das Java Kommando bekannt sein, welches Ihre Applikation startet. Wird das SUT normalerweise durch ein Skript gestartet, kann es einfacher sein, einen SUT-Client starten[728] Knoten zu verwenden (vgl. Kapitel 46[1000]).

Anstatt Java direkt auszuführen, startet QF-Test das Programm `qfclient`, welches seinerseits QF-Tests eigene Hülle für das `java` Programm aufruft (vgl. Kapitel 46[1000]). Als praktischen Nebeneffekt dieser indirekten Methode können Sie das Verzeichnis[726] für alle JDK-Versionen direkt angeben.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Kommandozeile für das Programm wird aus den Attributen zusammengesetzt und der Prozess gestartet. Ein- und Ausgabe des Programms werden umgeleitet und von QF-Test übernommen.

Attribute:

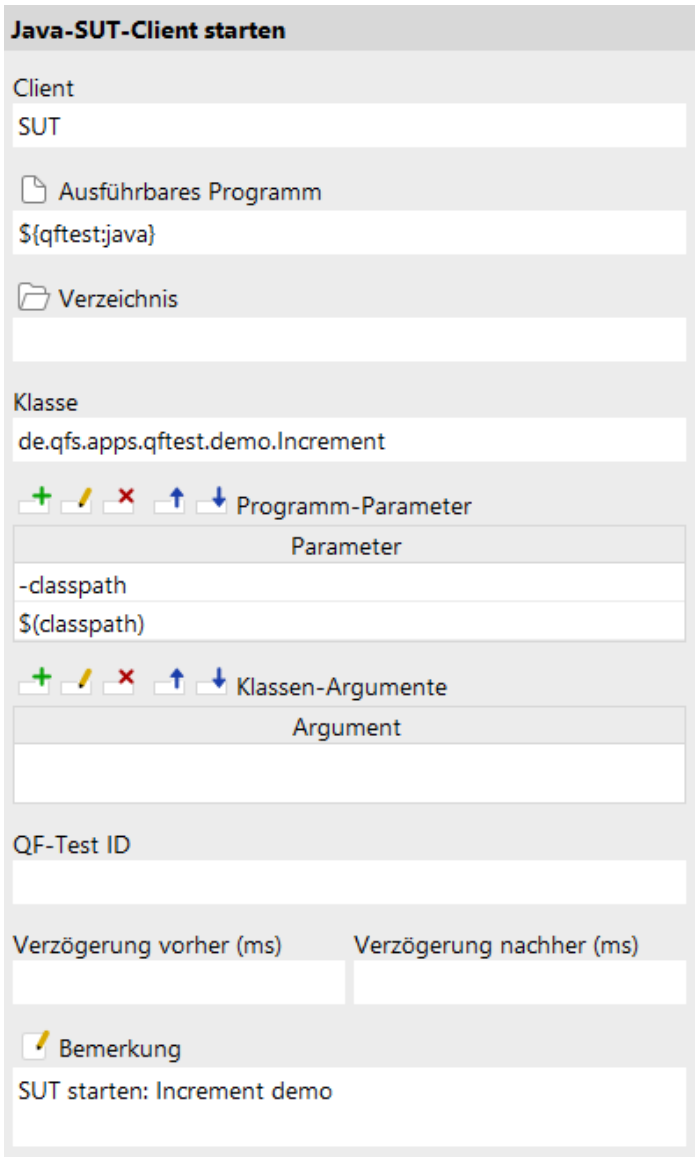


Abbildung 42.38: Java-SUT-Client starten Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange

der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Ausführbares Programm

Das ausführbare Java-Programm, das gestartet werden soll, normalerweise `java` bzw. `javaw` unter Windows. Wenn Sie eine spezielle Java-Version verwenden wollen, sollten Sie den vollen Pfad angeben.

Der "Datei auswählen" Button  öffnet einen Dialog, in dem Sie das Programm direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Verzeichnis

Hier können Sie das Arbeitsverzeichnis für das Programm festlegen. Wenn Sie nichts angeben, erbt das Programm das Arbeitsverzeichnis von QF-Test.

Der "Verzeichnis auswählen" Button  öffnet einen Dialog, in dem Sie das Verzeichnis direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Muss entweder leer, oder ein existierendes Verzeichnis sein

Klasse

Hier geben Sie den vollen Namen der Klasse des SUT an, deren `main` Methode zum Start des Programms aufgerufen werden soll. Wenn Ihre Applikation aus einem Ausführbaren jar Archiv mit dem `-jar` Parameter gestartet wird, müssen Sie dieses Attribut leer lassen.

Hinweis

QF-Test kann die `main` Methode der Klasse nur starten, wenn sowohl die Klasse selbst, als auch die Methode als `public` deklariert sind. Wenn sie beim Start Ihrer Applikation eine `IllegalAccessException` erhalten, prüfen Sie nach ob das der Fall ist.

Variabel: Ja

Einschränkungen: Gültiger Klassenname inklusive Packages

Programm-Parameter

Die Kommandozeilenargumente für das zu startende Java-Programm. Jeder

Parameter muss in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Parameters oder Sonderzeichen müssen nicht speziell behandelt werden.

Um zum Beispiel den classpath anzugeben, geben Sie in einer Zeile `-classpath` und der darauf folgenden Zeile den gewünschten Wert ein. Dabei brauchen Sie übrigens die jar Archive von QF-Test nicht zu bedenken.

Standardmäßig werden leere Parameter ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ”) übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren (535) ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Parameter verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5(20).

Variabel: Ja

Einschränkungen: Keine

Klassen-Argumente

Hiermit können Sie Argumente an die aufgerufene Klasse übergeben (für Programmierer: die Argumente der `main(String[])` Methode der Klasse). Jedes Argument muss dabei in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Arguments oder Sonderzeichen müssen nicht speziell behandelt werden.

Standardmäßig werden leere Argumente ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ”) übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren (535) ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Argumente verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5(20).

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung

bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.2 SUT-Client starten

Wenn Ihre Applikation normalerweise durch ein Shellskript oder ein spezielles Programm gestartet wird, ist dieser Knoten die einfachste Möglichkeit das SUT aus QF-Test heraus zu starten. Je nachdem wie die Applikation durch das Skript gestartet wird, können einige Modifikationen daran nötig sein. Ein deutliches Signal dafür ist es, wenn da SUT zwar wie gewünscht startet, jedoch keine Verbindung zu QF-Test hergestellt werden kann. In diesem Fall sollten Sie Kapitel 46[1000] lesen.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Kommandozeile für das Programm wird aus den Attributen zusammengesetzt und der Prozess gestartet. Ein- und Ausgabe des Programms werden umgeleitet und von QF-Test übernommen.

Attribute:

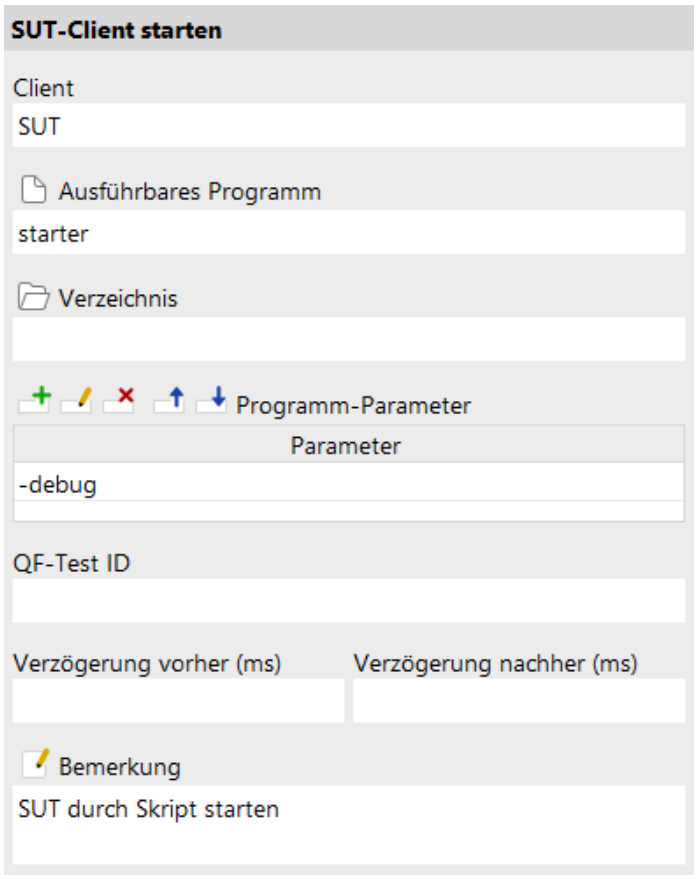


Abbildung 42.39: SUT-Client starten Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Ausführbares Programm

Das ausführbare Programm, das gestartet werden soll. Wenn sich das Programm nicht im `PATH` befindet, müssen Sie den vollen Pfad angeben.

Der ”Datei auswählen” Button öffnet einen Dialog, in dem Sie das Programm direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Verzeichnis

Hier können Sie das Arbeitsverzeichnis für das Programm festlegen. Wenn Sie nichts angeben, erbt das Programm das Arbeitsverzeichnis von QF-Test.

Der ”Verzeichnis auswählen” Button öffnet einen Dialog, in dem Sie das Verzeichnis direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Hinweis

Dieses Verzeichnis wird erst **nach** dem Starten des Programms zum Arbeitsverzeichnis. Das hat zur Folge, dass z.B. ein Skript namens `./kopiere_daten` relativ zum Arbeitsverzeichnis von QF-Test gesucht wird und nicht relativ zu diesem Verzeichnis. Erst die Pfadangaben innerhalb des Skriptes werden relativ zu diesem Verzeichnis aufgelöst.

Variabel: Ja

Einschränkungen: Muss entweder leer, oder ein existierendes Verzeichnis sein

Programm-Parameter

Die Kommandozeilenargumente für das zu startende Programm. Jeder Parameter muss in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Parameters oder Sonderzeichen müssen nicht speziell behandelt werden.

Standardmäßig werden leere Parameter ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ”) übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren (535) ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Parameter verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5(20).

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.3 Programm starten

Um während eines Tests ein beliebiges Programm zu starten, können Sie entweder diesen Knoten, oder einen Shell-Kommando ausführen[734] Knoten verwenden. Dieser Knoten ist vorzuziehen, wenn Sie mehrere, möglicherweise komplexe Parameter an das Programm übergeben wollen.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Kommandozeile für das Programm wird aus den Attributen zusammengesetzt und der Prozess gestartet. Ein- und Ausgabe des Programms werden umgeleitet und von QF-Test übernommen.

Attribute:

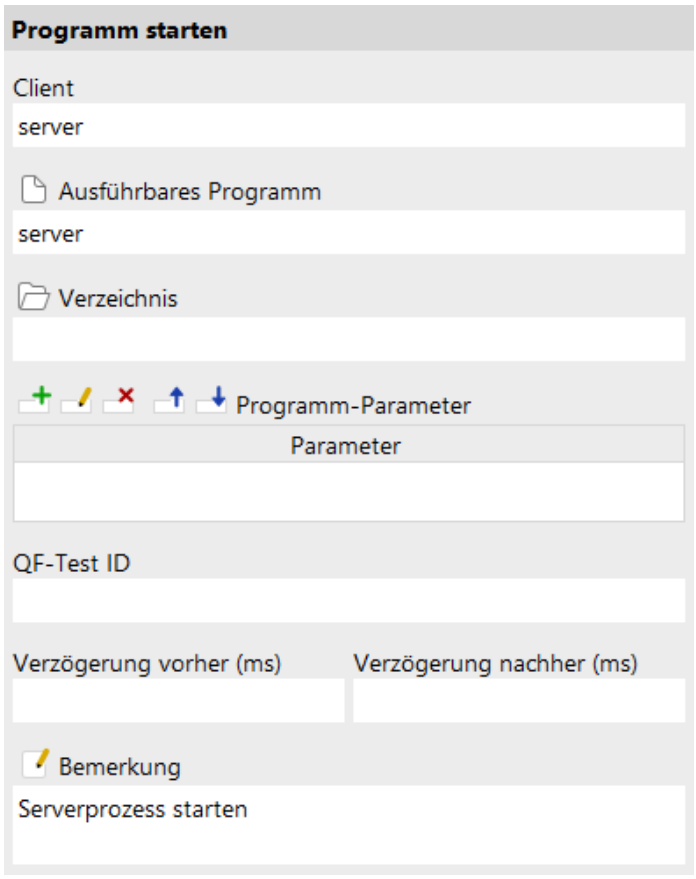


Abbildung 42.40: Programm starten Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Ausführbares Programm

Das ausführbare Programm, das gestartet werden soll. Wenn sich das Programm nicht im `PATH` befindet, müssen Sie den vollen Pfad angeben.

Der ”Datei auswählen” Button öffnet einen Dialog, in dem Sie das Programm direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Verzeichnis

Hier können Sie das Arbeitsverzeichnis für das Programm festlegen. Wenn Sie nichts angeben, erbt das Programm das Arbeitsverzeichnis von QF-Test.

Der "Verzeichnis auswählen" Button öffnet einen Dialog, in dem Sie das Verzeichnis direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Hinweis

Dieses Verzeichnis ist lediglich das Arbeitsverzeichnis des neu gestarteten Prozesses. Es hat keinen Einfluss auf das Arbeitsverzeichnis von QF-Test. Das hat zur Folge, dass z.B. ein Skript namens `./kopiere_daten` relativ zum Arbeitsverzeichnis von QF-Test gesucht wird und nicht relativ zu diesem Verzeichnis. Erst die Pfadangaben innerhalb des Programms werden relativ zum angegebenen Verzeichnis aufgelöst.

Variabel: Ja

Einschränkungen: Muss entweder leer, oder ein existierendes Verzeichnis sein

Programm-Parameter

Hier finden Sie Die Kommandozeilenargumente für das zu startende Programm. Jeder Parameter muss in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Parameters oder Sonderzeichen müssen nicht speziell behandelt werden.

Standardmäßig werden leere Parameter ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ") übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren (535) ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Parameter verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5(20).

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung

bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.4 Shell-Kommando ausführen

Hiermit lässt sich während eines Testlaufs auf einfache Weise ein Shell-Kommando ausführen. Die Shell zur Ausführung des Kommandos kann beim Start von QF-Test auf der Kommandozeile mittels `-shell <Programm>`[990] und `-shellarg <Argument>`[990] angegeben werden. Standardmäßig wird unter Linux `/bin/sh` verwendet, unter Windows `COMMAND.COM` oder `cmd.exe`.

Nach dem Starten der Shell wird sie wie jeder andere von QF-Test gestartete Prozess behandelt, d.h. Sie können sie beenden oder auf ihr Ende warten und den Exitcode auswerten.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Eine Shell wird gestartet, um das Kommando auszuführen. Ihre Ein- und Ausgabe werden umgeleitet und von QF-Test übernommen.

Attribute:

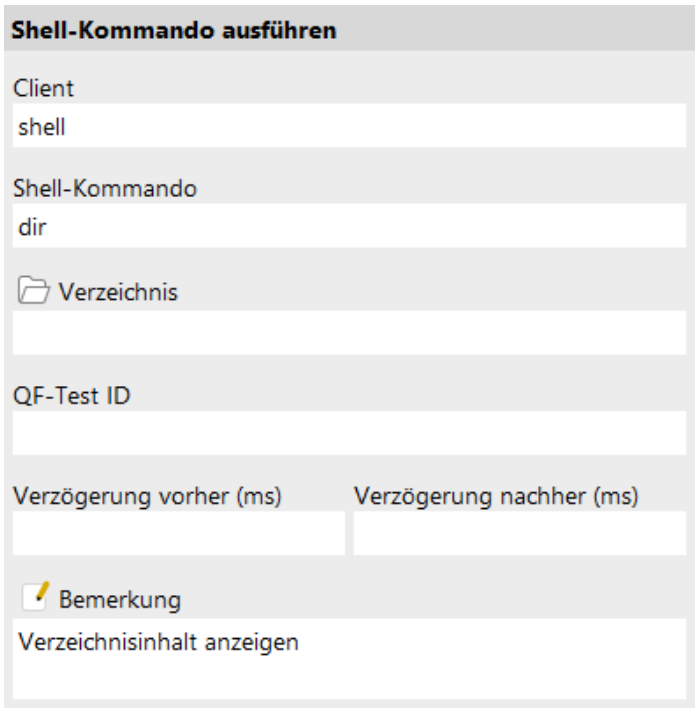


Abbildung 42.41: Attribute für Shell-Kommando ausführen

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Shell-Kommando

Das Kommando, das die Shell ausführen soll. Geben Sie es genau so wie an einem Shellprompt ein.

Windows

Unter Windows kann es zu Problemen beim Quoten von Argumenten mit Leerzeichen kommen. Wenn Sie die standard Windows Shell verwenden, schützen Sie die Argumente ganz normal mit Anführungsstrichen, z.B. `dir "C:\Program Files"`. Wenn Sie dagegen mit Hilfe des Kommandozeilenarguments `-shell <Programm>`(990) eine Linux-Shell unter Windows einsetzen, sollten Sie die Argumente mit Hochkommas schützen, also `ls 'C:/Program Files'`.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Verzeichnis

Hier können Sie das Arbeitsverzeichnis für die Shell festlegen. Wenn Sie nichts angeben, erbt die Shell das Arbeitsverzeichnis von QF-Test.

Der "Verzeichnis auswählen" Button öffnet einen Dialog, in dem Sie das Verzeichnis direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Muss entweder leer, oder ein existierendes Verzeichnis sein

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.5 Web-Engine starten

Mit diesem Knoten kann ein Browser speziell für das Testen von Web-Anwendungen gestartet werden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Kommandozeile für das Browser-Programm wird aus den Attributen zusammengesetzt und der Prozess gestartet. Ein- und Ausgabe des Programms werden umgeleitet und von QF-Test übernommen.

Attribute:

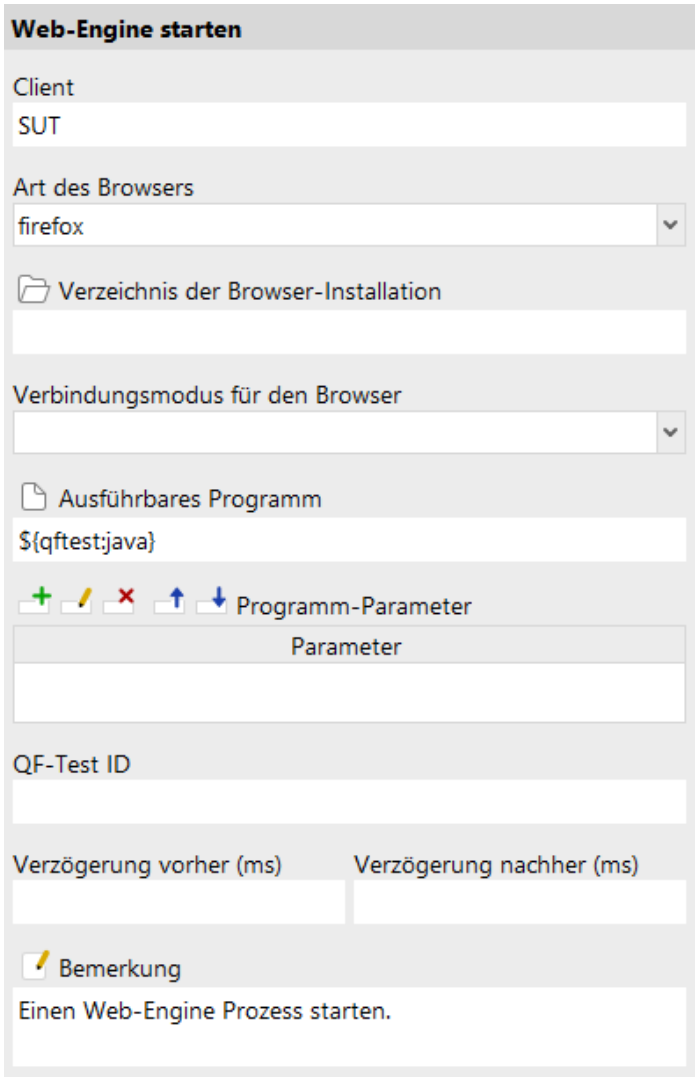


Abbildung 42.42: Web-Engine starten Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Art des Browsers

Der Typ des zu startenden Browsers. Offiziell unterstützt werden aktuell ”firefox” (oder ”mozilla”) für Mozilla Firefox, ”chrome” für Google Chrome, ”edge” für Microsoft Edge, ”opera” für Opera, ”safari” für Apple Safari, ”headless-firefox” für Mozilla Firefox ohne sichtbares Fenster, ”headless-chrome” (oder ”headless”) für Google Chrome ohne sichtbares Fenster und ”headless-edge” für Microsoft Edge ohne sichtbares Fenster.

Variabel: Ja

Einschränkungen: Erlaubte Werte sind ”ie”, ”firefox” (alternativ ”mozilla”), ”chrome”, ”edge”, ”msedge”, ”opera”, ”safari”, ”headless-firefox”, ”headless-chrome” (alternativ ”headless”) und ”headless-edge”.

Verzeichnis der Browser-Installation

Für die meisten Browser kann dieses Feld leer gelassen werden. Für Firefox mit QF-Driver Verbindungsmodus muss hier das Installations-Verzeichnis des Browsers festgelegt werden. Bei einer Verbindung über CDP-Driver oder WebDriver wird automatisch nach der Standardinstallation des angegebenen Browsers gesucht. Ist für diesen Browser die Installation mehrerer Versionen möglich, kann durch Angabe dieses Verzeichnisses gezielt eine Version gewählt werden.

Variabel: Ja

Einschränkungen: Muss entweder leer sein oder ein existierendes Verzeichnis

Verbindungsmodus für den Browser

QF-Test kann sich auf drei verschiedene Arten mit einem Browser verbinden: Im QF-Driver Verbindungsmodus durch Einbettung des Browsers in eine Java-VM, direkt über das Chrome DevTools Protokoll oder über Selenium WebDriver. Weitere Informationen hierzu sowie eine Übersicht der unterstützen Varianten finden Sie in Abschnitt 51.3(1128). Wenn verschiedene Verbindungsmodi zur Verfügung stehen, so legt dieses Attribut fest, welcher konkret gewählt wird. Die möglichen Werte sind:

Prefer QF-Driver
QF-Driver Verbindungsmodus vorziehen, falls nicht möglich CDP-Driver

oder WebDriver verwenden. Dies ist der Standard, der auch angewendet wird, wenn dieses Attribut leer bleibt.

QF-Driver only
Ausschließlich QF-Driver zulassen.

Prefer CDP-Driver
CDP-Driver Verbindungsmodus vorziehen, falls nicht möglich QF-Driver oder WebDriver verwenden.

CDP-Driver only
Ausschließlich CDP-Driver zulassen.

Prefer WebDriver
WebDriver Verbindungsmodus vorziehen, falls nicht möglich QF-Driver oder CDP-Driver verwenden.

WebDriver only
Ausschließlich WebDriver zulassen.

Variabel: Ja

Einschränkungen: Muss entweder leer sein oder einer der Werte 'Prefer QF-Driver', 'QF-Driver only', 'Prefer CDP-Driver' oder 'CDP-Driver only', 'Prefer WebDriver' oder 'WebDriver only'.

Ausführbares Programm

Das ausführbare Java-Programm, mit dem der Browser gestartet werden soll, normalerweise `java` bzw. `javaw` unter Windows. Wenn Sie eine spezielle Java-Version verwenden wollen, sollten Sie den vollen Pfad angeben.

Der Standard-Wert ist ${qftest:java}, das Java-Programm mit dem QF-Test selbst gestartet wurde.

Der "Datei auswählen" Button öffnet einen Dialog, in dem Sie das Programm direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Programm-Parameter

Die Kommandozeilenargumente für das java Programm in das der Browser eingebettet wird. Hierüber lassen sich der verfügbare Speicher (Standard ist 200MB), System Properties, Java-Debugging und ähnliches konfigurieren. Jeder Parameter muss in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Parameters oder Sonderzeichen müssen nicht speziell behandelt werden.

Standardmäßig werden leere Parameter ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ”) übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren (535) ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Parameter verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5(20).

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.6 PDF-Client starten

Über diesen Knoten wird ein zu testendes PDF-Dokument von QF-Test in einen Viewer geladen, der als Client-Prozess gestartet wird. Information zum Testen eines PDF-Dokuments finden Sie in Kapitel 18(286).

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Der Viewer wird in einem Client-Prozess gestartet und das PDF-Dokument geladen.

Attribute:

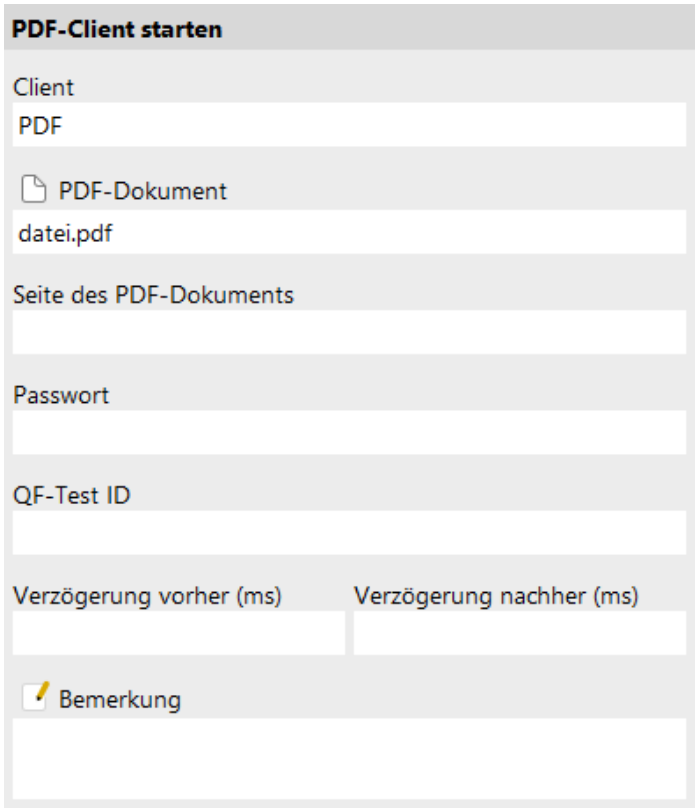


Abbildung 42.43: PDF-Client starten Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

PDF-Dokument

Das PDF-Dokument, das geöffnet werden soll. Falls ein relativer Pfad angegeben ist, wird dieser relativ zum Verzeichnis der aktuellen Testsuite aufgelöst.

Der ”Datei auswählen” Button öffnet einen Dialog, in dem Sie die Datei direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Seite des PDF-Dokuments

Hier kann eine Seite angegeben werden, die beim Start angezeigt wird.

Eine Zahl wird als Seitennummer interpretiert. Ein String in Anführungszeichen wird als Seitenname interpretiert.
Beispiel:
5 öffnet die Seite 5
”Index IV” öffnet die Seite mit dem Namen Index IV

Variabel: Ja

Einschränkungen: Muss eine Ganzzahl oder ein valider Seitenname in Anführungszeichen sein.

Passwort

Hier kann das Passwort für das PDF-Dokument angegeben werden, falls ein Passwort erforderlich ist.

Hinweis

Eventuell ist es unerwünscht, das Kennwort im Klartext in der Testsuite oder dem Protokoll abzulegen. Um das Kennwort nach Eingabe in dieses Feld zu verschlüsseln, wählen Sie nach einem Rechts-Klick Text verschlüsseln aus dem resultierenden Popupmenü. Geben Sie auf jeden Fall vor der Verschlüsselung einen Salt in der Option Salt für Verschlüsselung von Kennwörtern[532] an.

Variabel: Ja

Einschränkungen: Leer oder das gültige Passwort, falls dieses benötigt wird.

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.7 Windows-Anwendung starten

Über diesen Knoten kann man eine native Windows-Anwendung starten und sich damit verbinden. Detaillierte Informationen zum Testen nativer Windows-Anwendungen finden Sie in Kapitel 15(234).

Attribute:

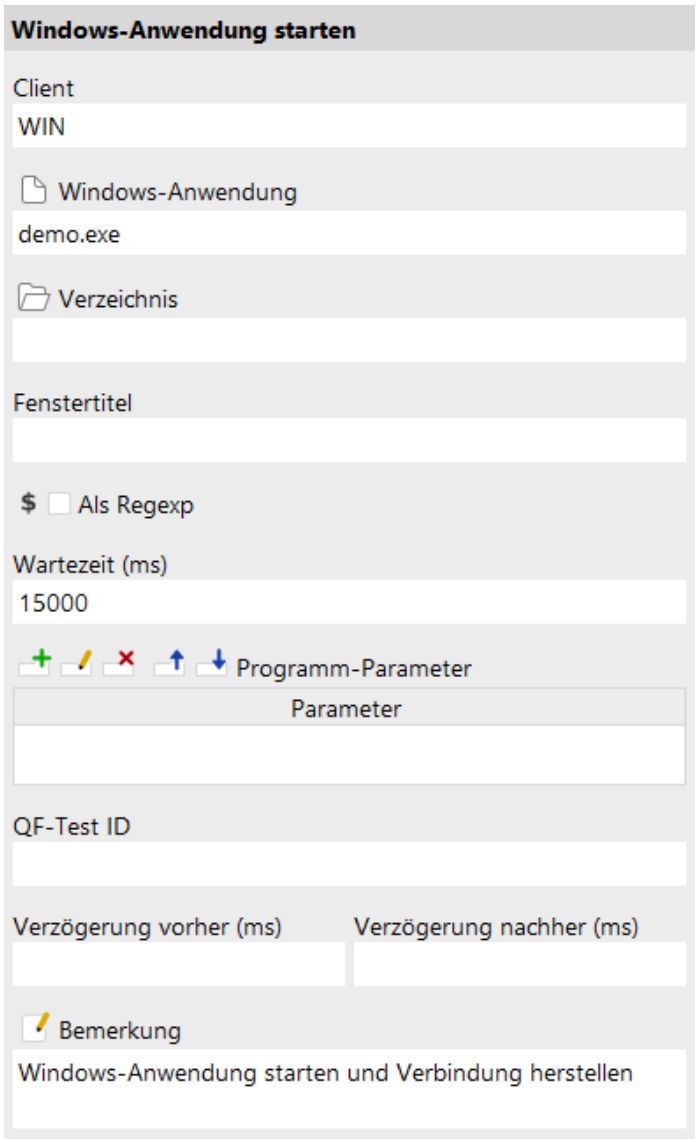


Abbildung 42.44: Windows-Anwendung starten Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Windows-Anwendung

Die zu startende ausführbare Datei der Windows-Anwendung.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Verzeichnis

Hier können Sie das Arbeitsverzeichnis für das Programm festlegen. Wenn Sie nichts angeben, erbt das Programm das Arbeitsverzeichnis von QF-Test.

Der "Verzeichnis auswählen" Button öffnet einen Dialog, in dem Sie das Verzeichnis direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Muss entweder leer, oder ein existierendes Verzeichnis sein

Fenstertitel

Der Titel des Windows Fensters, das verbunden werden soll.

Mittels Rechtsklick und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke mit '\' schützen.

Variabel: Ja

Einschränkungen: Gültige Regexp, falls Als Regexp gesetzt ist.

Als Regexp

Ist dieses Attribut gesetzt, wird der Titel des Windows Fensters als regulärer Ausdruck interpretiert (vgl. Abschnitt 49.3(1023)).

Variabel: Ja

Einschränkungen: Keine

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf.

Variabel: Ja

Einschränkungen: >= 0

Programm-Parameter

Hier finden Sie je einen Tab für folgende Parameter:

Die Kommandozeilenargumente für das zu startende Windows-Programm. Jeder Parameter muss in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Parameters oder Sonderzeichen müssen nicht speziell behandelt werden. Um Parameter an den QF-Test Win-Engine-Prozess zu übergeben, müssen diese mit "-qfengine:" beginnen.

Standardmäßig werden leere Parameter ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ”) übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren (535) ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Parameter verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5(20).

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.8 Windows-Anwendung verbinden

Über diesen Knoten kann man einer bereits laufenden nativen Windows-Anwendung verbinden. Detaillierte Informationen zum Testen nativer Windows-Anwendungen finden Sie in Kapitel 15(234).

Attribute:

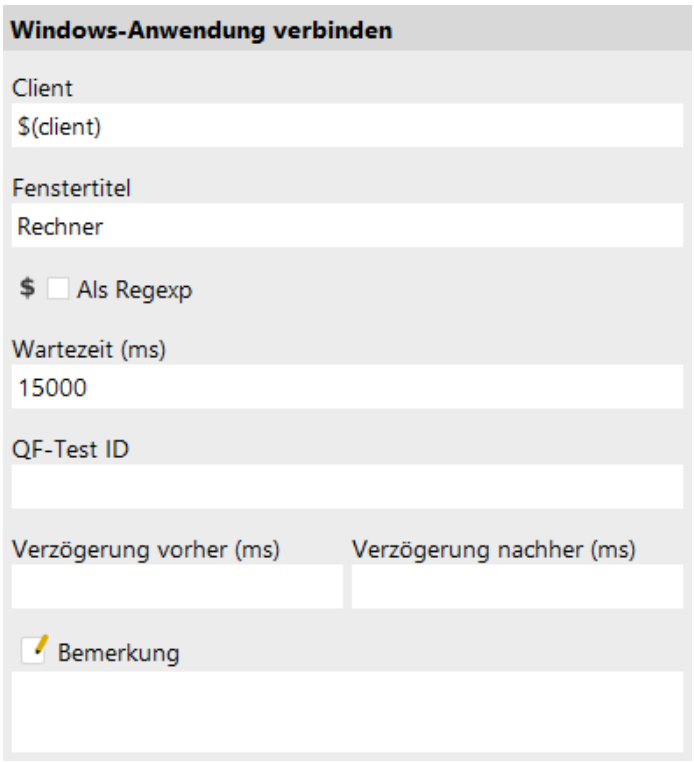


Abbildung 42.45: Windows-Anwendung verbinden Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Fenstertitel

Der Titel des Windows Fensters, das verbunden werden soll Man kann sich auch durch Angabe der Prozess-Id verbinden. In diesem Fall wird `-pid <process`

ID> eingetragen. Über -class <class name>, kann man sich durch Angabe des UI Automation Klassennames (class name) des Fensters zur Anwendung verbinden.

Mittels Rechtsklick und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke mit '\' schützen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein. Gültige Regexp, falls Als Regexp gesetzt ist.

Als Regexp

Ist dieses Attribut gesetzt, wird der Titel des Windows Fensters als regulärer Ausdruck interpretiert (vgl. Abschnitt 49.3(1023)).

Variabel: Ja

Einschränkungen: Keine

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf.

Variabel: Ja

Einschränkungen: >= 0

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.9 Android-Emulator starten

Über diesen Knoten kann man einen Android-Emulator starten und sich damit verbinden. Die Zusammenstellung der benötigten Knoten vom Emulator- bis zum App-Start mit den passenden Warten-Knoten gelingt am einfachsten über den Schnellstart Ihrer Anwendung[32]. Detaillierte Informationen zum Testen von Android-Anwendungen finden Sie in Kapitel 16[245].

Attribute:

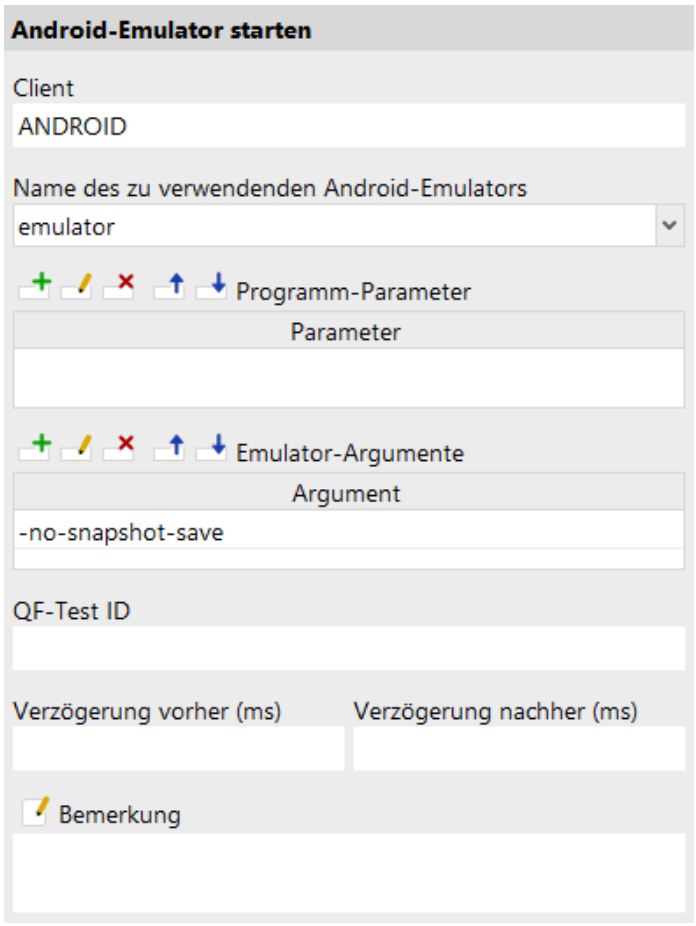


Abbildung 42.46: Android-Emulator starten Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Name des zu verwendenden Android-Emulators

Der Name des zu verwendenden Emulators wie er zum Beispiel im Android Virtual Device Manager in der Spalte 'Name' angezeigt wird (siehe Android Studio Dialog zeigt verfügbare AVDs[253]). Der Emulatorname braucht hier nicht 'fest verdrahtet' werden. Er kann auch in der Variablen `deviceName` abgespeichert und hier über die Variable referenziert werden.

Hinweis

Sollten Sie einen Emulator verwenden wollen, den Sie erst nach Start von QF-Test erstellt haben, kann es vorkommen, dass dieser noch nicht in der Drop-Down-Liste angezeigt wird. In diesem Fall können Sie den Namen direkt im Textfeld eintragen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Programm-Parameter

Die Kommunikation zwischen QF-Test und dem Emulator erfolgt über ein Java-Programm, das zusätzlich um Emulator gestartet wird und im Hintergrund läuft. Parameter für dieses Java-Programm können hier angegeben werden. Jeder Parameter muss in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Parameters oder Sonderzeichen müssen nicht speziell behandelt werden. Beispiel: `-Xmx1G`, um dem Java Programm 1 GB Speicher zu geben.

Standardmäßig werden leere Parameter ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ”) übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren [535] ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Parameter verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5[20].

Variabel: Ja

Einschränkungen: Keine

Emulator-Argumente

Die Kommandozeilenargumente für den zu startenden Emulator, zum Beispiel `-no-snapshot-save`. Jeder Parameter muss in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Parameters oder Sonderzeichen müssen nicht speziell behandelt werden.

Standardmäßig werden leere Argumente ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ”) übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren [535] ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Parameter verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5[20].

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.10 Mit Android-Gerät verbinden

Über diesen Knoten kann man sich mit einem laufenden Android-Emulator oder echtem Android-Gerät verbinden. Die Zusammenstellung der benötigten Knoten von der Verbindung zum laufenden Emulator beziehungsweise echtem Android-Gerät bis zum App-Start mit den passenden Warten-Knoten gelingt am einfachsten über den Schnellstart Ihrer Anwendung(32). Detaillierte Informationen zum Testen von Android-Anwendungen finden Sie in Kapitel 16(245).

Attribute:

Abbildung 42.47: Mit Android-Gerät verbinden Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Name des zu verwendenden Android-Gerätes

Der Name eines angeschlossenen Android-Gerätes oder des zu verwendenden Emulators. Die verfügbaren Namen werden in der Drop-Down-Liste angezeigt. Alternativ können Sie diese auch über den Menüpunkt Laufende Android-Geräte im Menü Extras herausfinden.

Der Emulator- beziehungsweise Gerätename braucht hier nicht 'fest verdrahtet' werden. Er kann auch in der Variablen `deviceName` abgespeichert und hier über die Variable referenziert werden.

Hinweis

Sollten Sie ein echtes Android-Gerät verwenden wollen, das Sie erst nach dem Start von QF-Test angeschlossen haben, kann es vorkommen, dass dieses noch nicht in der Drop-Down-Liste angezeigt wird. In diesem Fall können Sie den Namen direkt im Textfeld eintragen. Gleiches gilt für einen Emulator, den Sie erst nach Start von QF-Test erstellt haben.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Programm-Parameter

Die Kommunikation zwischen QF-Test und dem Android-Emulator beziehungsweise echtem Gerät erfolgt über ein Java-Programm, das zusätzlich gestartet wird und im Hintergrund läuft. Parameter für dieses Java-Programm können hier angegeben werden. Jeder Parameter muss in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Parameters oder Sonderzeichen müssen nicht speziell behandelt werden. Beispiel: `-Xmx1G`, um dem Java Programm 1 GB Speicher zu geben.

Standardmäßig werden leere Parameter ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ”) übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren (535) ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Parameter verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5(20).

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.11 Mit iOS-Gerät verbinden

Dieser Knoten verbindet QF-Test mit einem angeschlossenen oder einem simulierten iOS-Gerät. Falls notwendig, wird der Simulator ebenfalls gestartet. In QF-Test Vorbereitung Sequenz für iOS Tests[277] ist beschrieben, wie Sie eine Sequenz erstellen können, die die Voraussetzungen für das Testsystem überprüft, die Verbindung zu einem iOS-Gerät herstellt und eine App startet. Diese Sequenz enthält auch die notwendigen Warteknoten.

Detaillierte Informationen zum Testen von iOS-Apps finden Sie in Kapitel 17[269].

Attribute:

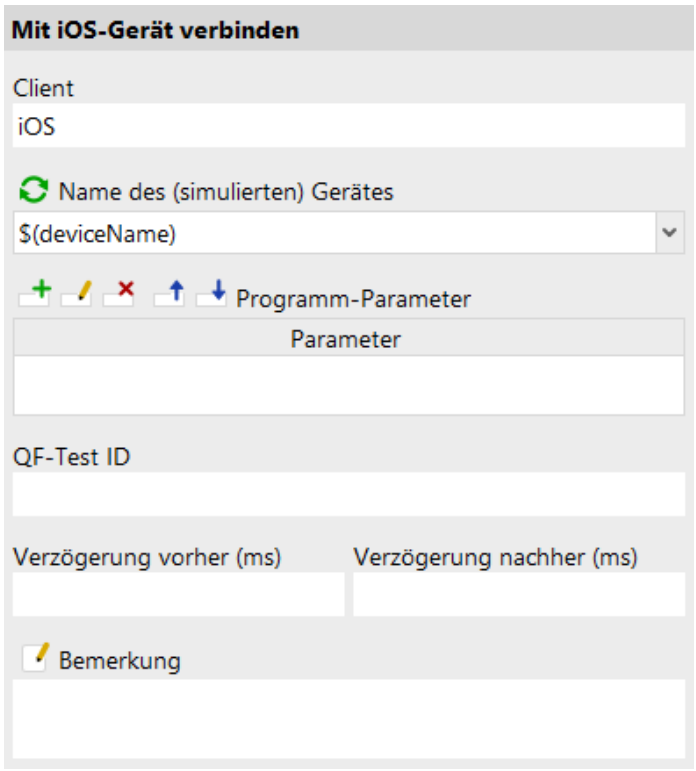


Abbildung 42.48: Mit iOS-Gerät verbinden Attribut

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Name des (simulierten) iOS-Gerätes

Der Name eines iOS-Geräts, das an den Rechner angeschlossen ist, oder ein simuliertes iOS-Gerät. Von den verfügbaren Geräten wird dasjenige ausgewählt, dessen Name am besten mit dem hier eingetragenen übereinstimmt. Um also einen Lauf auf einem beliebigen (simulieren) iPhone auszuführen, genügt es, hier einfach den Text iPhone einzutragen.

Wenn Sie den Namen hier nicht fest verdrahten wollen, können Sie stattdessen die Variable `deviceName` eintragen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Programm-Parameter

QF-Test kommuniziert mit dem iOS-Gerät über eine Java-basierte Controller-Applikation. Hier können Sie die Parameter für dieses Java-Programm angeben. Jeder Parameter muss in seiner eigenen Zeile stehen, Leerzeichen innerhalb eines Parameters oder Sonderzeichen müssen nicht speziell behandelt werden. Beispiel: `-Xmx1G` für die Zuweisung von 1 GB Speicher für das Java-Programm.

Standardmäßig werden leere Parameter ignoriert. Falls Sie explizit ein leeres Kommandozeilenargument (entsprechend ”) übergeben wollen, können Sie die Option Leere Argument-Zeilen beim Start des Clients ignorieren (535) ausschalten.

Expandiert der Wert in einer Zeile aus einer Variable zu einer Liste oder einem Array, so werden alle Elemente des Objektes als einzelne Parameter verwendet.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt Abschnitt 2.2.5(20).

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.12 Warten auf Client

Bei seiner Ausführung stellt dieser Knoten sicher, dass eine Verbindung zu einem bestimmten Client besteht. Ist dies nicht der Fall, wartet er eine gewisse Zeit und wirft im Fehlerfall eine ClientNotConnectedException[964]. Das Resultat des Knotens kann auch in einer Variable mittels des Attributes Variable für Ergebnis gesetzt werden. Mit Ausschalten des Attributes Im Fehlerfall Exception werfen kann das Werfen der Exceptions unterdrückt werden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: QF-Test wartet, bis der Java-Client des SUT mit dem entsprechenden Namen eine RMI Verbindung herstellt oder das Timeout abgelaufen ist.

Attribute:

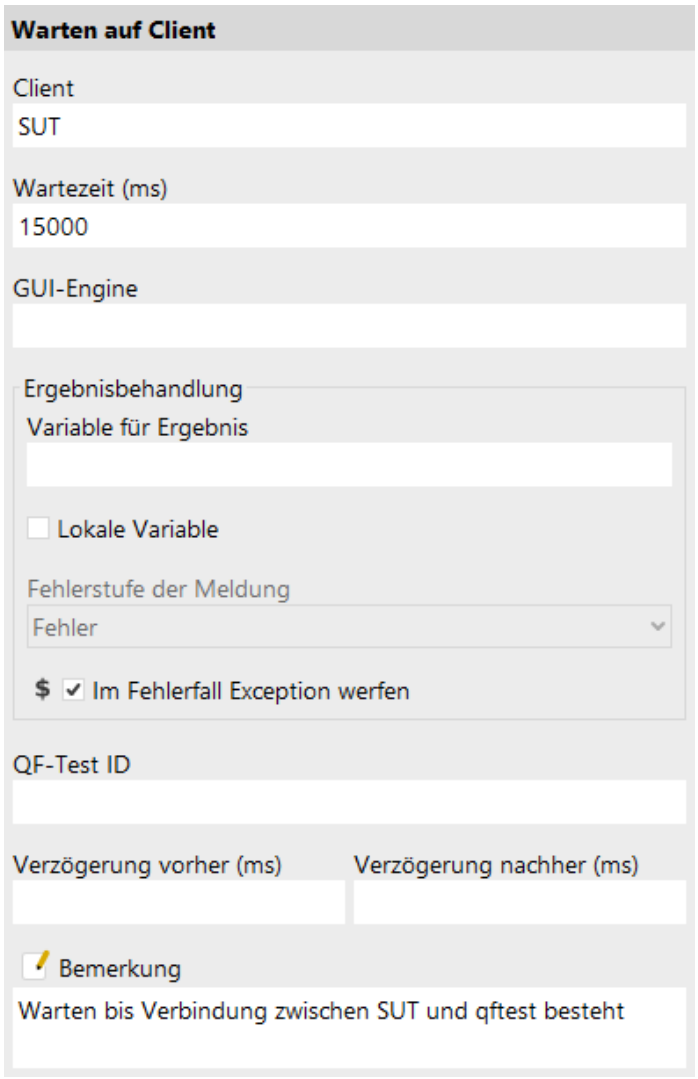


Abbildung 42.49: Warten auf Client Attribute

Client

Der Name des Java-Clients auf dessen Verbindung gewartet wird.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf.

Variabel: Ja

Einschränkungen: >= 0

GUI-Engine

Die GUI-Engine auf die gewartet werden soll. Nur relevant für SUTs mit mehr als einer GUI-Engine wie in Kapitel 45(998) beschrieben.

Variabel: Ja

Einschränkungen: Siehe Kapitel 45(998)

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur(672), Abhängigkeit(630) oder Testfall(599) Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6(116).

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`(963) geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.13 Warten auf Mobil-Gerät

Bei seiner Ausführung stellt dieser Knoten sicher, dass eine Verbindung zu einem bestimmten (virtuelle/simulierte) mobilen Gerät besteht. Ist dies nicht der Fall, wartet er eine gewisse Zeit und wirft nach Ablauf der Zeit eine `ClientNotConnectedException`(964). Falls bereits vor Ablauf der Zeit erkannt wird, dass die Verbindung nicht zustande kommt, wird eine `ConnectionFailureException`(964) geworfen.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: QF-Test wartet, bis das (virtuelle/simulierte) mobile Gerät mit dem entsprechenden Namen eine Verbindung herstellt oder das Timeout abgelaufen ist.

Attribute:

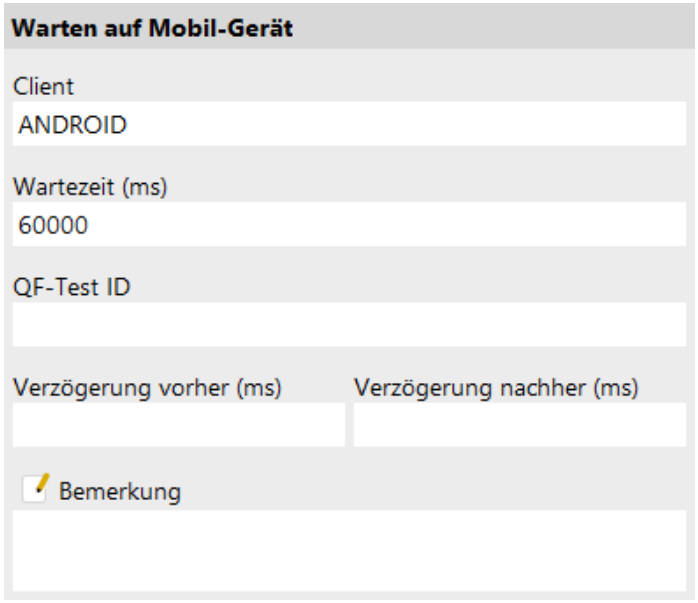


Abbildung 42.50: Warten auf Mobil-Gerät-Attribute

Client

Der Name des Android- oder iOS-Clients, auf dessen Verbindung gewartet wird.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf.

Variabel: Ja

Einschränkungen: >= 0

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.14 Browser-Fenster öffnen

Dieser Knoten öffnet eine Webseite in einem Browser-Fenster im laufenden Prozess. Der Prozess sollte vorher von einem Web-Engine starten Knoten gestartet werden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Es wird eine Webseite im gestarteten Web-Engine Prozess geöffnet.

Attribute:

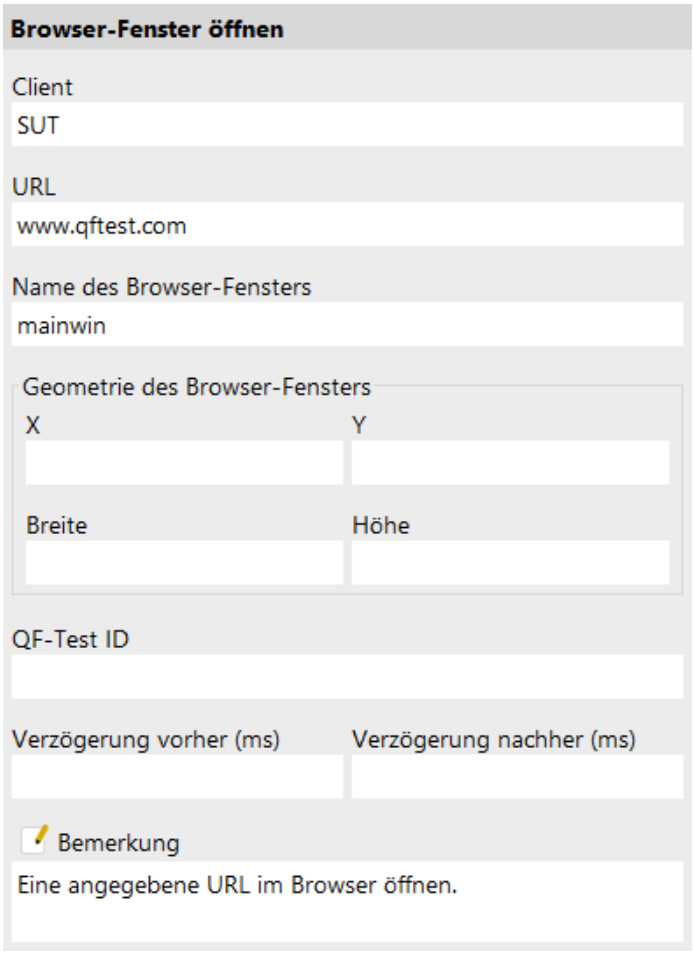


Abbildung 42.51: Browser-Fenster öffnen Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

URL

Die URL der Webseite, die im Browser dargestellt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Name des Browser-Fensters

Dieses Attribut können Sie ignorieren, sofern Sie nicht eine Web-Anwendung mit mehreren offenen Browser-Fenstern testen müssen, die das selbe Dokument darstellen. In diesem Fall kann das Attribut Name des Browser-Fensters dazu dienen, die Browser-Fenster zu unterscheiden. Hier können Sie den Namen des zu öffnenden Browser-Fensters festlegen. Der Umgang mit mehreren Fenstern wird in FAQ 25 genauer erläutert.

Variabel: Ja

Einschränkungen: Keine

Geometrie des Browser-Fensters

Diese optionalen Attribute für die X/Y Koordinate, Breite und Höhe können zur Definition der Geometrie des zu öffnenden Browser-Fensters verwendet werden.

Variabel: Ja

Einschränkungen: Breite und Höhe dürfen nicht negativ sein.

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.15 Mobile-App starten

Über diesen Knoten kann man eine mobile Applikation installieren und starten. Die Zusammenstellung der benötigten Knoten vom Emulator/Simulator-Start - oder alternativ der Verbindung mit einem laufenden Android-Emulator, einem iOS-Simulator oder einem echtem Gerät - bis zum App-Start mit den passenden Warten-Knoten gelingt am einfachsten über den Schnellstart Ihrer Anwendung[32]. Detaillierte Informationen zum Testen von mobilen Anwendungen finden Sie in Kapitel 16[245].

Attribute:

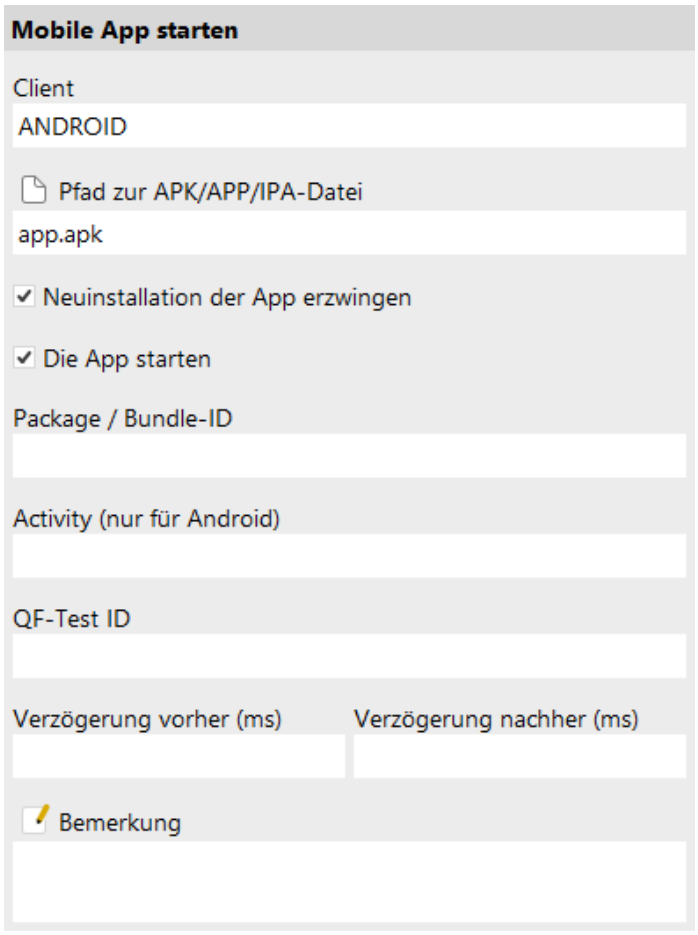


Abbildung 42.52: Mobile-App starten Attribute

Client

Dieser Name identifiziert den Client Prozess und muss eindeutig sein, so lange der Prozess läuft. Andere Knoten identifizieren den Client an Hand dieses Namens.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Pfad zur APK/APP/IPA-Datei

Der Pfad zu der zu verwendenden Android- oder iOS-App. Um einen Pfad relativ zur Testsuite anzugeben, nutzen Sie am besten die Variable `${qftest:suite.dir}` verwenden, siehe Abschnitt 6.8(127). Beispiel: `${qftest:suite.dir}/apps/myapp.apk`.

Falls die App vorinstalliert ist und die .apk/.app/.ipa-Datei nicht verfügbar ist, kann die App alternativ über die Attribute Package / Bundle-ID und Activity (nur für Android) gestartet werden.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Neuinstallation der App erzwingen

Wenn Sie das Attribut selektieren, wird die App auf jeden Fall neu installiert, egal, ob bereits eine Installation der App auf dem Emulator/Simulator oder dem echten Gerät vorhanden ist.

Hinweis

Diese Aktion löscht nicht zwingend vorhandene Einstellungen und Daten der App. Dies ist abhängig davon, wo die Daten und Einstellungen der App gespeichert werden.

Variabel: Nein

Einschränkungen: Keine

Die App starten

Attribut selektieren, um die Applikation zu starten.

Variabel: Nein

Einschränkungen: Keine

Package / Bundle-ID

Der Package-Name der zu startenden Android-App oder die BundleId der zu startenden iOS-App. Wird nur für den Sonderfall benötigt, dass die .apk/.app/.ipa-Datei nicht im Attribut Pfad zur APK/APP/IPA-Datei angegeben werden kann. Zusätzlich muss für Android dann auch das Attribut Activity (nur für Android) gesetzt werden. Das Package kann über die Android Debug Bridge ermittelt werden. Alternativ können zunächst nur den Emulator starten oder sich mit einem echten Gerät oder laufenden Emulator verbinden. Wenn Sie nun die App manuell starten, können Sie eine Komponente aufnehmen. Dann wird das Package in den Attributen des obersten Knotens der Komponentenhierarchie abgespeichert.

Variabel: Ja

Einschränkungen: Keine

Activity (nur für Android)

Der Activity-Name der zu startenden Android-App. Wird nur für den Sonderfall benötigt, dass die .apk-Datei nicht im Attribut Pfad zur APK/APP/IPA-Datei angegeben werden kann. Zusätzlich muss dann auch das Attribut Package /

Bundle-ID gesetzt werden. Die Activity kann über die Android Debug Bridge ermittelt werden.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.16 Programm beenden

Hiermit kann ein Prozess, der von QF-Test gestartet wurde, beendet werden. Sollte sich der Prozess bereits selbst beendet haben, ändert sich nichts. Andernfalls versucht QF-Test im Fall eines SUT Clients zunächst, die Applikation durch Aufruf der Java-Methode `System.exit(-1)` zu beenden. Ist der Client kein SUT Client oder anschließend immer noch nicht beendet, wird der Prozess ”abgeschossen” und erhält einen Exitcode von -1.

Windows

Achtung: Unter Windows werden Kindprozesse nicht beendet, wenn der Vaterprozess abgeschossen wird. Wie in Kapitel 46(1000) beschrieben, startet QF-Test das SUT über einige Umwege. Wenn das SUT sich nicht mit `System.exit(-1)` beenden lässt, kann das zu ”herrenlosen” Prozessen führen, die nach und nach das System blockieren.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Schießt den Prozess ab, der unter dem entsprechenden Namen gestartet wurde.

Attribute:

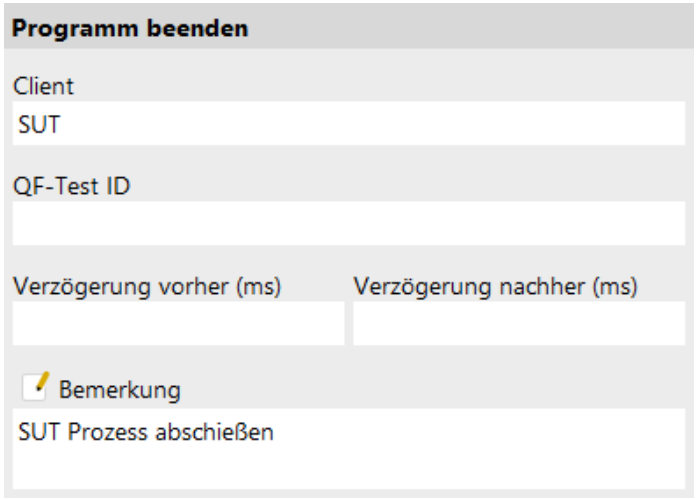


Abbildung 42.53: Programm beenden Attribute

Client

Der Name unter dem der Prozess, der abgeschossen werden soll, gestartet wurde.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.7.17 Warten auf Programmende

Hiermit kann auf das Ende eines Prozesses, der von QF-Test gestartet wurde, gewartet werden. Terminiert der Prozess nicht innerhalb des vorgegebenen Timeouts, wird eine `ClientNotTerminatedException`(965) ausgelöst. Andernfalls wird der Exitcode des Prozesses ermittelt und kann mit einem vorgegebenen Wert verglichen werden. Das Resultat des Knotens kann auch in einer Variable mittels des Attributes Variable für Ergebnis gesetzt werden. Mit Ausschalten des Attributes Im Fehlerfall Exception werfen kann das Werfen der Exceptions unterdrückt werden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Wartet auf das Ende des Prozesses, der unter dem entsprechenden Namen gestartet wurde und prüft dessen Exitcode.

Attribute:

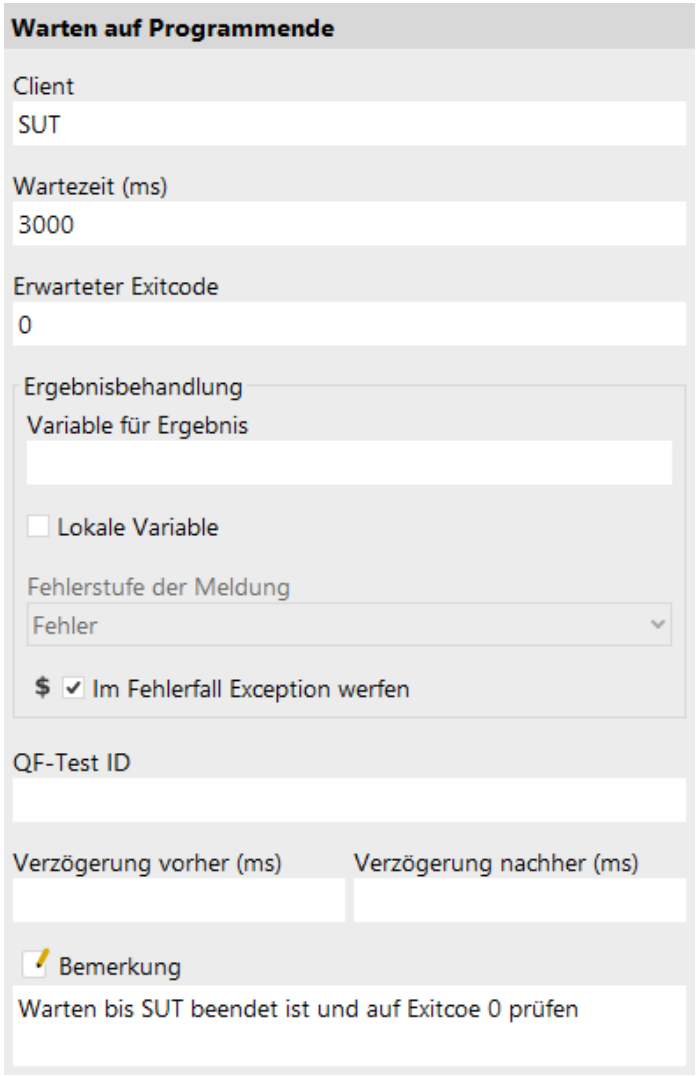


Abbildung 42.54: Warten auf Programmende Attribute

Client

Der Name unter dem der Prozess, auf dessen Ende gewartet werden soll, gestartet wurde.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf.

Variabel: Ja

Einschränkungen: >= 0

Erwarteter Exitcode

Ist dieses Attribut gesetzt, wird nach dem Ende des Prozesses dessen Exitcode ausgewertet und mit diesem Wert verglichen. Die Art des Vergleichs wird durch einen der vier Operatoren **==**, **!=**, **<** und **>** festgelegt, die dem Wert vorangestellt werden können. Ohne Operator wird auf Gleichheit geprüft. Scheitert die Prüfung, wird eine `TestException` ausgelöst.

Beispiele: Wird als Attributwert **0** angegeben, so führt jeder andere Exitcode als 0 zu einem Fehler. Dies entspricht der Angabe **==0**. Bei einem Wert von **>0** löst entsprechend jeder Exitcode kleiner oder gleich 0 eine Exception aus.

Variabel: Ja

Einschränkungen: Siehe oben

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur[672], Abhängigkeit[630] oder Testfall[599] Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6[116].

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren[593] kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`[963] geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt

hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.8 Events

In diesem Abschnitt sind alle Arten von Knoten zusammengefasst, die Aktionen im SUT auslösen. Neben den reinen Java-Events gibt es auch Pseudoevents mit besonderer Bedeutung.

42.8.1 Mausevent

Mausevents simulieren Mausbewegungen und -klicks, sowie Drag&Drop Operationen.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Koordinaten und weiteren Attribute des Events werden zusammen mit den Daten der Zielkomponente an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT die passende Komponente und rechnet die Koordinaten entsprechend um. Der so gebildete Event wird dann im SUT ausgelöst.

Attribute:

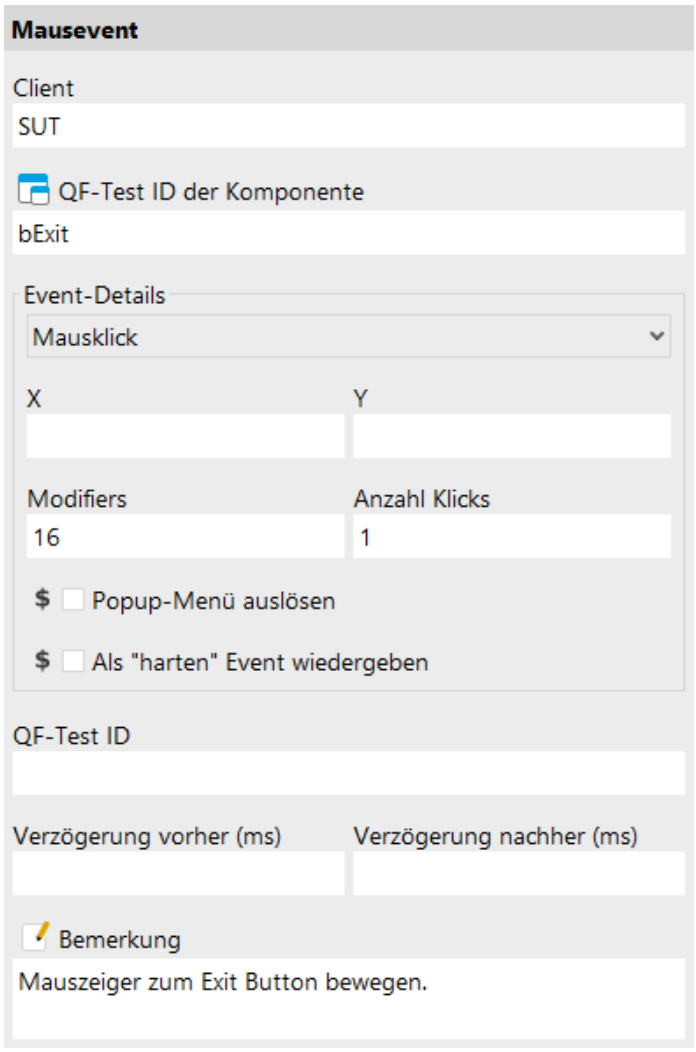


Abbildung 42.55: Mausevent Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, an den der Event geschickt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Event bezieht.

Der ”Komponente auswählen” Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5[47]) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Event

In dieser *ComboBox* können Sie die Art des Events festlegen. `MOUSE_MOVED`, `MOUSE_PRESSED`, `MOUSE_RELEASED`, `MOUSE_CLICKED` und `MOUSE_DRAGGED` entsprechen den Event IDs der Java-Klasse `MouseEvent`.

Der abstrakte ’Mausklick’-Event fasst die Events `MOUSE_MOVED`, `MOUSE_PRESSED`, `MOUSE_RELEASED` und `MOUSE_CLICKED` zusammen, d.h. bei der Wiedergabe wird der Pseudoevent durch vier Einzelevents simuliert. Hierdurch verbessert sich die Übersicht und die Bearbeitung der Suite wird vereinfacht.

4.1.3+ Der spezielle ’Doppelklick’-Event ist noch umfassender und beinhaltet alle Einzelevents, um einen kompletten Doppelklick zu simulieren.

`MOUSE_DRAG_FROM`, `MOUSE_DRAG_OVER` und `MOUSE_DROP_TO` dienen zur Simulation von Drag&Drop im SUT. Näheres zu diesem Thema finden Sie in Abschnitt 49.1[1021].

Variabel: Nein

Einschränkungen: Keine

X/Y

Dies sind die Koordinaten des *MouseEvents*. Sie sind relativ zur linken oberen Ecke des Fensters[919], der Komponente[930] oder des Elements[936], auf das sich der Event bezieht. Sie können negativ sein, z.B. um einen Ast in einem `JTree` mittels eines Klicks auf seinen Schalter ein- oder auszuklappen.

Meist sind die genauen Koordinaten für einen *MouseEvent* egal, so lange sie innerhalb des Ziels liegen. In solchen Fällen sollten Sie die Werte für X und Y leer lassen woraufhin QF-Test die Mitte des Ziels ansteuert. Wenn möglich belässt QF-Test die Werte auch bei der Aufnahme leer, sofern die Option Mausevents ohne Koordinaten aufnehmen wo möglich (510) gesetzt ist.

Variabel: Ja

Einschränkungen: Gültige Zahl oder leer

Modifiers

Bei Tastatur- und Mausevents wird mit diesem Attribut der Zustand der Tasten Shift, Strg und Alt sowie der Maustasten festgelegt. Jeder Taste ist dabei ein Wert zugeordnet, Kombinationen erhält man durch Addition.

Die Werte im Einzelnen:

Wert	Taste
1	Shift
2	Strg
4	Meta bzw. rechte Maustaste (Langer Klick bei Android und iOS)
8	Alt bzw. mittlere Maustaste
16	Linke Maustaste

Tabelle 42.16: Modifier Werte

Variabel: Ja

Einschränkungen: Gültige Zahl

Anzahl Klicks

Durch dieses Attribut unterscheidet Java, ob es sich bei einem Mausklick um einen einfachen, oder einen Doppelklick handelt.

Variabel: Ja

Einschränkungen: Gültige Zahl

Popup-Menü auslösen

Ist dieses Attribut gesetzt, kann der Event das Öffnen eines *PopupMenus* auslösen. Dieser etwas eigenwillige Mechanismus dient in Java dazu, die unterschiedlichen Konventionen zum Aufruf eines *PopupMenus* auf verschiedenen *native Systems* zu unterstützen.

Variabel: Ja

Einschränkungen: Keine

Als ”harten” Event wiedergeben

Wenn dieses Attribut gesetzt ist, wird der Event als ”harter” Event abgespielt. Dies bedeutet, dass er als echter Systemevent wiedergegeben wird, inklusive Mausbewegung und nicht nur als ”weicher” Event in die `TestEventQueue` eingespeist wird. ”Weiche” Events sind typischerweise besser, weil sie eine Beeinflussung durch gleichzeitige Mausaktionen des Benutzers vermeiden. Auch sind sie zuverlässiger gegenüber Beeinträchtigungen durch überlappende Fenster. Trotzdem gibt es spezielle Situationen, in denen ”harte” Events hilfreich sind.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.8.2 Tastaturevent

Tastaturevents spielen dem SUT Eingaben über die Tastatur vor. Mehrere Tastatureingaben in Folge können automatisch zu einer Texteingabe(784) zusammengefasst werden.

Ein `InputMethodEvent` ist eine spezielle Form von Tastaturevent, der die Eingabe von Zeichen aus internationalen Zeichensätzen erlaubt. QF-Test unterstützt `InputMethodEvents` nicht direkt, sondern konvertiert Events vom Typ `INPUT_METHOD_TEXT_CHANGED` automatisch in Tastaturevents vom Typ `KEY_TYPED`.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Die Tastaturcodes des Events werden zusammen mit den Daten der Zielkomponente an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT die passende Komponente. Der so gebildete Event wird dann im SUT ausgelöst.

Attribute:

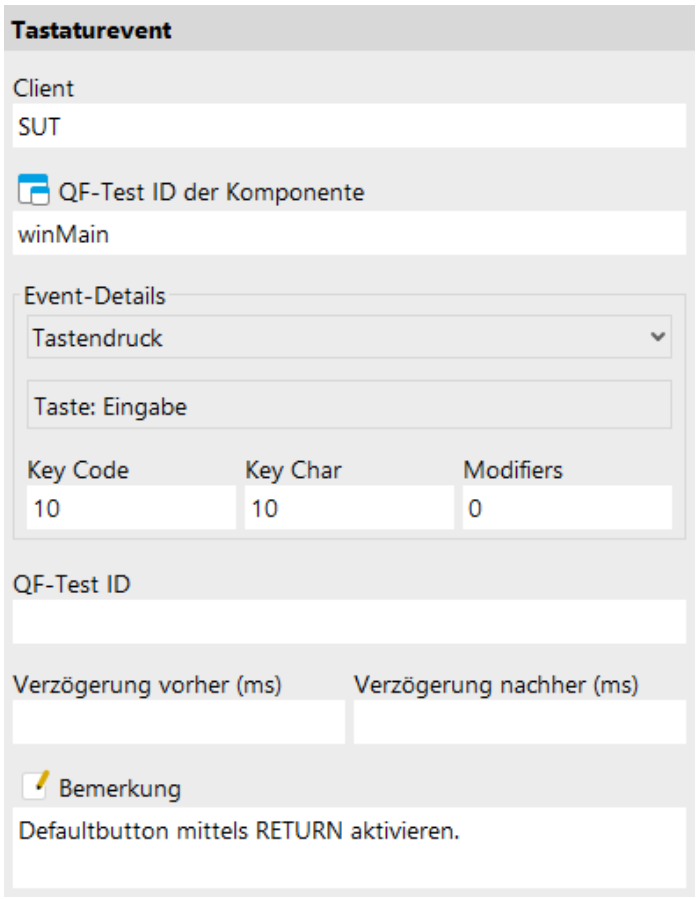


Abbildung 42.56: Tastaturevent Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, an den der Event geschickt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Event bezieht.

Der "Komponente auswählen" Button öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw.

Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Event

In dieser *ComboBox* können Sie die Art des Events festlegen. Die möglichen Werte sind `KEY_PRESSED`, `KEY_TYPED` und `KEY_RELEASED`, entsprechend den Event IDs der Java Klasse `KeyEvent`.

Zusätzlich können Sie den Pseudoevent 'Tastendruck' wählen, der bei der Wiedergabe durch eine entsprechende Abfolge von `KEY_PRESSED`, `KEY_TYPED` und `KEY_RELEASED` Events simuliert wird, wobei der `KEY_TYPED` Event nur generiert wird, wenn keyChar[783] nicht 65535 ist, d.h. wenn es sich nicht um eine Sonder- oder Funktionstaste handelt.

Variabel: Nein

Einschränkungen: Keine

Taste

Hiermit können Sie die Taste für den KeyEvent direkt festlegen und sich damit die Eingabe von KeyCode[782], KeyChar[783] und Modifiers[783] sparen. Wenn diese Komponente den Eingabefokus besitzt, werden die Werte der zuletzt gedrückten Taste übernommen. Für KEY_TYPED Events wird der keyCode automatisch auf 0 gesetzt.

Dabei ist eine kleine Besonderheiten zu beachten: Sie können die Tabulator Taste nicht auf diesem Weg wählen, da sie zum Wechsel des Fokus führt. KeyCode und KeyChar für diese Taste sind jeweils 9.

Variabel: Nein

Einschränkungen: Keine

Key Code

Dieser Code entspricht der Taste, die gedrückt oder losgelassen wurde. Bei einem `KEY_TYPED` Event ist dieser Wert 0.

Variabel: Ja

Einschränkungen: Gültige Zahl

Key Char

Dies ist das Zeichen, das durch den letzten Tastendruck gebildet wurde. Hierin ist der Zustand der Shifttaste berücksichtigt. Sonder- und Funktionstasten haben keinen individuellen Key Char, sondern alle den Wert 65535.

Variabel: Ja

Einschränkungen: Gültige Zahl

Modifiers

Bei Tastatur- und Mausevents wird mit diesem Attribut der Zustand der Tasten Shift, Strg und Alt sowie der Maustasten festgelegt. Jeder Taste ist dabei ein Wert zugeordnet, Kombinationen erhält man durch Addition.

Die Werte im Einzelnen:

Wert	Taste
1	Shift
2	Strg
4	Meta bzw. rechte Maustaste (Langer Klick bei Android und iOS)
8	Alt bzw. mittlere Maustaste
16	Linke Maustaste

Tabelle 42.17: Modifier Werte

Variabel: Ja

Einschränkungen: Gültige Zahl

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.8.3 Texteingabe

Hierbei handelt es sich um einen Pseudoevent, mit dessen Hilfe die Eingabe von Text im SUT vereinfacht wird. Anstatt für jedes Zeichen des einzugebenden Textes drei Tastaturevent Knoten anzulegen (`KEY_PRESSED`, `KEY_TYPED` und `KEY_RELEASED`), können Sie beinahe den selben Effekt mit einem Texteingabe Knoten erreichen.

Hierbei macht sich QF-Test zu Nutze, dass Java für Texteingaben in ein Feld nur die `KEY_TYPED` Events auswertet. Beim Abspielen einer Texteingabe werden im SUT daher nur `KEY_TYPED` Events simuliert, keine `KEY_PRESSED` oder `KEY_RELEASED` Events.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Zeichenfolge des Events wird zusammen mit den Daten der Zielkomponente an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT die passende Komponente und löst für jedes Zeichen einen `KEY_TYPED` Event im SUT aus.

Attribute:

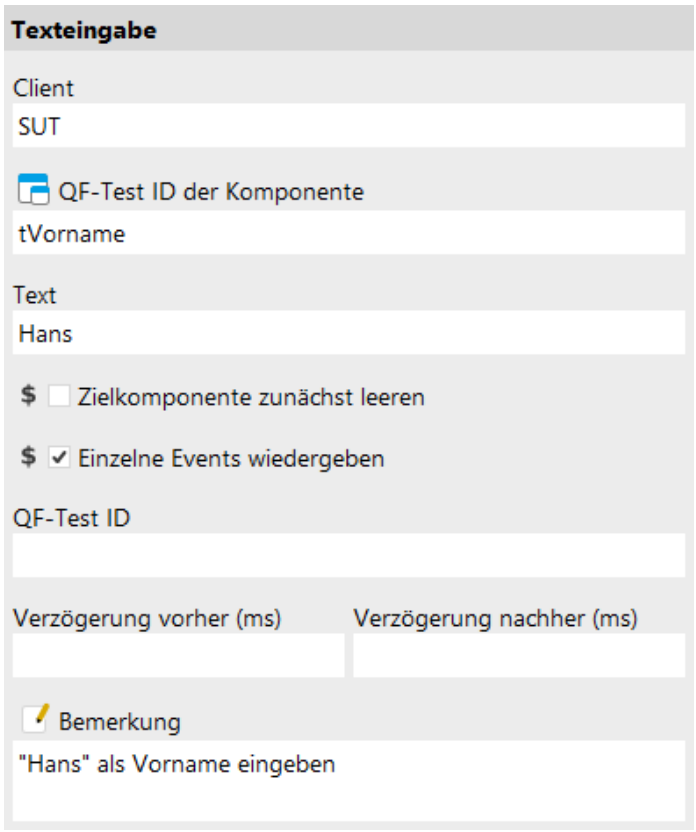


Abbildung 42.57: Texteingabe Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, an den der Event geschickt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Event bezieht.

Der "Komponente auswählen" Button öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1(359)). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9(92)). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID(81) und Komponente-Knoten versus SmartID(51).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Text

Der Text, der im SUT in ein Feld eingegeben werden soll.

Hinweis

Bei der Eingabe von Text in ein Kennwort-Feld ist es eventuell unerwünscht, das Kennwort im Klartext in der Testsuite oder dem Protokoll abzulegen. Um das Kennwort nach Eingabe in dieses Feld zu verschlüsseln, wählen Sie nach einem Rechts-Klick Text verschlüsseln aus dem resultierenden Popupmenü. Geben Sie auf jeden Fall vor der Verschlüsselung einen Salt in der Option Salt für Verschlüsselung von Kennwörtern(532) an.

Variabel: Ja

Einschränkungen: Keine Zeilenumbrüche möglich.

Zielkomponente zunächst leeren

Ist dieses Attribut gesetzt und handelt es sich bei der Zielkomponente um ein Textfeld, wird der Inhalt des Feldes automatisch gelöscht, bevor der neue Text eingegeben wird.

Variabel: Ja

Einschränkungen: Keine

Einzelne Events wiedergeben

Handelt es sich bei der Zielkomponente um ein Textfeld, kann der Text auch direkt über die API der Komponente eingegeben werden, ohne einzelne Events zu simulieren. Dies ist wesentlich schneller, hat aber den Nachteil, dass eventuell vorhandene KeyListener des SUT nicht benachrichtigt werden. Ist dieses Attribut gesetzt, werden für jedes einzelne Zeichen Events generiert.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.8.4 Fensterevent

WindowEvents sind nur begrenzt für eine Testsuite interessant, da die meisten von ihnen als Folge von Programmabläufen im Client generiert werden und nicht vom Anwender direkt. Eine Ausnahme ist der Event `WINDOW_CLOSING`, der dem Schließen eines Fensters durch den Anwender entspricht. Außerdem können auch `WINDOW_ICONIFIED` und `WINDOW_DEICONIFIED` Events ausgeführt werden.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Der Event wird zusammen mit den Daten des Zielfensters an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT das passende Fenster und löst den Event im SUT aus.

Attribute:

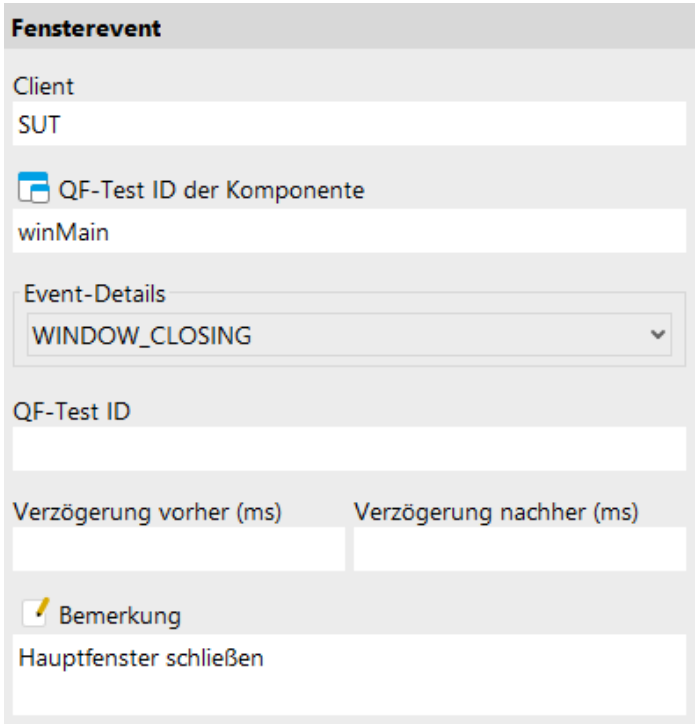


Abbildung 42.58: Fensterevent Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, an den der Event geschickt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Event bezieht.

Der "Komponente auswählen" Button öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw.

Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Event

In dieser *ComboBox* können Sie die Art des Events festlegen. Die möglichen Werte sind `WINDOW_OPENED`, `WINDOW_CLOSING`, `WINDOW_CLOSED`, `WINDOW_ACTIVATED`, `WINDOW_DEACTIVATED`, `WINDOW_ICONIFIED` und `WINDOW_DEICONIFIED`, entsprechend den Event IDs der Java-Klasse `WindowEvent`.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.8.5 Komponentenevent

Für QF-Test sind *ComponentEvents* eigentlich auch *WindowEvents*, da es alle *ComponentEvents* herausfiltert, die sich nicht auf Fenster beziehen. Übrig bleiben die Events `COMPONENT_MOVED` und `COMPONENT_SIZED`, die über den WindowManager an das SUT gelangen. Wird bei Web Tests statt einer Webseite[925] -Komponente eine HTML-Komponente angegeben, so wird das Browserfenster so angepasst, dass der Rendering-Bereich des Browsers in Größe bzw. Position mit den angegebenen Werten übereinstimmt.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Der Event wird zusammen mit den Daten des Zielfensters an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT das passende Fenster und löst den Event im SUT aus.

Attribute:

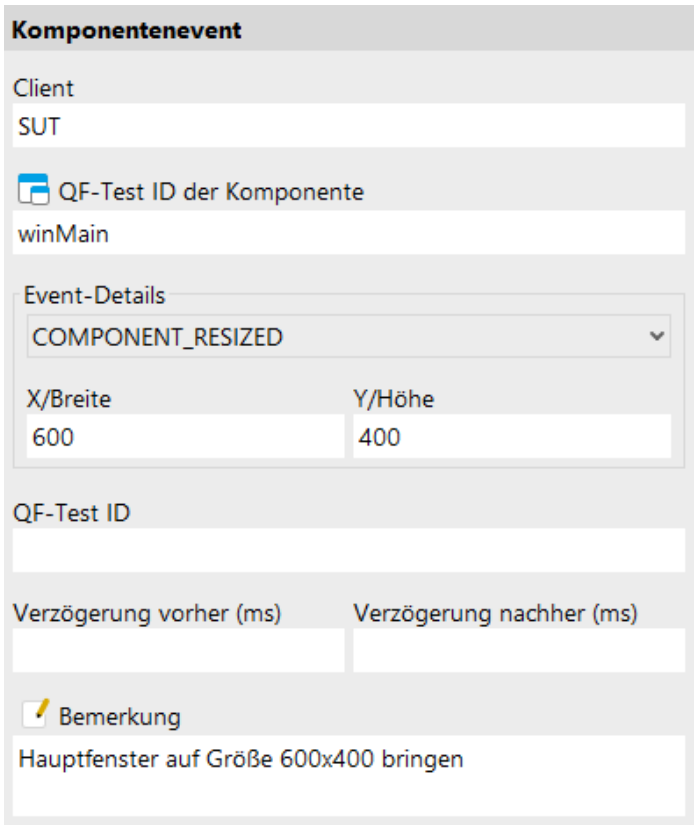


Abbildung 42.59: Komponentenevent Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, an den der Event geschickt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Event bezieht.

Der "Komponente auswählen" Button öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1(359)). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9(92)). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID(81) und Komponente-Knoten versus SmartID(51).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Event

In dieser *ComboBox* können Sie die Art des Events festlegen. Die möglichen Werte sind `COMPONENT_SIZED` und `COMPONENT_MOVED`, entsprechend den Event IDs der Java-Klasse `ComponentEvent`.

Variabel: Nein

Einschränkungen: Keine

X/Breite

Für einen `COMPONENT_MOVED` Event geben Sie hier die neue X-Koordinate des Fensters an, für einen `COMPONENT_SIZED` Event die neue Breite.

Variabel: Ja

Einschränkungen: Gültige Zahl, Breite > 0

Y/Höhe

Für einen `COMPONENT_MOVED` Event geben Sie hier die neue Y-Koordinate des Fensters an, für einen `COMPONENT_SIZED` Event die neue Höhe.

Variabel: Ja

Einschränkungen: Gültige Zahl, Höhe > 0

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.8.6 Auswahl

Ein Auswahl Knoten repräsentiert einen abstrakten Event, wie das Selektieren eines Menüpunktes, das Auswählen eines Eintrags in einer *ComboBox* oder eine Auswahl in oder das Schließen von einem System-Dialog. Aktuell wird dieser Event-Knoten nur für SWT, Web und Electron SUTs verwendet, wo einige Aktionen nicht über "weiche" Mausevents angestoßen werden können. Die Alternative des Verwendens von "harten" Mausevents hat einige Nachteile, wie sie beim Als "harten" Event wiedergeben[778] Attribut des Mausevent[775] Knotens beschrieben werden.

Das Detail[795] Attribut bestimmt den Typ der Aktion oder den zu selektierenden Wert, je nach Zielkomponente.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Der Event wird an das SUT geschickt, zusammen mit den Daten über die Zielkomponente. Die Komponente wird aufgelöst und eine Aktion durchgeführt, die vom Typ der Komponente abhängt, wie es die Tabelle oben zeigt.

Attribute:

Abbildung 42.60: Auswahl Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, an den der Event geschickt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Event bezieht.

Der "Komponente auswählen" Button öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1(359)). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener

Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9(92)). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID(81) und Komponente-Knoten versus SmartID(51).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Detail

Dieses Attribut bestimmt den Typ der Aktion oder den zu selektierenden Wert, je nach Zielkomponente. Die möglichen Kombinationen sind nachfolgend im Detail aufgeführt.

SWT

Die folgenden Kombinationen von SWT Komponenten (Widgets) und Detail Attribut werden aktuell für SWT unterstützt:

Klasse	Detail Attribut	Aktion
`CCombo` Unterelement	Leer	Unterelement als aktuellen Wert auswählen
`ColorDialog`	Farbwert in hexadezimalem #rrggbb Format	Angegebene Farbe auswählen
`ColorDialog`	`CANCEL`	Farbauswahl abbrechen
`Combo` Unterelement	Leer	Unterelement als aktuellen Wert auswählen
`CTabFolder` Unterelement	Leer	Reiter auswählen
`CTabFolder` Unterelement	`close`	Reiter schließen
`DirectoryDialog`	Verzeichnis	Angegebenes Verzeichnis auswählen
`DirectoryDialog`	`CANCEL`	Verzeichnisauswahl abbrechen
`FileDialog`	Dateiname, inklusive Verzeichnis	Angegebene Datei auswählen
`FileDialog`	`CANCEL`	Dateiauswahl abbrechen
`FontDialog`	Font-Beschreibung, systemspezifisch	Angegebenen Font auswählen
`FontDialog`	`CANCEL`	Fontauswahl abbrechen
`Menu`	`close`	Menü abbrechen (Schließen ohne Selektion)
`MenuItem`	Leer	Menüeintrag auswählen
`MessageBox`	Einer der Werte `OK YES NO CANCEL ABORT RETRY IGNORE OPEN SAVE`	Mit dem angegebenen Wert als Auswahl schließen
`ProgressBar`	Wert für `ProgressBar`	Angegebenen Wert setzen

`Scale`	Wert für `Scale`	Angegebenen Wert setzen
`Slider`	Wert für `Slider`	Angegebenen Wert setzen
`Spinner`	Wert für `Spinner`	Angegebenen Wert setzen
`TabFolder` Unterelement	Leer	Reiter auswählen

Tabelle 42.18: Unterstützte SWT Widgets für einen Auswahl Event

Hinweis

Eclipse/RCP verwendet viele dynamische `CTabFolder` bei denen die Unterelemente, also die Reiter, zwischen den Foldern verschoben werden können. Die Reiter repräsentieren die eigentlichen Business Objects, während die Folder nur ein Gerüst bilden um diese aufzunehmen. Zudem kann sich die Anordnung der Reiter und Folder bei einem Wechsel der Perspektive drastisch verändern. Daher ist es oft wünschenswert, einen Reiter direkt selektieren oder schließen zu können, unabhängig davon in welchem `CTabFolder` er sich gerade befindet. Dies kann mit Hilfe der Prozeduren(672) `qfs.qft#qfs.swt.ctabfolder.selectTab` und `qfs.qft#qfs.swt.ctabfolder.closeTab` erreicht werden, die mit der Standardbibliothek `qfs.qft` bereitgestellt werden. Neben dem allgegenwärtigen `client` Parameter muss nur der Parameter `tabname` mit dem Namen des gewünschten Reiters angegeben werden.

Web

Die folgenden Kombinationen von DOM-Knoten und Detail Attribut werden aktuell für Web SUTs unterstützt:

Knotentyp	**Detail Attribut**	**Aktion**
Bestätigungsdialog	Einer der Werte `OK YES NO CANCEL RETRY`	Dialog mit dem angegebenen Wert als Auswahl schließen
Dateiauswahldialog für Download	Die zu speichernde Datei oder `CANCEL`	Schließt den Dialog und speichert die gewählte Datei oder bricht den Download ab
Anmeldungsdialog	Name|Kennwort oder `CANCEL`	Schließt den Dialog zur Anmeldung mit den angegebenen Daten oder bricht die Anmeldung ab. Das Kennwort kann durch Rechts-Klick und Auswahl von Text verschlüsseln im resultierenden Popupmenü verschlüsselt werden. Geben Sie auf jeden Fall vor der Verschlüsselung einen Salt in der Option Salt für Verschlüsselung von Kennwörtern(532) an.
Eingabedialog	Der einzugebende Text oder `CANCEL`	Schließt den Dialog liefert den eingegebenen Text oder bricht die Eingabe ab

DOCUMENT auf oberster Ebene	`back`	Navigiert zurück zur vorhergehenden Seite
DOCUMENT auf oberster Ebene	`forward`	Navigiert vorwärts zur nächsten Seite
DOCUMENT auf oberster Ebene	`goto`:URL	Navigiert zur angegebenen URL
DOCUMENT auf oberster Ebene	`refresh`	Lädt die aktuelle Seite neu
DOCUMENT auf oberster Ebene	`stop`	Beendet das Laden der aktuelle Seite
OPTION oder Unterelement von SELECT	`0`	Wählt die OPTION als aktuellen Wert
OPTION oder Unterelement von SELECT	`1`	Fügt die OPTION zur Selektion hinzu
OPTION oder Unterelement von SELECT	`-1`	Entfernt die OPTION aus der Selektion

Tabelle 42.19: Unterstützte DOM-Knoten für einen Auswahl Event

Electron

Die folgenden Kombinationen von DOM-Knoten und Detail Attribut werden aktuell für Electron-Anwendungen (siehe Kapitel 21[310]) unterstützt, zusätzlich zu den oben für Web genannten:

Knotentyp	Detail Attribut	Aktion
Menü	`clickmenu:@/<Menüpfad>`	Wobei `<Menüpfad>` das Menü und den oder die Untermenüpunkte, getrennt durch `/` bezeichnet. Wenn Sie zum Beispiel im Menü `Datei` den Unterpunkt `Speichern unter` aktivieren wollen, lautet der Eintrag `clickmenu:@Datei/Speichern unter`. Im Attribut QF-Test ID der Komponente ist die QF-Test ID des Knotens `Webseite` des SUT einzutragen.
Dialog	`select:@/<Rückgabewert>`	Salt für Verschlüsselung von Kennwörtern[532] Schließt einen geöffneten Dialog. Weitere Informationen zu Rückgabewerten finden Sie in Native Dialoge[311]. Im Attribut QF-Test ID der Komponente muss die QF-Test ID des Knotens `Web page` des SUTs angegeben werden.
Error-Dialog	select:1	Schließt den Error-Dialog.

Message-Box	select:2:true	Wählt den Button mit der Id 2 aus und setzt den Wert der Checkbox auf ”true”.
Speichern-Dialog	select:”C:\path\to\my.file”	Schließt den Speichern-Dialog und gibt den angegebenen Pfad zurück.
Öffnen-Dialog	select:[”C:\path\to\my.file”]	Schließt den Öffnen-Dialog und gibt den angegebenen Pfad zurück.
Öffnen-Dialog	select:[”C:\path\to\my\first.file”, ”C:\path\to\my\second.file”]	Schließt den Öffnen-Dialog und gibt die angegebenen Pfade zurück.

Tabelle 42.20: Unterstützte DOM-Knoten bei Electron SUTs für einen Auswahl Event

Android
iOS

Die folgenden Werte für das Detail Attribut werden aktuell für Android-Anwendungen (siehe Kapitel 16[245]) und iOS-Anwendungen (siehe Kapitel 17[269]) unterstützt. Wenn Sie den Auswahl Knoten lieber über eine Prozedur ausführen, ohne sich um die Syntax im Detail Attribut kümmern zu müssen, können Sie die Prozeduren in den Packages `qfs.android.device`, beziehungsweise `qfs.ios.device`, in der Standardbibliothek[183] `qfs.qft` nutzen.

Knotentyp	**Detail Attribut**	**Aktion**
Alle (wird ignoriert)	`HOME`	Klick auf den Home-Button des Emulators.
Nur Android, alle (wird ignoriert)	`BACK`	Klick auf den Back-Button des Emulators.
Nur Android, alle (wird ignoriert)	`APP_SWITCH`	Klick auf den App-Switch-Button des Emulators.
Alle (wird ignoriert)	`rotate: <Winkel>`	`rotate` dreht die gesamte Anzeige um den angegebenen Winkel. Gültige Werte: 0, 90, 180, 270.
Alle (wird ignoriert)	`turn: <Richtung>`	`turn` dreht die gesamte Anzeige 90 Grad in der angegebenen Richtung. Gültige Werte: `left, right`.

Alle	`swipe: <Richtung>`	Swipe auf einer Komponente in eine bestimmte Richtung. Mögliche Angaben für die Wischbewegung sind (jeweils ohne Anführungszeichen): • vom linken zum rechten Rand der Komponente: ”right”, ”→”, ”go_left”, ”prevPage”, ””, • vom rechten zum linken Rand der Komponente: ”left”, ”←”, ”go_right”, ”nextPage”, ””, • vom unteren zum oberen Rand der Komponente: ”up”, ”↑”, ”go_down”, ”scrollDown”, ””, • vom oberen zum unteren Rand der Komponente: ”down”, ”↓”, ”go_up”, ”scrollUp”, ””, • von der oberen linken zur unteren rechten Ecke der Komponente: ”down_right”, ”↘”, ”go_up_left”, ””, • von der unteren rechten zur oberen linken Ecke der Komponente: ”up_left”, ”↖”, ”go_down_right”, ””, • von der oberen rechten zur unteren linken Ecke der Komponente: ”down_left”, ””, ”go_up_right”, ””, • von der unteren linken zur oberen rechten Ecke der Komponente: ”up_right”, ””, ”go_down_left” and ””.

Alle	`swipe: <Startkoordinate X> <Startkoordinate Y> <Endkoordinate X> <Endkoordinate Y> [<Zeit in ms> [<Teilschritte>]]`	Swipe auf einer Komponente, wobei der Swipe-Anfang und das Swipe-Ende durch die angegebenen Koordinaten spezifiziert wird. Optional kann die Zeit angegeben werden, die die Aktion dauern soll. Wenn die Zeit angegeben wird, kann optional zusätzlich die Anzahl Teilschritte angegeben werden, in die die Wischaktion aufgegliedert wird. Die Teilschritte müssen nur in Spezialfällen angegeben werden. Die Koordinaten können als Anzahl der Pixel (bezogen auf die obere linke Ecke der Komponente) oder als Position angegeben werden, siehe Tabelle Positionsangaben für Gesten(801). Beispiele: `swipe: W C E C` wischt also mittig (C = Center) von links (W = Westen) der Komponente nach rechts (E = Osten) mittig der Komponente, `swipe: C N C S` von oben nach unten und `swipe: 0 0 E S` von links oben nach rechts unten.
Alle	`zoom: <Startkoordinate X> <Startkoordinate Y> <Entfernung> <Winkel>`	`zoom` ist eine Zweifingeraktion. Die Startposition der beiden Finger wird über die Startkoordinaten festgelegt, die Bewegung über die Entfernung (in Pixeln) und den Winkel (0 bis 359). Der Winkel 0 stellt eine horizontale, 90 eine senkrechte Bewegung dar. Hierbei entfernen sich die Finger jeweils um die angegebene Entfernung in entgegengesetzter Richtung vom Startpunkt. Die Koordinaten können als Anzahl der Pixel (bezogen auf die obere linke Ecke der Komponente) oder als Position angegeben werden, siehe Tabelle Positionsangaben für Gesten(801). Beispiele: `zoom: C C 50 0` zoom, ausgehend von der Mitte bewegen sich die ”Finger” jeweils 50 Pixel horizontal.

Alle	`pinch: <Finger 1 X> <Finger 1 Y> <Finger 2 X> <Finger 2 Y>`	`pinch` ist eine Zweifingeraktion. Die Koordinaten geben die Positionen der beiden Finger an. Bei der Bewegung bewegen sich die Finger gleichmäßig auf einander zu bis sie sich treffen. Die Koordinaten können als Anzahl der Pixel (bezogen auf die obere linke Ecke der Komponente) oder als Position angegeben werden, siehe Tabelle Positionsangaben für Gesten[801]. Beispiele: `pinch: 20 C 50 C` horizontal in der Mitte der Komponenten bewegen sich die ”Finger” von den Pixelpositionen 20 und 50 zueinander.

Tabelle 42.21: Unterstützte Werte für ein Auswahl Knoten für Android und iOS

Für Gesten werden Start- und Endpunkte spezifiziert. Jeder Punkt hat eine X- und eine Y-Komponente. Diese kann entweder als Anzahl der Pixel (bezogen auf die obere linke Ecke der Komponente) oder als Position angegeben werden. Folgende Positionsangaben sind möglich:

Positionsangabe	Erläuterung
`N`	Oberer Rand der Komponente (Norden).
`E`	Rechter Rand der Komponente (East = Osten).
`S`	Unterer Rand der Komponente (Süden).
`W`	Linker Rand der Komponente (Westen).
`C`	Mitte der Komponente (Center) für die jeweilige Dimension.

Tabelle 42.22: Positionsangaben für Gesten

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.8.7 Dateiauswahl

Eine Dateiauswahl nimmt unter den Ereignissen eine Sonderstellung ein, da es sich nicht um ein einfaches, sondern ein komplexes Pseudo-Ereignis handelt.

Der Hintergrund ist, dass Java bei der Dateiauswahl mit Hilfe eines `java.awt.FileDialog` die Kontrolle über den Dialog komplett an das Betriebssystem abgibt. Damit lassen sich während der Dateiauswahl keine Maus- und Tastaturevents aufzeichnen und ebenso wenig simulieren. Dieses Problem tritt nicht bei Verwendung eines eigenen Dateiauswahldialogs, oder des Swing `javax.swing.JFileChooser` auf.

Daher nimmt QF-Test nur das Ergebnis der Dateiauswahl auf und legt es in Form dieses Knotens ab. Bei der Wiedergabe werden die Daten entsprechend an den Dialog übergeben, bevor er geschlossen wird. Für das SUT ist das im Endeffekt nicht von einer Dateiauswahl durch den Anwender zu unterscheiden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Datei und Verzeichnis werden an einen geöffneten `java.awt.FileDialog` übergeben und dieser wird geschlossen. Ist kein FileDialog geöffnet, wird eine `ComponentNotFoundException`(958) geworfen.

Attribute:

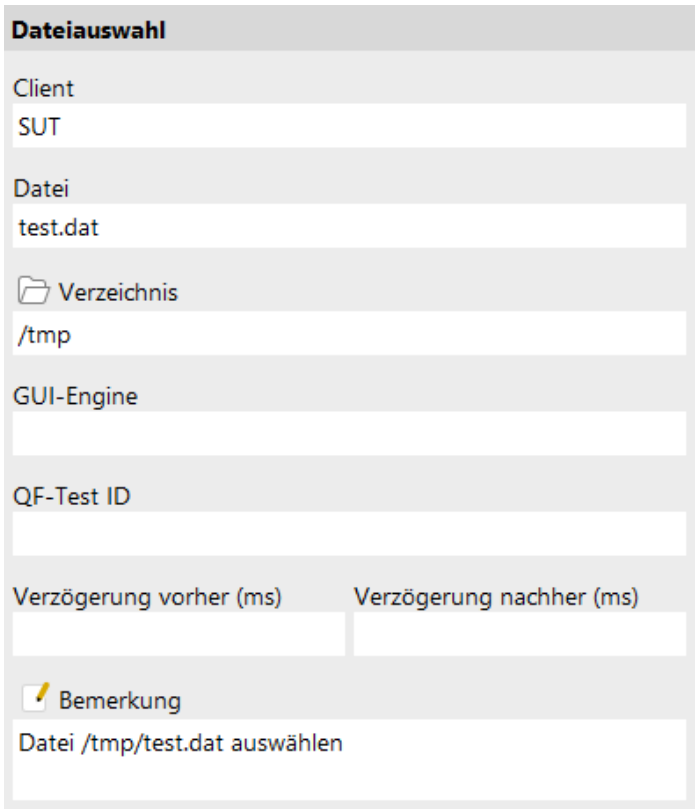


Abbildung 42.61: Dateiauswahl Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, an den der Event geschickt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Datei

Der Name der Datei (ohne Verzeichnispfad), die im Dialog ausgewählt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Verzeichnis

Das Verzeichnis der Datei, die im Dialog ausgewählt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

GUI-Engine

Die GUI-Engine zu der der Dateiauswahl Dialog gehört. Nur relevant für SUTs mit mehr als einer GUI-Engine wie in Kapitel 45(998) beschrieben.

Variabel: Ja

Einschränkungen: Siehe Kapitel 45(998)

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.9 Checks

Checks vergleichen die Anzeige oder den Status von Komponenten im SUT mit vorgegebenen Werten. Stimmen die Daten nicht überein, wird ein Fehler protokolliert. Zusätzlich oder alternativ kann ein fehlschlagender Check eine `CheckFailedException`(963) auslösen, und das Ergebnis des Checks einer Variablen zugewiesen werden.

Ein Check kann mit Hilfe seiner Wartezeit(809) auch dazu eingesetzt werden, im SUT darauf zu warten, dass eine Komponente einen bestimmten Zustand annimmt, also z.B. ein *MenuItem* aktiviert oder eine *CheckBox* selektiert wird.

Wie bei den Events(775) muss für jeden Check ein Fenster, eine Komponente oder ein Unterelement als Ziel angegeben werden. Je nach Ziel werden unterschiedliche Checks unterstützt, die wie in Abschnitt 4.3(43) beschrieben im Checkmodus durch einen Rechts-Klick auf die Zielkomponente und Auswahl im zugehörigen Menü aufgenommen werden können. Werden für spezielle Komponenten Checks benötigt, die von QF-Test standardmäßig nicht angeboten werden, lassen sich diese über das `Checker` Erweiterungs-API selbst implementieren. Näheres hierzu finden Sie in Abschnitt 54.4(1208).

Für die verschiedenen Arten von Checks stehen sechs verschiedene Datentypen zu Verfügung und jedem dieser Datentypen entspricht ein spezieller Check-Knoten. Da für eine Komponente mehrere Checks vom selben Datentyp möglich sind, z.B. enabled-Status und editable-Status für ein Textfeld, beide vom Typ Boolean, werden Checks zusätzlich anhand des Attributs Name des Check-Typs unterschieden. In den meisten Fällen, wenn der Datentyp und die Zielkomponente ausreichend sind, um einen Standard-Check zu identifizieren, hat dieses Attribut den Wert *default*. Wird der angegebene Check-Typ für die Zielkomponente nicht unterstützt führt dies zu einer `CheckNotSupportedException`(963).

QF-Test stellt die folgenden Check-Knoten zur Verfügung: Checks können unabhängig vom Ergebnis im HTML-Report angezeigt werden. Dies kann bei der interaktiven Generierung das Reports über die Option `Check auflisten` oder im Batch-Modus über das Kommondozeilenargument `-report-checks`(986) aktiviert werden. Bitte beachten Sie, dass dies nur für Checks mit Standard-Ergebnisbehandlung zutrifft, also nur Dokumentation im Protokoll, kein Setzen einer Ergebnisvariablen oder Werfen einer Exception. Weitere Informationen finden Sie unter Abschnitt 24.1.2(333).

QF-Test stellt die folgenden Check-Knoten zur Verfügung:

- Check Text(806)
- Check Boolean(812)
- Check Elemente(818)
- Check selektierbare Elemente(823)
- Check Abbild(828)
- Check Geometrie(834)

42.9.1 Check Text

Vergleicht einen vorgegebenen Text mit der Anzeige einer Komponente oder eines Unterelements.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Der Vergleichstext wird zusammen mit den Daten der Zielkomponente an das SUT geschickt. Dort wird die dazu passende Komponente ermittelt, deren aktueller Text ausgelesen und mit den Vorgaben verglichen.

Attribute:

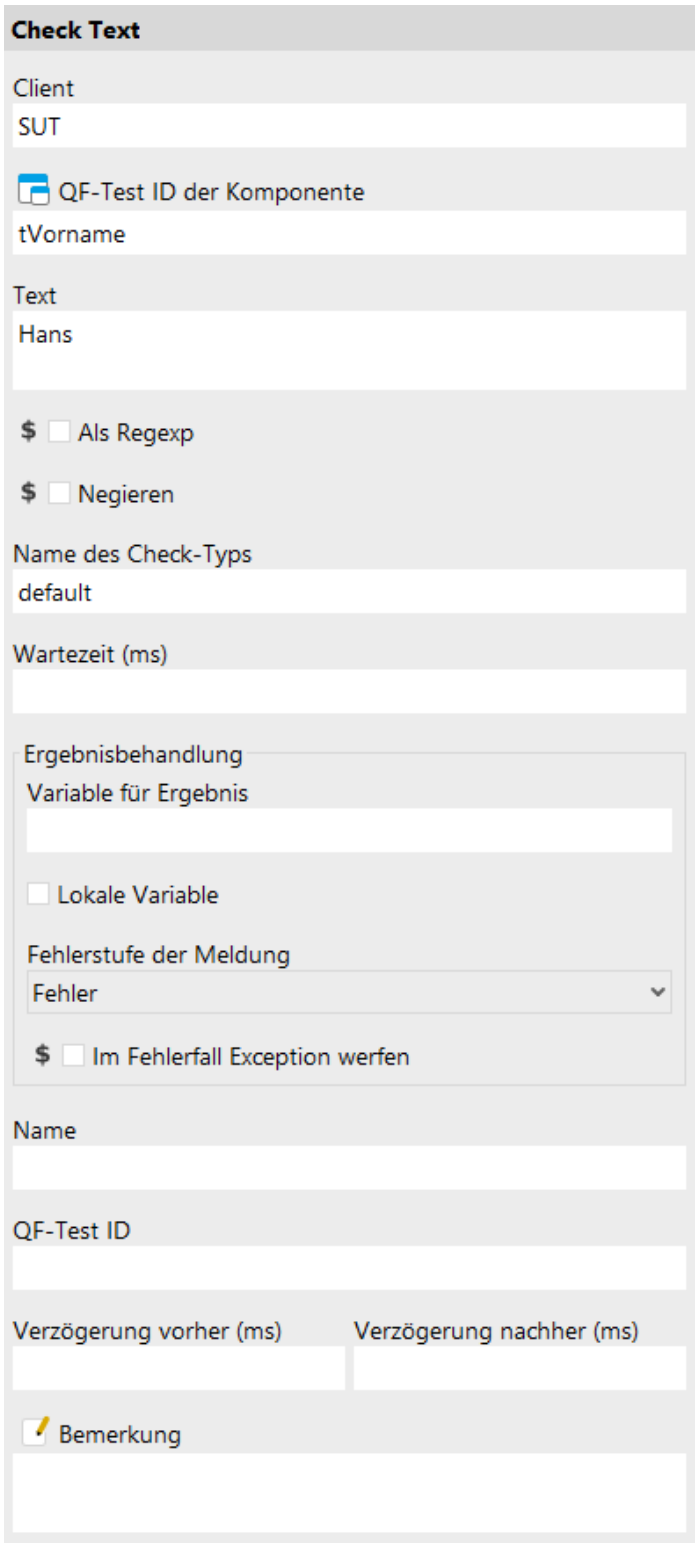


Abbildung 42.62: Check Text-Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, in dem der Check vorgenommen werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster[919], Komponente[930] oder Element[936] Knotens, auf den sich der Check bezieht.

Der "Komponente auswählen" Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5[47]) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Text

Die Vorgabe, mit der die Anzeige der Komponente verglichen wird.

4.0+ Mittels Rechtsklick und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke mit '\' schützen.

Variabel: Ja

Einschränkungen: Gültige Regexp, falls Als Regexp gesetzt ist.

Als Regexp

Ist dieses Attribut gesetzt, findet der Vergleich mittels eines regulären Ausdrucks statt (vgl. Abschnitt 49.3[1023]), andernfalls als 1:1 Textvergleich.

Variabel: Ja

Einschränkungen: Keine

Negieren

Ist dieses Attribut gesetzt, wird das Ergebnis des Checks negiert, d.h. die geprüfte Eigenschaft darf in diesem Fall nicht mit dem erwarteten Wert übereinstimmen.

Variabel: Ja

Einschränkungen: Keine

Name des Check-Typs

Hiermit wird konkretisiert, welcher Check ausgeführt werden soll. Dadurch wird es möglich, für eine Komponente verschiedene Checks mit dem selben Datentyp anzubieten, ohne dass es dadurch zu Konflikten kommt. Mit Hilfe eines `Checkers` können zusätzliche Check-Typen implementiert werden (vgl. Abschnitt 54.4(1208)). Standardmäßig erlaubt der Check Text Knoten folgende Check-Typen (bei unterstützenden Komponenten):

Check	Beschreibung	Engines
default	Der Text der Komponente	Alle
tooltip	Der Tooltip der Komponente. Hierfür müssen Sie ggf. vorher mit einem Mausevent den Cursor über die Komponente bewegen, damit der Tooltip auch wirklich initialisiert wird.	Alle
text_positioned	Details siehe PDF Check Text(290).	Nur PDF
text_font	Der Textfont der Komponente. Details siehe PDF 'Check Font'(295).	Nur PDF
text_fontsize	Die Textfontgröße der Komponente. Details siehe PDF 'Check Font-Größe'(295).	Nur PDF
class	Die CSS Klasse(n) der Komponente	Nur Web
id	Das Attribut 'id' der Komponente	Nur Web
name	Das Attribut 'name' der Komponente	Nur Web
value	Das Attribut 'value' der Komponente	Nur Web
href	Das Attribut 'href' der Komponente	Nur Web
attribute:NAME	Das Attribut namens NAME der Komponente	Nur Web

Tabelle 42.23: Standardmäßig implementierte Check-Typen des Check Text

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf, bis die Komponente den geforderten Zustand angenommen haben muss. Lassen Sie das Attribut leer, oder geben Sie 0 an, wenn Sie keine Verzögerung zulassen wollen.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur(672), Abhängigkeit(630) oder Testfall(599) Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6(116).

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`(963) geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

Name

Ein optionaler Name für den Check, der für mehr Klarheit im Report hilfreich sein kann.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.9.2 Check Boolean

Vergleicht einen erwarteten Status einer Komponente oder eines Unterelements mit dem aktuellen Zustand.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Der erwartete Status wird zusammen mit den Daten der Zielkomponente an das SUT geschickt. Dort wird die dazu passende Komponente ermittelt und deren aktueller Status mit den Vorgaben verglichen.

Attribute:

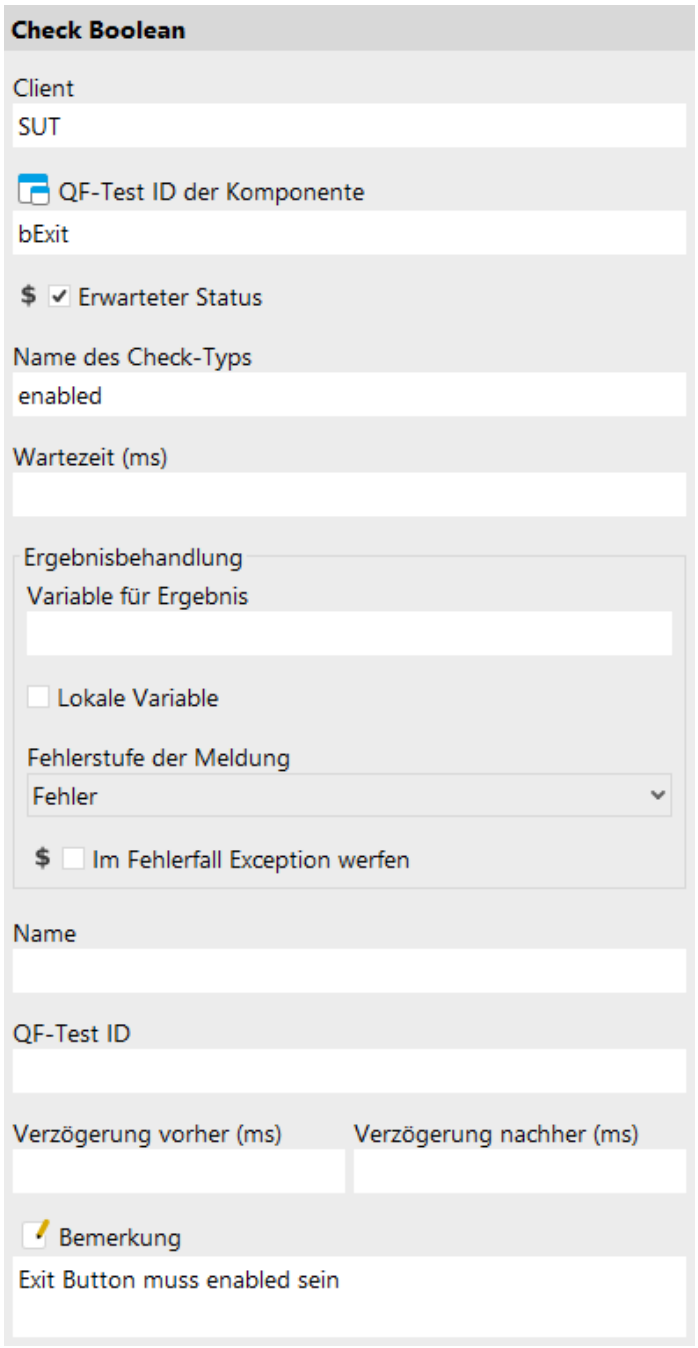


Abbildung 42.63: Check Boolean-Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, in dem der Check vorgenommen werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster[919], Komponente[930] oder Element[936] Knotens, auf den sich der Check bezieht.

Der "Komponente auswählen" Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5[47]) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Erwarteter Status

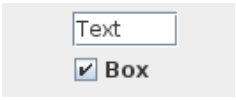
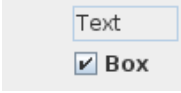
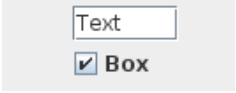
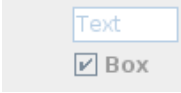
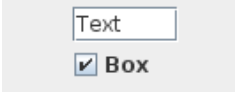
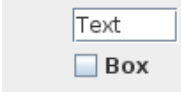
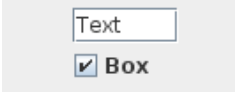
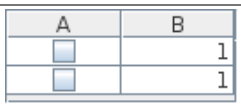
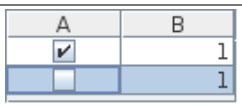
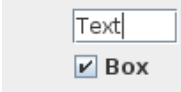
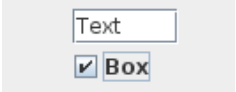
Die Vorgabe, mit der der Status der Komponente verglichen wird.

Variabel: Ja

Einschränkungen: Keine

Name des Check-Typs

Hiermit wird konkretisiert, welcher Check ausgeführt werden soll. Dadurch wird es möglich, für eine Komponente verschiedene Checks mit dem selben Datentyp anzubieten, ohne dass es dadurch zu Konflikten kommt. Insbesondere bei Check Boolean Knoten können oft mehrere verschiedene Zustände geprüft werden, z.B. 'enabled', 'editable' oder 'selected'. In der nachfolgenden Tabelle werden einige Check-Typen erläutert. Welche Check-Typen konkret für eine Komponente zur Verfügung stehen, hängt von der Komponentenklasse ab. Mit Hilfe eines `Checkers` können zusätzliche Check-Typen implementiert werden (vgl. Abschnitt 54.4[1208]).

Check-Typ	Beispiel 1	Beispiel 2	Beschreibung
visible			In Beispiel 1 ist das Textfeld und die CheckBox unsichtbar. In Beispiel 2 sind beide sichtbar.
editable			Beispiel 1 zeigt ein nicht editierbares Textfeld Im Gegensatz dazu ist es bei Beispiel 2 editierbar. Die CheckBox unterstützt diesen Check nicht.
enabled			In Beispiel 1 kann man weder mit dem Textfeld noch mit der CheckBox interagieren. In Beispiel 2 kann man mit beiden interagieren.
checked (früher selected)			Beispiel 1 zeigt eine nicht selektierte CheckBox. In Beispiel 2 ist sie selektiert. Das Textfeld unterstützt diesen Check nicht.
selected (Tabelle)			Beispiel 1 zeigt eine Tabelle, in der keine Zelle selektiert ist. In Beispiel 2 sind die unteren Zellen selektiert, wie am Rahmen ersichtlich. Der Check betrifft die Selektion der Zeile, nicht der CheckBox.
focused			In Beispiel 1 ist das Textfeld fokussiert (erkennbar am Cursor). In Beispiel 2 dagegen ist die CheckBox fokussiert (erkennbar am Rahmen).

attribute:NAME	Check ”attribute:sel” ergibt ”True”:
	`<p sel></p>`
	`<p sel=””></p>`
	`<p sel=”text”></p>`
	Check ”attribute:sel” ergibt ”False”:
	`<p sel=”0”></p>`
	`<p sel=”False”></p>`
	`<p></p>`
			Nur Web: Wenn das Attribut namens NAME der Komponente existiert, wird dieser Check ”True” zurückgeben, es sei denn der Attributwert ist ”0” oder ”false” (unabhängig von Groß-/Kleinschreibung).
			Wenn das Attribut nicht existiert ergibt der Check ”False”.

Tabelle 42.24: Standardmäßig implementierte Check-Typen des Check Boolean

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf, bis die Komponente den geforderten Zustand angenommen haben muss. Lassen Sie das Attribut leer, oder geben Sie 0 an, wenn Sie keine Verzögerung zulassen wollen.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur(672), Abhängigkeit(630) oder Testfall(599) Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6(116).

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`(963) geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

Name

Ein optionaler Name für den Check, der für mehr Klarheit im Report hilfreich sein kann.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.9.3 Check Elemente

Vergleicht eine Liste von vorgegebenen Texten mit der Anzeige einer Komponente oder eines Unterelements.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Vergleichstexte werden zusammen mit den Daten der Zielkomponente an das SUT geschickt. Dort wird die dazu passende Komponente ermittelt, deren Elemente ausgelesen und mit den Vorgaben verglichen.

Attribute:

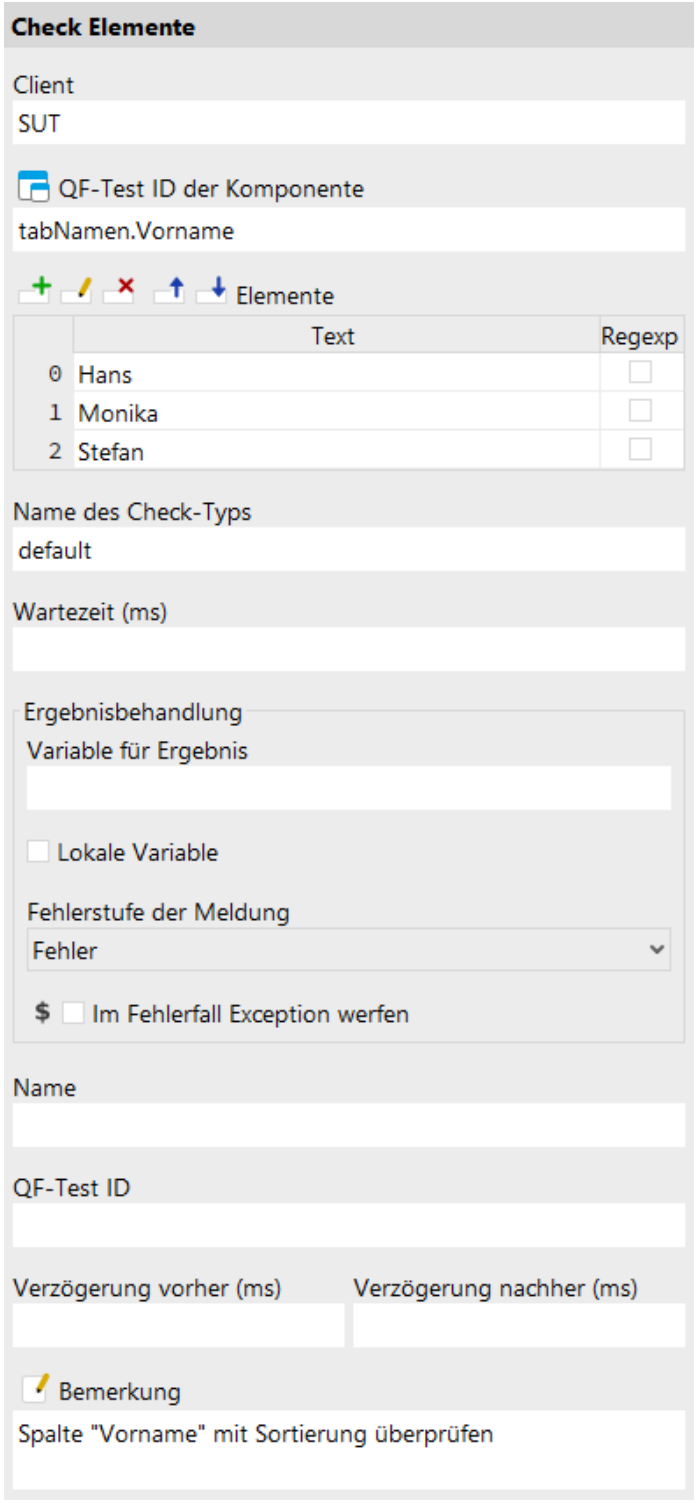


Abbildung 42.64: Check Elemente-Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, in dem der Check vorgenommen werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Check bezieht.

Der "Komponente auswählen" Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1(359)). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9(92)). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID(81) und Komponente-Knoten versus SmartID(51).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Elemente

Diese Tabelle enthält die Vorgaben, mit denen die Anzeige der Komponente verglichen wird. Jede Zeile entspricht einem Unterelement der Zielkomponente. In der "Text" Spalte wird der Vergleichswert angegeben, in der "Regexp" Spalte wird festgelegt, ob es sich dabei um einen regulären Ausdruck handelt (vgl. Abschnitt 49.3(1023)). Die "Regexp" Spalte erlaubt übrigens auch die Verwendung von Variablen. Hierfür klicken Sie doppelt auf die Spalte und tragen den entsprechenden Wert ein.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5(20).

4.0+ Mittels Rechtsklick in eine Zelle und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke im Zellentext mit '\' schützen.

Variabel: In der "Text" Spalte ja, sonst nein.

Einschränkungen: Gültige Regexp, falls Als Regexp gesetzt ist.

Name des Check-Typs

Hiermit wird konkretisiert, welcher Check ausgeführt werden soll. Dadurch wird es möglich, für eine Komponente verschiedene Checks mit dem selben Datentyp anzubieten, ohne dass es dadurch zu Konflikten kommt. Mit Hilfe eines `Checkers` können zusätzliche Check-Typen implementiert werden (vgl. Abschnitt 54.4[1208]).

Die für generische Klassen implementierten Check-Typen sind im Kapitel 61[1329] jeweils im Absatz "Zusätzliche Checks" beschrieben, zum Beispiel für Accordion[1330], List[1339], Table[1349], TabPanel[1352], TextArea[1353], Tree[1357] und TreeTable[1358].

Bei den generischen Klassen Table[1349] und TreeTable[1358] ist es für die beiden Check-Typen `column` und `row` möglich, nur einen Teil der Einträge zu prüfen. Hierfür stehen die Parameter `start` und `count` zur Verfügung. Mit dem Ausdruck `row;start=2;count=3` werden zum Beispiel nur die Einträge der Zeilen drei bis fünf geprüft, `column;start=0;count=4` prüft die Einträge in den ersten vier Spalten.

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf, bis die Komponente den geforderten Zustand angenommen haben muss. Lassen Sie das Attribut leer, oder geben Sie 0 an, wenn Sie keine Verzögerung zulassen wollen.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition

der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur[672], Abhängigkeit[630] oder Testfall[599] Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6[116].

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren[593] kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`[963] geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

Name

Ein optionaler Name für den Check, der für mehr Klarheit im Report hilfreich sein kann.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.9.4 Check selektierbare Elemente

Vergleicht zusätzlich zu einer Liste von vorgegebenen Texten auch die Selektion der Unterelemente einer Komponente.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Die Vergleichstexte und die Selektionsinformation werden zusammen mit den Daten der Zielkomponente an das SUT geschickt. Dort wird die dazu passende Komponente ermittelt, deren Elemente ausgelesen und mit den Vorgaben verglichen.

Attribute:

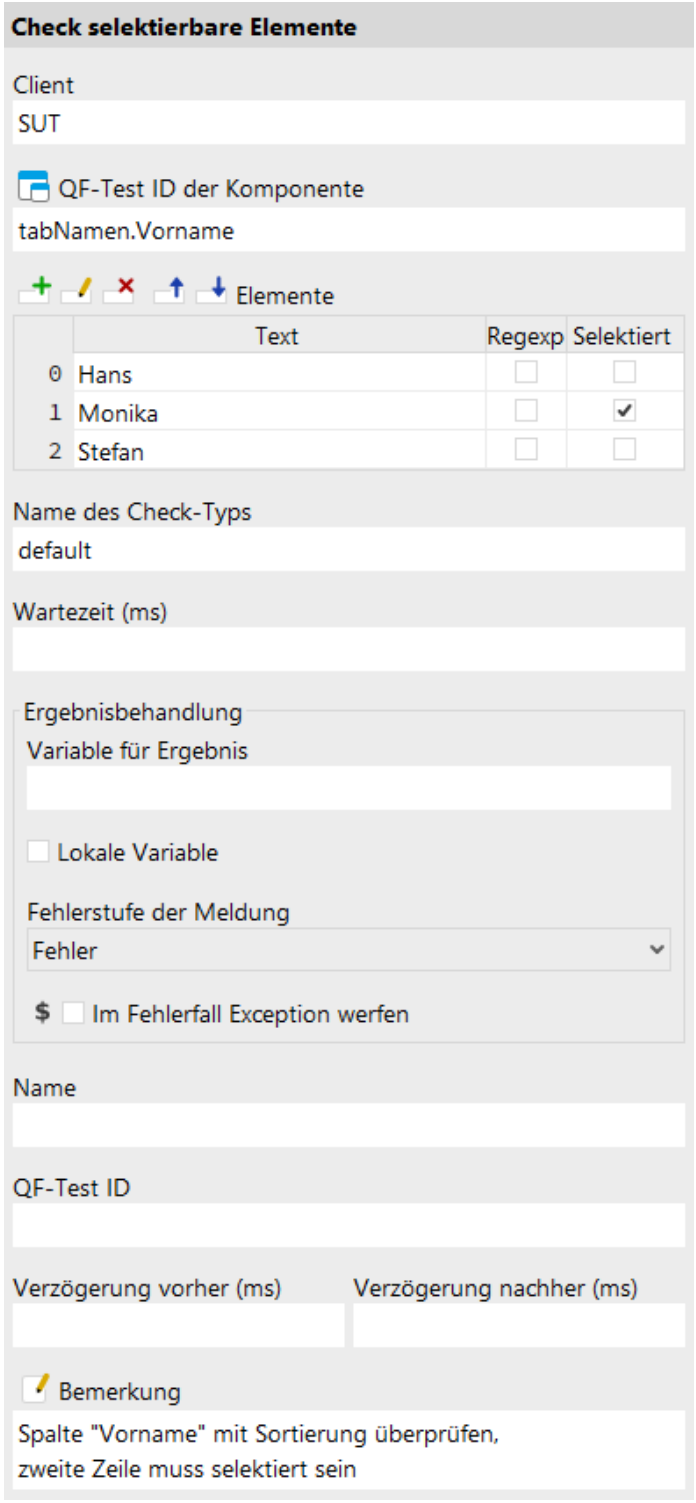


Abbildung 42.65: Check selektierbare Elemente-Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, in dem der Check vorgenommen werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Check bezieht.

Der "Komponente auswählen" Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1(359)). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9(92)). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID(81) und Komponente-Knoten versus SmartID(51).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Elemente

Diese Tabelle enthält die Vorgaben, mit denen die Anzeige der Komponente verglichen wird. Jede Zeile entspricht einem Unterelement der Zielkomponente. In der "Text" Spalte wird der Vergleichswert angegeben, in der "Regexp" Spalte wird festgelegt, ob es sich dabei um einen regulären Ausdruck handelt (vgl. Abschnitt 49.3(1023)). In der "Selektiert" Spalte legen Sie fest, ob das Element im SUT selektiert sein muss. Die "Regexp" sowie die "Selektiert" Spalte erlauben übrigens auch die Verwendung von Variablen. Hierfür klicken Sie doppelt auf die benötigte Spalte und tragen den entsprechenden Wert ein.

Näheres zur Arbeit mit den Tabellen finden Sie in Abschnitt 2.2.5(20).

4.0+ Mittels Rechtsklick in die Zelle und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke im Zellentext mit '\' schützen.

Variabel: In der "Text" Spalte ja, sonst nein.

Einschränkungen: Gültige Regexp, falls Als Regexp gesetzt ist.

Name des Check-Typs

Hiermit wird konkretisiert, welcher Check ausgeführt werden soll. Dadurch wird es möglich, für eine Komponente verschiedene Checks mit dem selben Datentyp anzubieten, ohne dass es dadurch zu Konflikten kommt. Mit Hilfe eines `Checkers` können zusätzliche Check-Typen implementiert werden (vgl. Abschnitt 54.4[1208]).

Die für generische Klassen implementierten Check-Typen sind im Kapitel 61[1329] jeweils im Absatz "Zusätzliche Checks" beschrieben, zum Beispiel für Accordion[1330], List[1339], Table[1349], Tree[1357] und TreeTable[1358].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf, bis die Komponente den geforderten Zustand angenommen haben muss. Lassen Sie das Attribut leer, oder geben Sie 0 an, wenn Sie keine Verzögerung zulassen wollen.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur[672], Abhängigkeit[630] oder Testfall[599] Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem

Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6[116].

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren[593] kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`[963] geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

Name

Ein optionaler Name für den Check, der für mehr Klarheit im Report hilfreich sein kann.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung

bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.9.5 Check Abbild

Vergleicht ein Abbild einer Komponente mit dem aktuellen Zustand. Der Check funktioniert mit allen Komponenten und mit Unterelementen.

Es können auch Ausschnitte eines Bildes geprüft werden. Zu diesem Zweck kann ein rechteckiger Ausschnitt durch Ziehen mit der Maus oder direkte Angabe der Koordinaten festgelegt werden. Ist das Bild kleiner als die Komponente, kann außerdem die Position des Abbilds relativ zum Ursprung der Komponente festgelegt werden. Beim Aufnehmen des sichtbaren Bereichs einer Komponente oder beim Zuschneiden des Bildes auf den festgelegten Ausschnitt, wird dieser Versatz automatisch bestimmt.

Neben allen Arten von Komponenten können auch Unterelement geprüft werden.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Das Abbild wird zusammen mit den Daten der Zielkomponente an das SUT geschickt. Dort wird die dazu passende Komponente ermittelt und deren Abbild mit den Vorgaben verglichen.

Attribute:

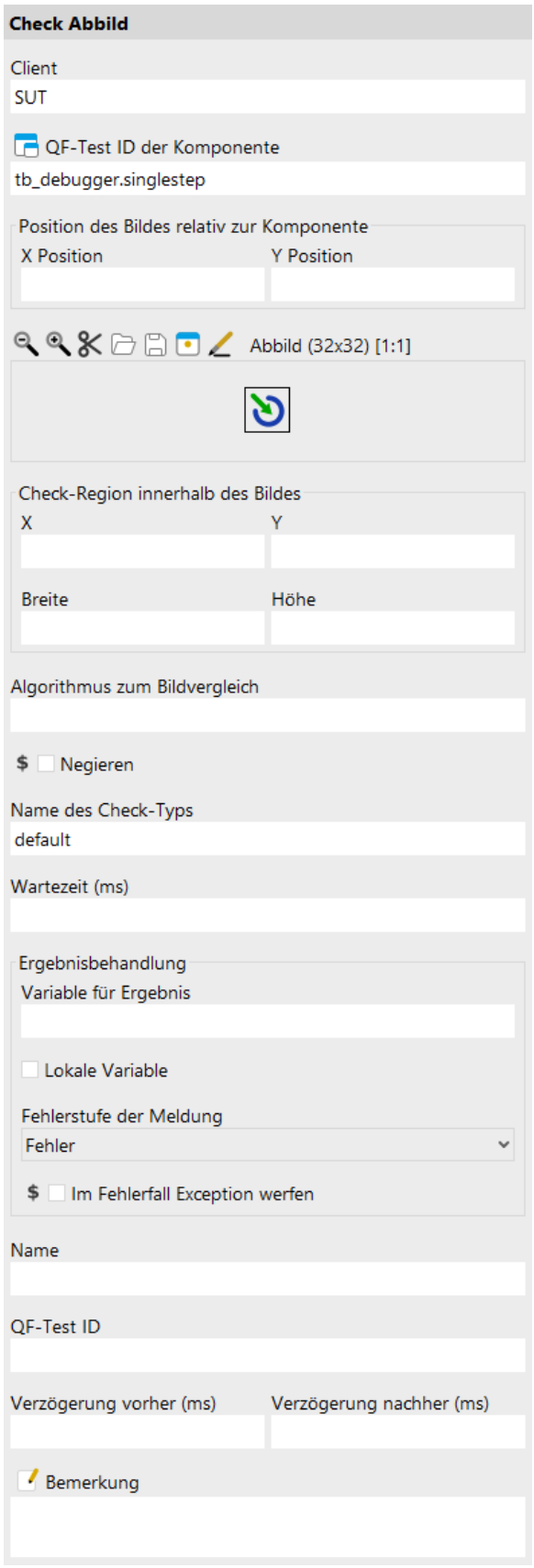


Abbildung 42.66: Check Abbild-Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, in dem der Check vorgenommen werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Check bezieht.

Der "Komponente auswählen" Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1(359)). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9(92)). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID(81) und Komponente-Knoten versus SmartID(51).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

'Position des Bildes relativ zur Komponente'

Ist das Bild kleiner als die zu prüfende Komponente, legen diese Koordinaten die Position des Ausschnitts innerhalb der Komponente fest.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

Abbild

Das Abbild der Komponente. Die Darstellung kann in unterschiedlichen Zoom-Stufen erfolgen, das Bild kann in einer PNG Datei gespeichert oder aus einer solchen geladen werden. Außerdem kann ein externes Grafikprogramm zur Bearbeitung des Bildes gestartet werden. Dieses muss zunächst über die Option Kommando für externes Grafikprogramm(499) festgelegt werden.

Der Text neben den Icons gibt die Größe und die aktuelle Zoomstufe des Bildes wieder. Weiterhin wird hier auch noch der Farbwert des Pixels über dem sich die

Maus befindet angezeigt, sofern sich der Mauscursor gerade über dem Bild befinden sollte. Der Farbwert kann hier entweder im Hexadezimalformat dargestellt sein oder im rgba Format, wobei man durch einen Klick auf diesen Text zwischen den beiden Darstellungen hin und her wechseln kann.

Variabel: Nein

Einschränkungen: Keine

'Check-Region innerhalb des Bildes'

Soll nur ein Ausschnitt des Abbildes überprüft werden, kann mit diesen Werten ein rechteckiger Bereich festgelegt werden, der dann im Abbild der Komponente gesucht wird.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

'Algorithmus zum Bildvergleich'

Mit diesem Attribut kann ein spezieller Algorithmus zum Bildvergleich definiert werden. Ist der Wert leer, kann über die Option Standardalgorithmus für Bildvergleiche(545) ein Standardwert definiert werden. Ist auch dieser leer, wird Pixel für Pixel verglichen, unter Einbeziehung der Option Erlaubte Abweichung beim Check von Abbildern(545). Eine genaue Beschreibung der verfügbaren Algorithmen und ihrer Parameter finden Sie in Details des Algorithmus zum Bildvergleich(1309).

Variabel: Ja

Einschränkungen: Muss spezieller Syntax entsprechen.

Negieren

Ist dieses Attribut gesetzt, wird das Ergebnis des Checks negiert, d.h. die geprüfte Eigenschaft darf in diesem Fall nicht mit dem erwarteten Wert übereinstimmen.

Variabel: Ja

Einschränkungen: Keine

Name des Check-Typs

Hiermit wird konkretisiert, welcher Check ausgeführt werden soll. Dadurch wird es möglich, für eine Komponente verschiedene Checks mit dem selben Datentyp anzubieten, ohne dass es dadurch zu Konflikten kommt. Der standardmäßig zur Verfügung stehende Check-Typ ist 'default'. Für PDF stehen außerdem die in Abschnitt 18.3.2(293) beschriebenen Check-Typen 'skaliert' und 'unskaliert' zur Verfügung. Mit Hilfe eines `Checkers` können zusätzliche Check-Typen implementiert werden (vgl. Abschnitt 54.4(1208)).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf, bis die Komponente den geforderten Zustand angenommen haben muss. Lassen Sie das Attribut leer, oder geben Sie 0 an, wenn Sie keine Verzögerung zulassen wollen.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur(672), Abhängigkeit(630) oder Testfall(599) Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6(116).

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`(963) geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

Name

Ein optionaler Name für den Check, der für mehr Klarheit im Report hilfreich sein kann.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.9.6 Check Geometrie

Überprüft Position und Größe einer Komponente. Der Check funktioniert mit allen Komponenten, nicht aber mit Unterelementen.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Geometriedaten werden zusammen mit den Daten der Zielkomponente an das SUT geschickt. Dort wird die dazu passende Komponente ermittelt und deren Position und Größe mit den Vorgaben verglichen.

Attribute:

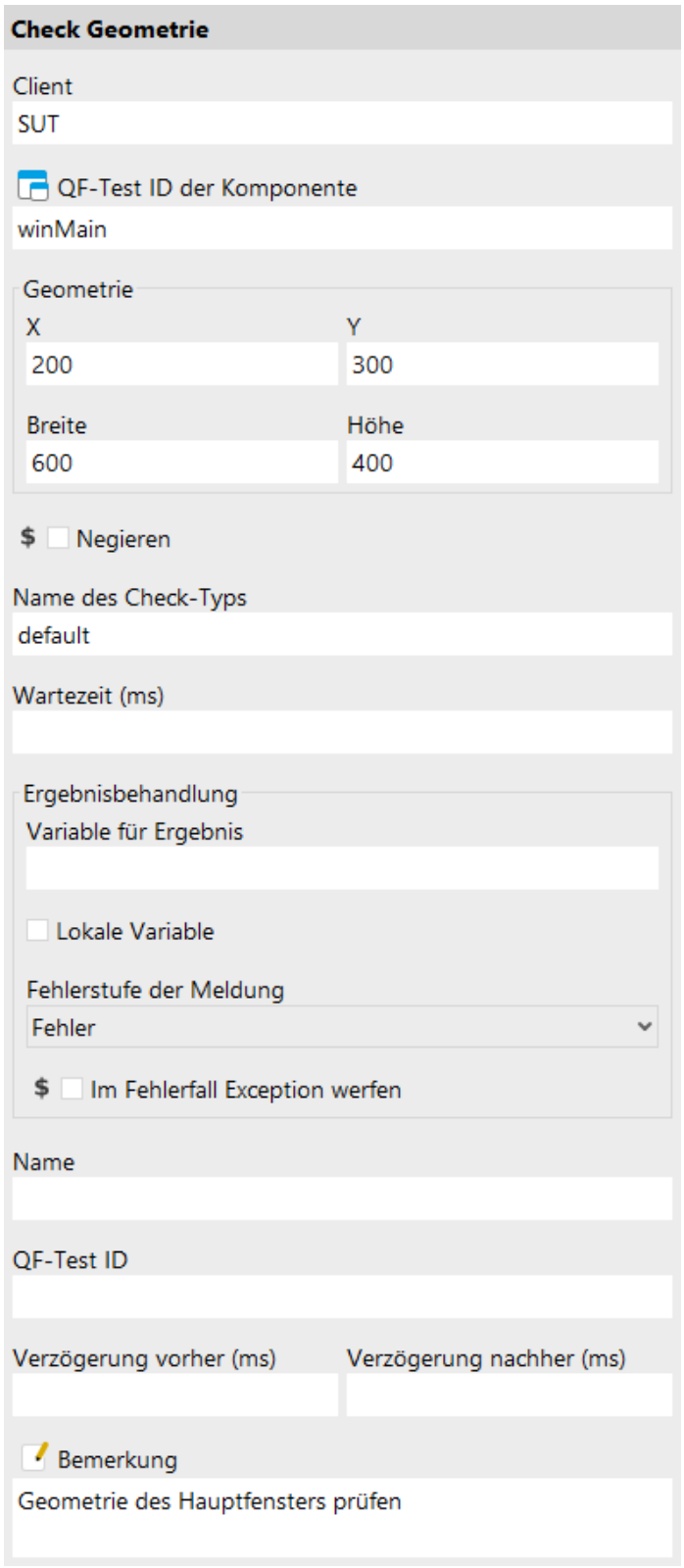


Abbildung 42.67: Check Geometrie-Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, in dem der Check vorgenommen werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens, auf den sich der Check bezieht.

Der "Komponente auswählen" Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1(359)). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9(92)). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID(81) und Komponente-Knoten versus SmartID(51).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Geometrie

Die X/Y Koordinate, Breite und Höhe, mit denen die entsprechenden Werte der Komponente verglichen werden. Wenn Sie nicht alle Werte prüfen wollen, sondern z.B. nur die Position oder nur die Größe, lassen Sie die anderen Werte leer.

Variabel: Ja

Einschränkungen: Gültige Zahlen, Breite und Höhe > 0

Negieren

Ist dieses Attribut gesetzt, wird das Ergebnis des Checks negiert, d.h. die geprüfte Eigenschaft darf in diesem Fall nicht mit dem erwarteten Wert übereinstimmen.

Variabel: Ja

Einschränkungen: Keine

Name des Check-Typs

Hiermit wird konkretisiert, welcher Check ausgeführt werden soll. Dadurch wird es möglich, für eine Komponente verschiedene Checks mit dem selben Datentyp anzubieten, ohne dass es dadurch zu Konflikten kommt. Mit Hilfe eines `Checkers` können zusätzliche Check-Typen implementiert werden (vgl. Abschnitt 54.4(1208)).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf, bis die Komponente den geforderten Zustand angenommen haben muss. Lassen Sie das Attribut leer, oder geben Sie 0 an, wenn Sie keine Verzögerung zulassen wollen.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur(672), Abhängigkeit(630) oder Testfall(599) Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6(116).

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`(963) geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

Name

Ein optionaler Name für den Check, der für mehr Klarheit im Report hilfreich sein kann.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.10 Abfragen

Das Automatisieren eines Tests wird dann schwierig, wenn sich das SUT dynamisch verhält, d.h. wenn sich Dinge bei jedem Programmlauf anders verhalten. Das ist z.B. der Fall, wenn IDs für Datenbankfelder generiert werden, die aktuelle Uhrzeit Eingang in Datensätze findet, etc.

Neben seinen Kontrollstrukturen bietet QF-Test die Möglichkeit, Daten aus Komponenten der Oberfläche des SUT auszulesen und in der Testsuite weiter zu verwenden, z.B. als Index für ein Element[936], oder als Vergleichswert für einen Check[805].

42.10.1 Text auslesen

Mit diesem Knoten kann zur Laufzeit eines Tests der dargestellte Text einer Komponente oder eines Unterelements ausgelesen und einer lokalen oder globalen Variable zugewiesen werden. Um unerwünschte Seiteneffekte zu vermeiden wird der Inhalt der Variable beim Auslesen nicht automatisch expandiert.

Diese Operation macht nur für Komponenten oder Elemente Sinn, die einen Text darstellen. Die Angabe einer unzulässigen Komponente führt zu einer `OperationNotSupportedException`, ein überflüssiger Index zu einer `UnexpectedIndexException`. Die folgende Tabelle führt die möglichen Zielkomponenten und Unterelemente auf. (P/S) steht für Primärindex/Sekundärindex.

Web

Bei Web-Anwendungen kann theoretisch jeder Knoten einen Text beinhalten, deshalb wird dort entweder der Text oder ein leerer Text zurückgegeben und des kommt nicht zu einer `OperationNotSupportedException`.

Klasse	Index (P/S)	Ergebnis
`AbstractButton`	-/-	`getText()`
`Dialog`	-/-	`getTitle()`
`Frame`	-/-	`getTitle()`
`JComboBox`	-/-	aktueller Wert (mit Renderer)
`JComboBox`	Listenelement/-	Listenelement (mit Renderer)
`JEditorPane`	Zeichenposition/-	Strukturelement an Position (experimentell)
`JLabel`	-/-	`getText()`
`JList`	Listenelement/-	Element (mit Renderer)
`JTabbedPane`	Tab/-	Titel des Tab
`JTable`	Spalte/Zeile	Zelleninhalt (mit Renderer)
`JTableHeader`	Spalte/-	Spaltentitel (mit Renderer)
`JTextArea`	Zeilennummer/-	Text in der Zeile
`JTextComponent`	-/-	`getText()`
`JTree`	Knoten/-	Knoten (mit Renderer)
`Label`	-/-	`getText()`
`TextField`	-/-	`getText()`

Tabelle 42.25: Zulässige Komponenten für Text auslesen

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Daten der Zielkomponente werden an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT die passende Komponente, liest deren Text aus und liefert ihn zurück an QF-Test, wo er in einer globalen Variable abgelegt wird.

Attribute:

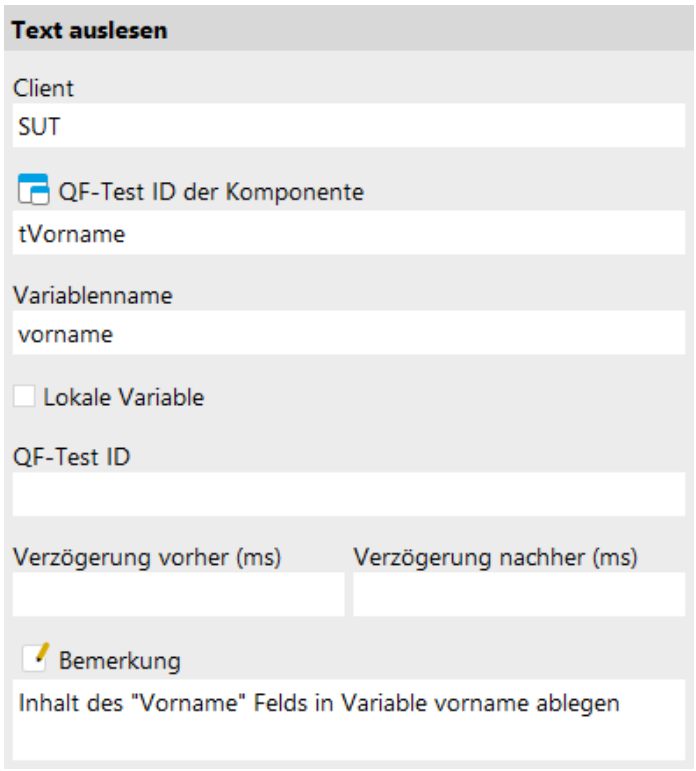


Abbildung 42.68: Text auslesen Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, aus dem die Daten gelesen werden sollen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens dessen Daten abgefragt werden.

Der ”Komponente auswählen” Button öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Variablenname

Der Name der globalen Variable, in der das Ergebnis der Abfrage abgelegt wird (vgl. Kapitel 6[116]).

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur[672], Abhängigkeit[630] oder Testfall[599] Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6[116].

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren[593] kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.10.2 Index auslesen

Hiermit kann die Position eines Unterelements einer komplexen Komponente ermittelt werden, dessen Text bekannt ist. Als Ziel muss daher ein Unterelement angegeben werden. Das Ergebnis wird einer lokalen oder globalen Variable zugewiesen.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Die Daten der Zielkomponente werden an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT die passende Komponente, sucht das Unterelement und liefert seine Position zurück an QF-Test, wo sie in einer globalen Variable abgelegt wird.

Attribute:

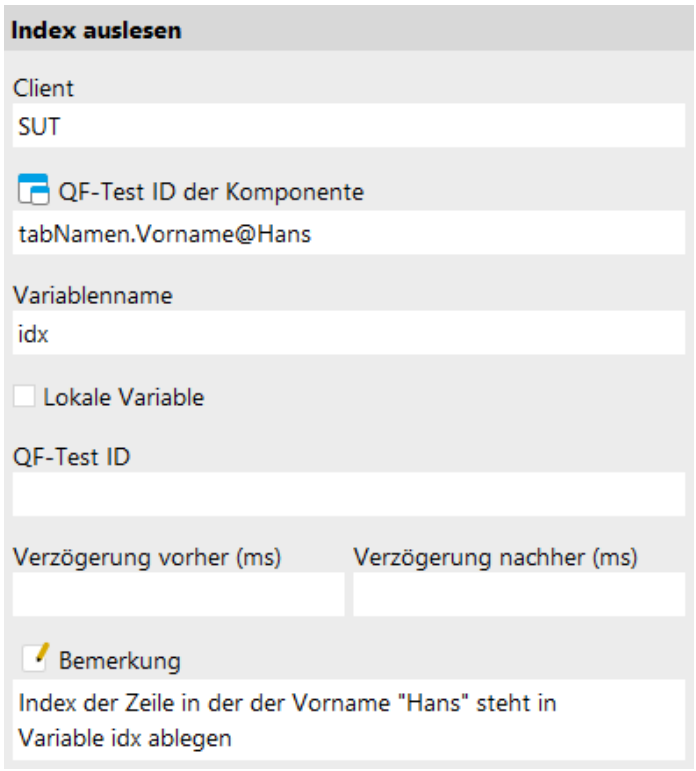


Abbildung 42.69: Index auslesen Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, aus dem die Daten gelesen werden sollen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens dessen Daten abgefragt werden.

Der ”Komponente auswählen” Button öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Variablenname

Der Name der globalen Variable, in der das Ergebnis der Abfrage abgelegt wird (vgl. Kapitel 6[116]).

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur[672], Abhängigkeit[630] oder Testfall[599] Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6[116].

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren[593] kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.10.3 Geometrie auslesen

Hiermit ermitteln Sie die Geometrie eines Fensters, einer Komponente oder eines Unterelements im SUT. Das Ergebnis wird in bis zu vier lokalen oder globalen Variablen abgelegt, je einer für X und Y-Koordinaten, Breite und Höhe.

Interessant ist dieser Knoten z.B. dann, wenn Sie einen Mausevent[775] relativ zur rechten oder unteren Kante einer Komponente definieren wollen. Lesen Sie hierzu zunächst Breite und Höhe der Komponente aus und definieren Sie dann X[777] und Y[777] Koordinate des Mausevents über die erweiterte Variablensyntax für Berechnungen (vgl. Abschnitt 11.2[189]).

Die folgende Tabelle führt die möglichen Unterelemente auf. (P/S) steht für Primärindex/Sekundärindex.

Klasse	Index (P/S)	Ergebnis
`JList`	Listenelement/-	Element
`JTabbedPane`	Tab/-	Tab
`JTable`	Spalte/-	Spalte
`JTable`	Spalte/Zeile	Zelle
`JTableHeader`	Spalte/-	Spaltentitel
`JTree`	Knoten/-	Knoten

Tabelle 42.26: Zulässige Unterelemente für Geometrie auslesen

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Daten der Zielkomponente werden an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT die passende Komponente, berechnet deren Geometrie und schickt die Daten zurück an QF-Test, wo sie in globalen Variablen abgelegt werden.

Attribute:

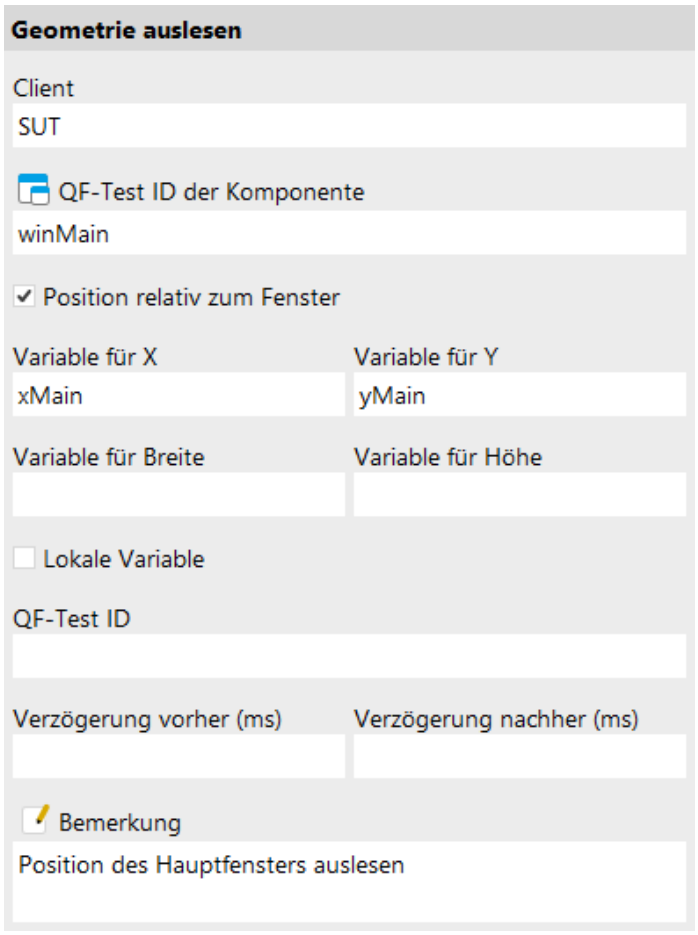


Abbildung 42.70: Geometrie auslesen Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, aus dem die Daten gelesen werden sollen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster(919), Komponente(930) oder Element(936) Knotens dessen Daten abgefragt werden.

Der "Komponente auswählen" Button öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5(47)) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld

befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1(359)). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9(92)). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID(81) und Komponente-Knoten versus SmartID(51).

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Position relativ zum Fenster

Legt fest, ob die X und Y-Koordinaten einer Komponente oder eines Unterelements relativ zum Fenster oder relativ zur Parentkomponente sind.

Variabel: Nein

Einschränkungen: Keine.

Variable für X

Der Name der globalen Variable, in der die X-Koordinate abgelegt wird (vgl. Kapitel 6(116)). Wenn Sie sich für die X-Koordinate nicht interessieren, lassen Sie dieses Feld leer.

Variabel: Ja

Einschränkungen: Keine

Variable für Y

Der Name der globalen Variable, in der die Y-Koordinate abgelegt wird (vgl. Kapitel 6(116)). Wenn Sie sich für die Y-Koordinate nicht interessieren, lassen Sie dieses Feld leer.

Variabel: Ja

Einschränkungen: Keine

Variable für Breite

Der Name der globalen Variable, in der die Breite abgelegt wird (vgl. Kapitel 6(116)). Wenn Sie sich für die Breite nicht interessieren, lassen Sie dieses Feld leer.

Variabel: Ja

Einschränkungen: Keine

Variable für Höhe

Der Name der globalen Variable, in der die Höhe abgelegt wird (vgl. Kapitel 6[116]). Wenn Sie sich für die Höhe nicht interessieren, lassen Sie dieses Feld leer.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, werden die Variablen in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - jeweils die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur[672], Abhängigkeit[630] oder Testfall[599] Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6[116].

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren[593] kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11 Verschiedenes

Dieser Abschnitt führt die restlichen Knotentypen auf, die in keine der bisher behandelten Kategorien fallen.

42.11.1 Kommentar

Dieser Knoten ist zu Dokumentationszwecken gedacht. Sie können ihn dafür benutzen, Kommentare zu Ihrer Testsuite hinzuzufügen.

Enthalten in: Überall

Kinder: Keine

Ausführung: Ein Kommentarknoten beeinflusst nicht die Testausführung.

Attribute:

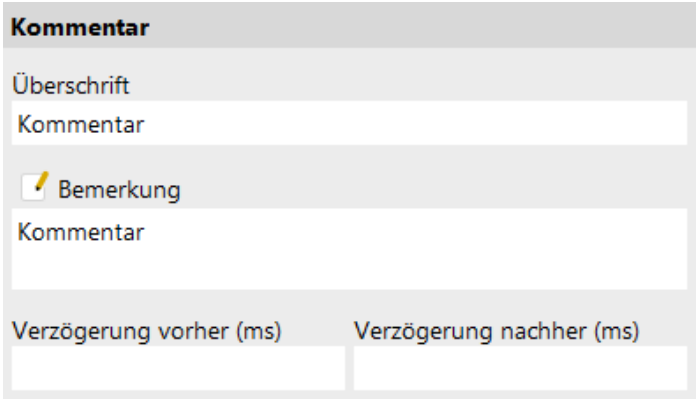


Abbildung 42.71: Attribute des Kommentar Knotens

Überschrift

Der Text, der im Baum angezeigt wird - eine Zusammenfassung des Kommentares oder der Kommentar selbst.

In diesem Attribut ist es möglich die folgenden HTML-Tags kursiver Text, <u>unterstrichener text</u>, <s>durchgestrichener text</s> und fetter text zu benutzen um die Repräsentation im Baum aufzuhübschen. Desweiteren sind farbliche Unterlegungen mittels style="color:farbname" und color="farbname" möglich, auch im Zusammenhang mit dem , , <font> und Tag möglich.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.2 Fehler

Mit diesem Knoten können Sie einen Fehler in das Protokoll schreiben. Über die Attribute kann gesteuert werden, welche Informationen in das Protokoll geschrieben werden.

Der Fehler Knoten kann Skripte ersetzen, die ausschließlich dazu genutzt werden `rc.logError` auszuführen. Somit kann er ebenfalls Aufrufe der Prozedur `qfs.run-log.logError` aus der mitgelieferten Standardbibliothek(183) ersetzen.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Der Fehler Knoten schreibt einen Fehler in das Protokoll.

Attribute:

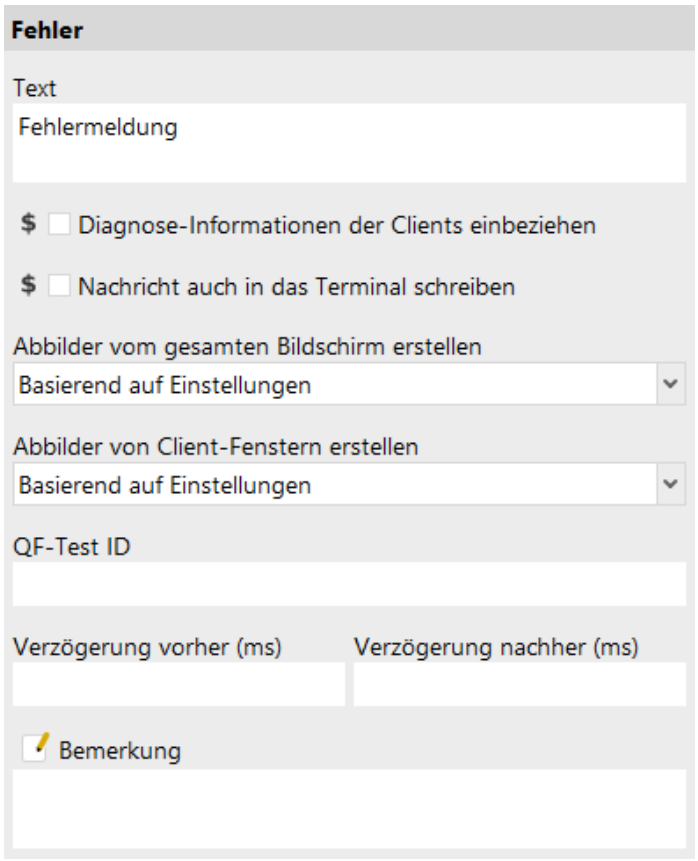


Abbildung 42.72: Attribute des Fehler Knotens

Text

Der Text der Fehlermeldung. Im Baumknoten werden lange Texte abgeschnitten.

Variabel: Ja

Einschränkungen: Keine

Diagnose-Informationen der Clients einbeziehen

Wenn die Option selektiert ist, werden zusätzliche Informationen über jeden verbundenen Client Prozess protokolliert.

Variabel: Ja

Einschränkungen: Keine

Nachricht auch in das Terminal schreiben

Falls aktiv wird die Nachricht zusätzlich in das QF-Test Terminal geschrieben.

Variabel: Ja

Einschränkungen: Keine

Abbilder vom gesamten Bildschirm erstellen

Das Attribut steuert, ob komplette Abbilder des oder der angeschlossenen Monitore in das Protokoll geschrieben werden, wobei die sicherheits- und datenschutzrelevante Option Abbilder auf relevante Bildschirme beschränken(588) berücksichtigt wird. Die Standardeinstellung dieser Option lässt nur Aufnahmen von Bildschirmen zu, auf denen entweder QF-Test oder Fenster der getesteten Applikation(en) zu sehen sind. Im Batchmodus (kein QF-Test GUI) werden daher standardmäßig keine Bildschirmabbilder protokolliert, wenn kein SUT Fenster sichtbar ist.

Einstellung	Beschreibung
Immer	Bildschirmabbilder werden immer ins Protokoll geschrieben Wenn man diese Option wählt, muss man sich bewusst sein, dass dies schnell zu Speicherproblemen führen kann, wenn viele Fehler auftreten. Wir empfehlen in diesem Fall geteilte Protokolle zu verwenden, siehe Optionen zur Aufteilung von Protokollen(582). Die Option setzt die folgenden Einstellungen außer Kraft: • Maximale Anzahl von Fehlern mit Bildschirmabbild pro Protokoll(588) • Bildschirmabbilder für geteilte Protokolle separat zählen(588) • Bei einem Fehler Abbilder vom gesamten Bildschirm erstellen(588)
Nie	Es werden keine Bildschirmabbilder in das Protokoll geschrieben.
Basierend auf Einstellungen	Bildschirmabbilder werden gemäß der eingestellten Optionen (siehe Optionen für den Inhalt von Protokollen(586)) erstellt.
Variable, zum Beispiel `$(logScreenshots)`	Referenz auf eine Variable, die einen der folgenden Werte enthält, wobei die Groß-/Kleinschreibung irrelevant ist. Die Werte `immer`, `always`, `1`, `true`, `ja`, `wahr` oder `yes` weisen QF-Test an, immer Bildschirmabbilder loggen soll. Die Werte `nie`, `never`, `0`, `false`, `nein`, `falsch` oder `no` weisen QF-Test an, nie Bildschirmabbilder zu erstellen. Die Werte `basierend auf einstellungen`, `based on options`, `options`, `option`, `einstellung` oder `einstellungen` weisen QF-Test an, die gesetzten Optionen zu berücksichtigen.

Tabelle 42.27: Einstellungen für ”Bildschirmabbilder erstellen”

Variabel: Ja

Einschränkungen: Keine

Abbilder von Client-Fenstern erstellen

Das Attribut steuert wie entschieden wird, ob Abbilder der Fenster der getesteten Applikationen in das Protokoll geschrieben werden, auch wenn sie durch andere Fenster verdeckt sein sollten.

Einstellung	Beschreibung
Immer	Abbilder der SUT-Fenster werden immer ins Protokoll geschrieben. Wenn man diese Option wählt, muss man sich bewusst sein, dass dies schnell zu Speicherproblemen führen kann, wenn viele Fehler protokolliert werden. Wir empfehlen in diesem Fall geteilte Protokolle zu verwenden, siehe Optionen zur Aufteilung von Protokollen(582). Die folgenden Optionen werden ignoriert: • Maximale Anzahl von Fehlern mit Bildschirmabbild pro Protokoll(588) • Bildschirmabbilder für geteilte Protokolle separat zählen(588) • Bei einem Fehler im Client Abbilder der Fenster des Clients erstellen(588) • Bei einem Fehler Abbilder aller Client-Fenster erstellen(589)
Nie	Es werden keine Bildschirmabbilder in das Protokoll geschrieben.
Basierend auf Einstellungen	Bildschirmabbilder werden gemäß der eingestellten Optionen (siehe Optionen für den Inhalt von Protokollen(586)) erstellt. Die Einstellung von Bei einem Fehler im Client Abbilder der Fenster des Clients erstellen(588) ist irrelevant, da der Knoten für alle Clients greift, analog zu einem Server-Skript.
Variable, zum Beispiel `$(logScreenshots)`	Referenz auf eine Variable, die einen der folgenden Werte enthält, wobei die Groß-/Kleinschreibung irrelevant ist. Die Werte `immer`, `always`, `1`, `true`, `ja`, `wahr` oder `yes` weisen QF-Test an, immer Bildschirmabbilder loggen soll. Die Werte `nie`, `never`, `0`, `false`, `nein`, `falsch` oder `no` weisen QF-Test an, nie Bildschirmabbilder zu erstellen. Die Werte `basierend auf einstellungen`, `based on options`, `options`, `option`, `einstellung` oder `einstellungen` weisen QF-Test an, die gesetzten Optionen zu berücksichtigen.

Tabelle 42.28: Einstellungen für ”Client-Bildschirmabbilder erstellen”

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.3 Warnung

Mit diesem Knoten können Sie eine Warnung in das Protokoll schreiben. Über die Attribute kann gesteuert werden, welche Informationen in das Protokoll geschrieben werden.

Der Warnung Knoten kann Skripte ersetzen, die ausschließlich dazu genutzt werden `rc.logWarning` auszuführen. Somit kann er ebenfalls Aufrufe der Prozedur `qfs.run-log.logWarning` aus der mitgelieferten Standardbibliothek(183) ersetzen.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Der Warnung Knoten schreibt eine Warnung in das Protokoll.

Attribute:

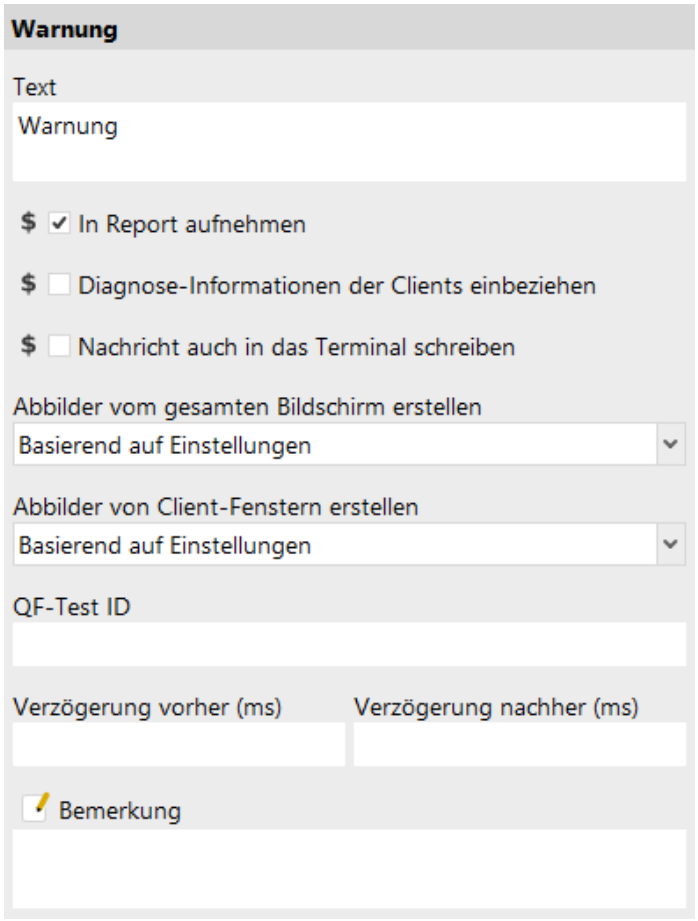


Abbildung 42.73: Attribute des Warnung Knotens

Text

Der Text zur Warnung. Im Baumknoten werden lange Texte abgeschnitten.

Variabel: Ja

Einschränkungen: Keine

In Report aufnehmen

Wenn die Option selektiert ist, wird der Knoten im Report aufgeführt.

Variabel: Ja

Einschränkungen: Keine

Diagnose-Informationen der Clients einbeziehen

Wenn die Option selektiert ist, werden zusätzliche Informationen über jeden verbundenen Client Prozess protokolliert.

Variabel: Ja

Einschränkungen: Keine

Nachricht auch in das Terminal schreiben

Falls aktiv wird die Nachricht zusätzlich in das QF-Test Terminal geschrieben.

Variabel: Ja

Einschränkungen: Keine

Abbilder vom gesamten Bildschirm erstellen

Das Attribut steuert, ob komplette Abbilder des oder der angeschlossenen Monitore in das Protokoll geschrieben werden, wobei die sicherheits- und datenschutzrelevante Option Abbilder auf relevante Bildschirme beschränken(588) berücksichtigt wird. Die Standardeinstellung dieser Option lässt nur Aufnahmen von Bildschirmen zu, auf denen entweder QF-Test oder Fenster der getesteten Applikation(en) zu sehen sind. Im Batchmodus (kein QF-Test GUI) werden daher standardmäßig keine Bildschirmabbilder protokolliert, wenn kein SUT Fenster sichtbar ist.

Einstellung	Beschreibung
Immer	Bildschirmabbilder werden immer ins Protokoll geschrieben Wenn man diese Option wählt, muss man sich bewusst sein, dass dies schnell zu Speicherproblemen führen kann, wenn viele Fehler auftreten. Wir empfehlen in diesem Fall geteilte Protokolle zu verwenden, siehe Optionen zur Aufteilung von Protokollen(582). Die Option setzt die folgenden Einstellungen außer Kraft: • Maximale Anzahl von Fehlern mit Bildschirmabbild pro Protokoll(588) • Bildschirmabbilder für geteilte Protokolle separat zählen(588) • Bei einem Fehler Abbilder vom gesamten Bildschirm erstellen(588) • Bei Warnungen Screenshots erstellen(589)
Nie	Es werden keine Bildschirmabbilder in das Protokoll geschrieben.
Basierend auf Einstellungen	Bildschirmabbilder werden gemäß der eingestellten Optionen (siehe Optionen für den Inhalt von Protokollen(586)) erstellt. Bitte beachten: Es werden nur dann Bildschirmabbilder erstellt, wenn die Option Bei Warnungen Screenshots erstellen(589) aktiviert wurde.
Variable, zum Beispiel `$(logScreenshots)`	Referenz auf eine Variable, die einen der folgenden Werte enthält, wobei die Groß-/Kleinschreibung irrelevant ist. Die Werte `immer`, `always`, `1`, `true`, `ja`, `wahr` oder `yes` weisen QF-Test an, immer Bildschirmabbilder loggen soll. Die Werte `nie`, `never`, `0`, `false`, `nein`, `falsch` oder `no` weisen QF-Test an, nie Bildschirmabbilder zu erstellen. Die Werte `basierend auf einstellungen`, `based on options`, `options`, `option`, `einstellung` oder `einstellungen` weisen QF-Test an, die gesetzten Optionen zu berücksichtigen.

Tabelle 42.29: Einstellungen für ”Bildschirmabbilder erstellen”

Variabel: Ja

Einschränkungen: Keine

Abbilder von Client-Fenstern erstellen

Das Attribut steuert wie entschieden wird, ob Abbilder der Fenster der getesteten Applikationen in das Protokoll geschrieben werden, auch wenn sie durch andere Fenster verdeckt sein sollten.

Einstellung	Beschreibung
Immer	Abbilder der SUT-Fenster werden immer ins Protokoll geschrieben. Wenn man diese Option wählt, muss man sich bewusst sein, dass dies schnell zu Speicherproblemen führen kann, wenn viele Fehler protokolliert werden. Wir empfehlen in diesem Fall geteilte Protokolle zu verwenden, siehe Optionen zur Aufteilung von Protokollen(582). Die folgenden Optionen werden ignoriert: • Maximale Anzahl von Fehlern mit Bildschirmabbild pro Protokoll(588) • Bildschirmabbilder für geteilte Protokolle separat zählen(588) • Bei einem Fehler im Client Abbilder der Fenster des Clients erstellen(588) • Bei einem Fehler Abbilder aller Client-Fenster erstellen(589) • Bei Warnungen Screenshots erstellen(589)
Nie	Es werden keine Bildschirmabbilder in das Protokoll geschrieben.
Basierend auf Einstellungen	Bildschirmabbilder werden gemäß der eingestellten Optionen (siehe Optionen für den Inhalt von Protokollen(586)) erstellt. Die Einstellung von Bei einem Fehler im Client Abbilder der Fenster des Clients erstellen(588) ist irrelevant, da der Knoten für alle Clients greift, analog zu einem Server-Skript. Bitte beachten: Es werden nur dann Bildschirmabbilder erstellt, wenn die Option Bei Warnungen Screenshots erstellen(589) aktiviert wurde.
Variable, zum Beispiel `$(logScreenshots)`	Referenz auf eine Variable, die einen der folgenden Werte enthält, wobei die Groß-/Kleinschreibung irrelevant ist. Die Werte `immer`, `always`, `1`, `true`, `ja`, `wahr` oder `yes` weisen QF-Test an, immer Bildschirmabbilder loggen soll. Die Werte `nie`, `never`, `0`, `false`, `nein`, `falsch` oder `no` weisen QF-Test an, nie Bildschirmabbilder zu erstellen. Die Werte `basierend auf einstellungen`, `based on options`, `options`, `option`, `einstellung` oder `einstellungen` weisen QF-Test an, die gesetzten Optionen zu berücksichtigen.

Tabelle 42.30: Einstellungen für ”Client-Bildschirmabbilder erstellen”

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.4 Nachricht

Mit diesem Knoten können Sie eine Meldung in das Protokoll schreiben. Über die Attribute kann gesteuert werden, welche Informationen in das Protokoll geschrieben werden.

Der Nachricht Knoten kann Skripte ersetzen, die ausschließlich dazu genutzt werden `rc.logMessage` auszuführen. Somit kann er ebenfalls Aufrufe der Prozedur `qfs.run-log.logMessage` aus der mitgelieferten Standardbibliothek(183) ersetzen.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Der Nachricht Knoten schreibt eine Meldung in das Protokoll.

Attribute:

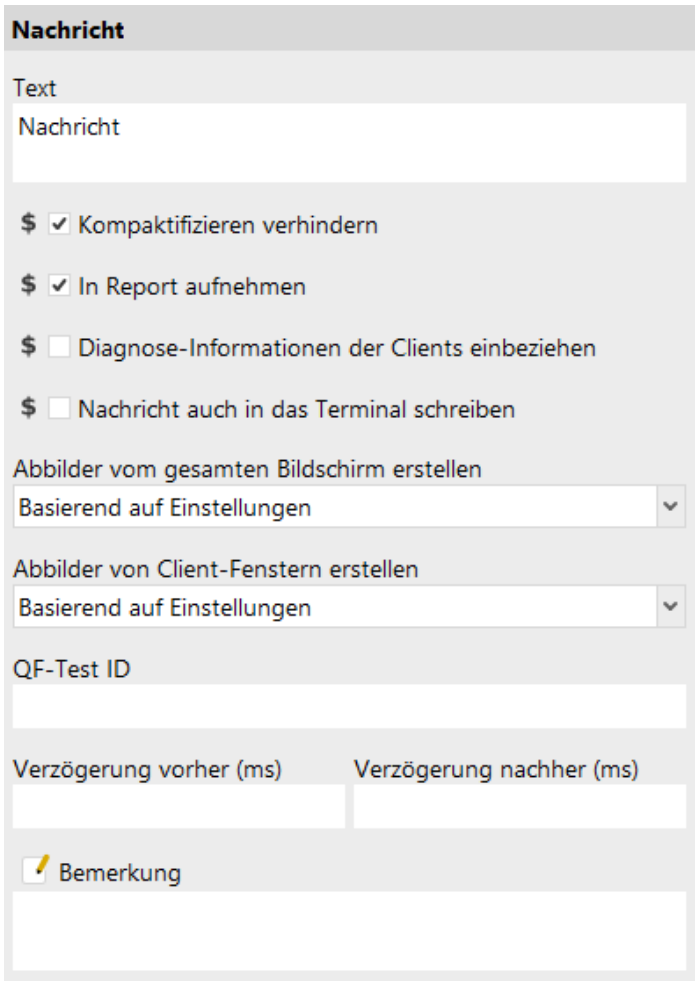


Abbildung 42.74: Attribute des Nachricht Knotens

Text

Der Text der Meldung. Im Baumknoten werden lange Texte abgeschnitten.

Variabel: Ja

Einschränkungen: Keine

Kompaktifizieren verhindern

Diese Option hat nur einen Einfluss wenn eine Nachricht geloggt wird und gleichzeitig kompakte Protokolle (siehe Option Kompakte Protokolle erstellen[590]) verwendet werden. In diesem Fall verhindert das Anhaken dieser Option, dass der Knoten aus dem Protokoll entfernt wird.

Variabel: Ja

Einschränkungen: Keine

In Report aufnehmen

Wenn die Option selektiert ist, wird der Knoten im Report aufgeführt.

Variabel: Ja

Einschränkungen: Keine

Diagnose-Informationen der Clients einbeziehen

Wenn die Option selektiert ist, werden zusätzliche Informationen über jeden verbundenen Client Prozess protokolliert.

Variabel: Ja

Einschränkungen: Keine

Nachricht auch in das Terminal schreiben

Falls aktiv wird die Nachricht zusätzlich in das QF-Test Terminal geschrieben.

Variabel: Ja

Einschränkungen: Keine

Abbilder vom gesamten Bildschirm erstellen

Das Attribut steuert, ob komplette Abbilder des oder der angeschlossenen Monitore in das Protokoll geschrieben werden, wobei die sicherheits- und datenschutzrelevante Option Abbilder auf relevante Bildschirme beschränken[588] berücksichtigt wird. Die Standardeinstellung dieser Option lässt nur Aufnahmen von Bildschirmen zu, auf denen entweder QF-Test oder Fenster der getesteten Applikation(en) zu sehen sind. Im Batchmodus (kein QF-Test GUI) werden daher standardmäßig keine Bildschirmabbilder protokolliert, wenn kein SUT Fenster sichtbar ist.

Einstellung	Beschreibung
Immer	Bildschirmabbilder werden immer ins Protokoll geschrieben Wenn man diese Option wählt, muss man sich bewusst sein, dass dies schnell zu Speicherproblemen führen kann, wenn viele Fehler auftreten. Wir empfehlen in diesem Fall geteilte Protokolle zu verwenden, siehe Optionen zur Aufteilung von Protokollen(582). Die Option setzt die folgenden Einstellungen außer Kraft: • Maximale Anzahl von Fehlern mit Bildschirmabbild pro Protokoll(588) • Bildschirmabbilder für geteilte Protokolle separat zählen(588) • Bei einem Fehler Abbilder vom gesamten Bildschirm erstellen(588)
Nie	Es werden keine Bildschirmabbilder in das Protokoll geschrieben.
Basierend auf Einstellungen	Bildschirmabbilder werden gemäß der eingestellten Optionen (siehe Optionen für den Inhalt von Protokollen(586)) erstellt.
Variable, zum Beispiel `$(logScreenshots)`	Referenz auf eine Variable, die einen der folgenden Werte enthält, wobei die Groß-/Kleinschreibung irrelevant ist. Die Werte `immer`, `always`, `1`, `true`, `ja`, `wahr` oder `yes` weisen QF-Test an, immer Bildschirmabbilder loggen soll. Die Werte `nie`, `never`, `0`, `false`, `nein`, `falsch` oder `no` weisen QF-Test an, nie Bildschirmabbilder zu erstellen. Die Werte `basierend auf einstellungen`, `based on options`, `options`, `option`, `einstellung` oder `einstellungen` weisen QF-Test an, die gesetzten Optionen zu berücksichtigen.

Tabelle 42.31: Einstellungen für ”Bildschirmabbilder erstellen”

Variabel: Ja

Einschränkungen: Keine

Abbilder von Client-Fenstern erstellen

Das Attribut steuert wie entschieden wird, ob Abbilder der Fenster der getesteten Applikationen in das Protokoll geschrieben werden, auch wenn sie durch andere Fenster verdeckt sein sollten.

Einstellung	Beschreibung
Immer	Abbilder der SUT-Fenster werden immer ins Protokoll geschrieben. Wenn man diese Option wählt, muss man sich bewusst sein, dass dies schnell zu Speicherproblemen führen kann, wenn viele Fehler protokolliert werden. Wir empfehlen in diesem Fall geteilte Protokolle zu verwenden, siehe Optionen zur Aufteilung von Protokollen(582). Die folgenden Optionen werden ignoriert: • Maximale Anzahl von Fehlern mit Bildschirmabbild pro Protokoll(588) • Bildschirmabbilder für geteilte Protokolle separat zählen(588) • Bei einem Fehler im Client Abbilder der Fenster des Clients erstellen(588) • Bei einem Fehler Abbilder aller Client-Fenster erstellen(589)
Nie	Es werden keine Bildschirmabbilder in das Protokoll geschrieben.
Basierend auf Einstellungen	Bildschirmabbilder werden gemäß der eingestellten Optionen (siehe Optionen für den Inhalt von Protokollen(586)) erstellt. Die Einstellung von Bei einem Fehler im Client Abbilder der Fenster des Clients erstellen(588) ist irrelevant, da der Knoten für alle Clients greift, analog zu einem Server-Skript.
Variable, zum Beispiel `$(logScreenshots)`	Referenz auf eine Variable, die einen der folgenden Werte enthält, wobei die Groß-/Kleinschreibung irrelevant ist. Die Werte `immer`, `always`, `1`, `true`, `ja`, `wahr` oder `yes` weisen QF-Test an, immer Bildschirmabbilder loggen soll. Die Werte `nie`, `never`, `0`, `false`, `nein`, `falsch` oder `no` weisen QF-Test an, nie Bildschirmabbilder zu erstellen. Die Werte `basierend auf einstellungen`, `based on options`, `options`, `option`, `einstellung` oder `einstellungen` weisen QF-Test an, die gesetzten Optionen zu berücksichtigen.

Tabelle 42.32: Einstellungen für ”Client-Bildschirmabbilder erstellen”

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.5 Variable setzen

Mit diesem Knoten können Sie zur Laufzeit eines Tests den Wert einer globalen Variable setzen. Wenn der Test interaktiv aus QF-Test gestartet wurde und nicht mittels `qftest -batch` (vgl. Abschnitt 1.7[13]), können Sie den Wert auf Wunsch interaktiv über einen Dialog festlegen.

Um einen Wert nach Eingabe in dieses Feld zu verschlüsseln, wählen Sie nach einem Rechts-Klick Text verschlüsseln aus dem resultierenden Popupmenü. Geben Sie auf jeden Fall vor der Verschlüsselung einen Salt in der Option Salt für Verschlüsselung von Kennwörtern[532] an.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Wenn der Test interaktiv abläuft und das Interaktiv[874] Attribut gesetzt ist, wird ein Dialog geöffnet, in dem der Wert für die Variable eingegeben werden kann. Verstreicht die Wartezeit[875] oder wird der Wert mit dem *OK* Button bestätigt, wird die Variable entsprechend in den globalen Variablen gesetzt. Bricht der Anwender den Dialog mit dem *Abbrechen* Button ab, wird der Test beendet. Im nicht-interaktiven Fall wird die Variable direkt auf den Defaultwert[874] gesetzt.

Attribute:

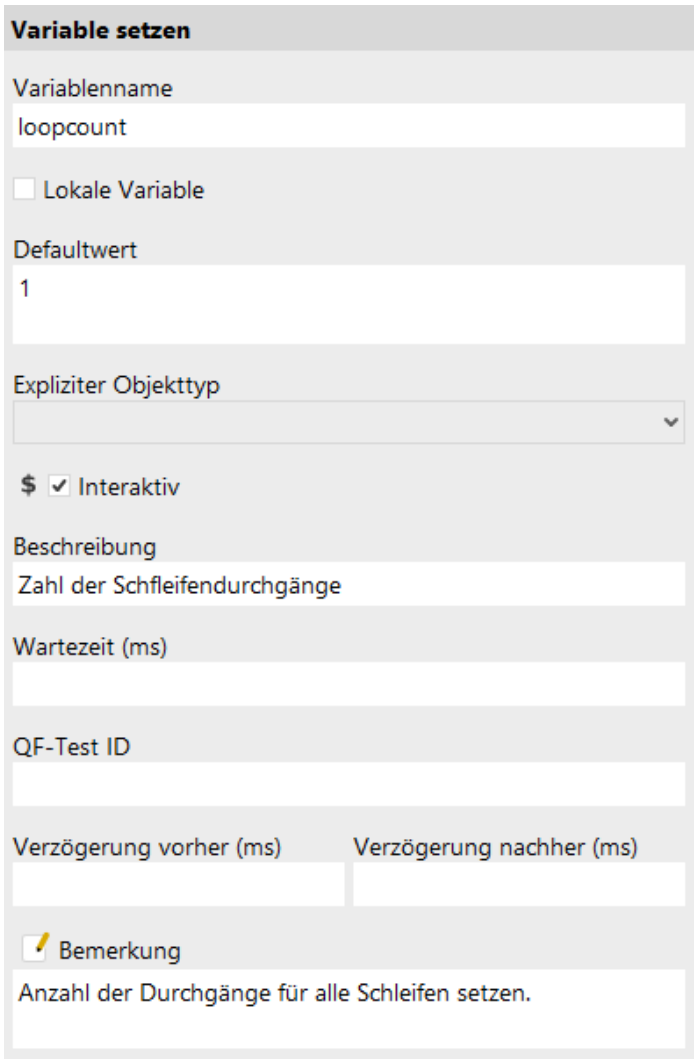


Abbildung 42.75: Variable setzen Attribute

Variablenname

Der Name der globalen Variable, der der Wert zugewiesen wird (vgl. Kapitel 6[116]).

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur[672], Abhängigkeit[630] oder Testfall[599] Knotens liegt. Gibt es keine solche Definition, wird

eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6(116).

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Defaultwert

Der Standardwert für die Variable, falls der Test nicht interaktiv abläuft, das Interaktiv(874) Attribut nicht gesetzt ist oder die Wartezeit(875) verstreicht.

Variabel: Ja

Einschränkungen: Keine

Expliziter Objekttyp

9.0+ In QF-Test Variablen können neben Zeichenketten auch beliebige andere Objekte gespeichert werden. Im Textfeld für den Wert kann zwar nur eine Zeichenkette angegeben werden, doch lässt sich hiermit festlegen, wie die Eingabe von QF-Test interpretiert werden soll:

- Keine Auswahl: Die Eingabe wird nicht weiter interpretiert. In den meisten Fällen wird das gespeicherte Objekt eine Zeichenkette sein, es sei denn, die Eingabe wurde durch Variablen-Expansion vollständig durch eine andere Variable ersetzt, wodurch deren Objekt unverändert übernommen wird.
- String: Die Eingabe wird in eine Zeichenkette umgewandelt.
- Boolean: Die Eingabe wird in einen boolschen Wahrheitswert umgewandelt, wobei `0`, leere Strings sowie die Zeichenketten `false`, `no` und `nein` als `false`, andere Werte als `true` ausgewertet werden.
- Number: Die Eingabe wird in eine Zahl umgewandelt. Je nach Eingabewert kann dies ein Integer, Long, BigInteger, Double oder ein BigDecimal sein. Schlägt die Umwandlung fehl, so wird eine `ValueCastException`(966) geworfen.
- Object aus JSON: Die Eingabe wird als JSON-String interpretiert und in verschachtelte Maps und Lists mit Strings, Numbers und Booleans umgewandelt. Schlägt die Umwandlung fehl, so wird eine `ValueCastException`(966) geworfen.

Interaktiv

Legt fest, ob der Wert über einen Dialog vom Anwender bestimmt werden kann, sofern der Test selbst interaktiv abläuft.

Variabel: Ja

Einschränkungen: Keine

Beschreibung

Eine kurze Beschreibung, die im Dialog angezeigt wird. Ist dieser Wert leer, wird als Beschreibung `Wert für` <Variablenname> verwendet.

Variabel: Ja

Einschränkungen: Keine

Wartezeit

Ein optionales Zeitlimit für die Eingabe. Wird der Dialog angezeigt und verstreicht die angegebene Zeitspanne, ohne dass der Wert verändert wurde, wird der Dialog automatisch geschlossen und der Defaultwert(874) übernommen. Damit kann das blockieren eines Tests verhindert werden, der zwar interaktiv gestartet wurde aber unbeaufsichtigt ablaufen soll.

Variabel: Ja

Einschränkungen: Leer oder > 0.

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.6 Warten auf Komponente

Dieser Knoten ist sehr wichtig für das Timing eines Testlaufs. Je nach Auslastung des Testsystems kann es unterschiedlich lange dauern, bis z.B. ein neu geöffnetes Fenster tatsächlich erscheint. Mit diesem Knoten läßt sich der Test so lange anhalten, bis eine gewünschte Komponente oder ein Unterelement verfügbar ist. Das Resultat des Knotens kann auch in einer Variable mittels des Attributs Variable für Ergebnis gesetzt werden. Mit Ausschalten des Attributs Im Fehlerfall Exception werfen kann das Werfen der Exceptions unterdrückt werden.

In den globalen Optionen läßt sich eine Zeitspanne einstellen, die grundsätzlich verstreichen darf, bis eine beliebige Komponente verfügbar ist, sowie ein zusätzliches Timeout zum Warten auf ein Unterelement. Da diese Timeouts für alle Events gelten, sollten sie nicht größer als ca. zwei bis fünf Sekunden gewählt werden. Dieser Knoten erlaubt es, diese Timeouts in einzelnen Fällen zu vergrößern, ohne damit den gesamten Testlauf unnötig zu verlangsamen.

Durch Setzen des Warten auf Verschwinden[878] Attributs kann dieser Knoten auch dazu verwendet werden, auf das Verschwinden einer Komponente zu warten.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Daten der Zielkomponente werden an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT die passende Komponente und wartet gegebenenfalls auf deren Erscheinen, oder das Verstreichen des Timeouts.

Attribute:

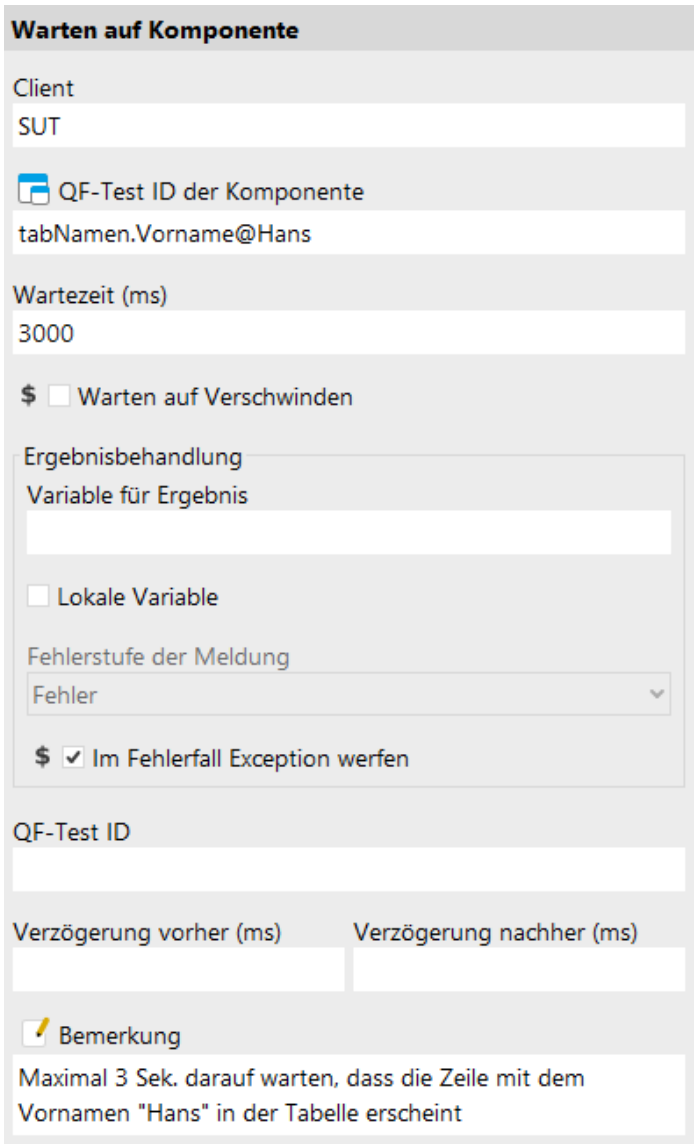


Abbildung 42.76: Warten auf Komponente Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, aus dem die Daten gelesen werden sollen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster[919], Komponente[930] oder Element[936] Knotens dessen Daten abgefragt werden.

Der "Komponente auswählen" Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5[47]) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf.

Variabel: Ja

Einschränkungen: >= 0

Warten auf Verschwinden

Ist dieses Attribut gesetzt, wird darauf gewartet, dass eine Komponente oder ein Unterelement verschwindet, also nicht sichtbar ist. Dies ist z.B. hilfreich um zu prüfen, dass ein Dialog geschlossen oder gar nicht erst geöffnet wurde. Bleibt die Komponente bzw. das Unterelement die ganze Zeit über sichtbar, wird eine `ComponentFoundException`[959] geworfen.

Variabel: Ja

Einschränkungen: Keine

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es

möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur[672], Abhängigkeit[630] oder Testfall[599] Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6[116].

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren[593] kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`[963] geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.7 Warten auf Laden des Dokuments

Web

Dieser Knoten ist eine besondere Variante des Warten auf Komponente(876) Knotens speziell für Webseiten. Er wartet nicht nur auf das Vorhandensein eines Dokuments, sondern, falls das Dokument beim Abspielen des letzten Events bereits geladen und bekannt war, darauf, dass dieses neu geladen wird. Da beim Nachladen von Webseiten oder Frames oft das selbe Dokument neu geladen wird, könnte QF-Test ohne diese Funktionalität nicht entscheiden, ob zu einem gegebenen Zeitpunkt noch das alte Dokument vorhanden ist oder schon ein neues geladen wurde. Im ersten Fall würde daraufhin ggf. ein Event im alten Dokument wiedergegeben, der keine Wirkung mehr erzielt, weil inzwischen das Laden der neuen Seite beginnt.

Über das Attribut Name des Browser-Fensters kann die Suche auf ein vorhandenes Browser-Fenster beschränkt, oder einem neuen Browser-Fenster ein Name zugewiesen werden. Ist Ladevorgang nach Ablauf der Wartezeit abbrechen gesetzt, bricht QF-Test nach Ablauf der Wartezeit das Laden des Dokuments ab. Das Resultat des Knotens kann auch in einer Variable mittels des Attributs Variable für Ergebnis gesetzt werden. Mit Ausschalten des Attributs Im Fehlerfall Exception werfen kann das Werfen einer `DocumentNotLoadedException`(959) unterdrückt werden.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Die Daten des Zieldokuments werden an das SUT geschickt. Die `TestEventQueue` ermittelt daraus im SUT das passende Dokument und wartet darauf, dass es komplett geladen wird, oder das Timeout verstreicht, woraufhin eine `DocumentNotLoadedException` geworfen wird.

Attribute:

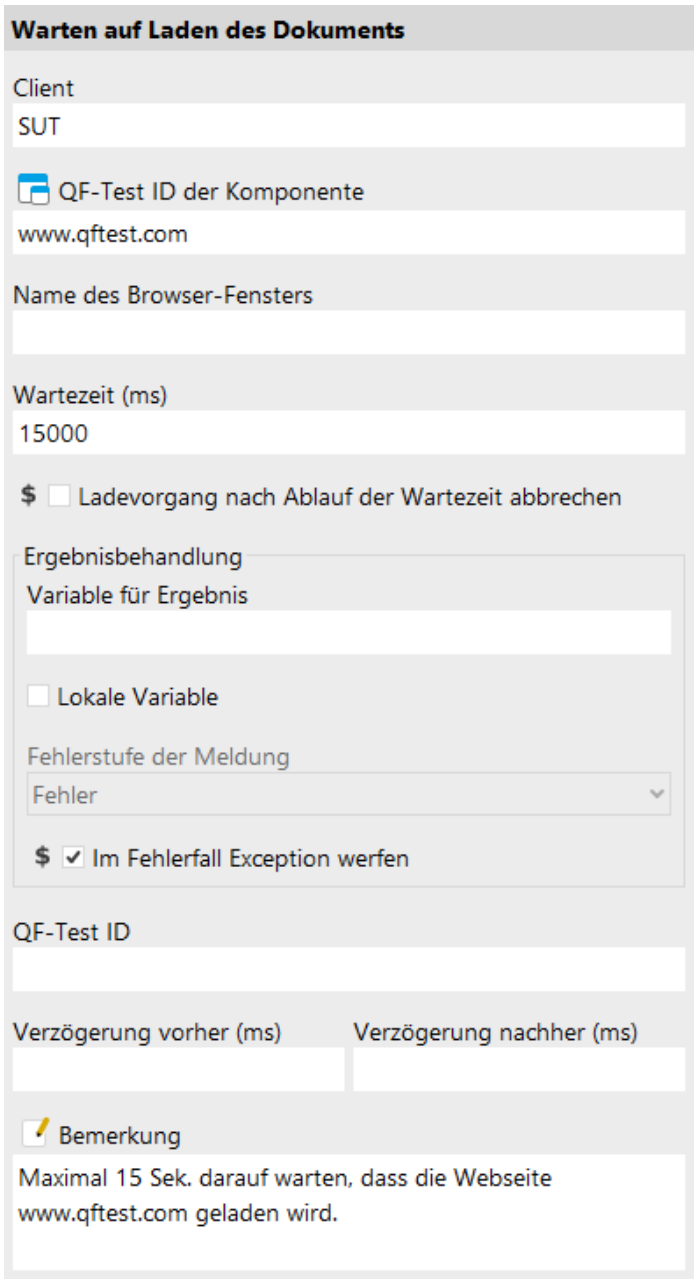


Abbildung 42.77: Warten auf Laden des Dokuments Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, aus dem die Daten gelesen werden sollen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID des Fenster[919], Komponente[930] oder Element[936] Knotens dessen Daten abgefragt werden.

Der "Komponente auswählen" Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5[47]) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

Name des Browser-Fensters

Dieses Attribut hat eine doppelte Funktion. Wird es auf den Namen eines existierenden Browser-Fensters gesetzt, wartet QF-Test auf das Laden des Dokuments nur in diesem Fenster. Ist der Name gesetzt und existiert kein zugehöriges Browser-Fenster, wird die Suche auf Dokumente in neuen oder noch nicht benannten Browser-Fenstern beschränkt. Wird dabei ein passendes Dokument gefunden, wird der Name dem zugehörigen Browser-Fenster zugewiesen. Dies ist die einzige Möglichkeit einem Popup-Fenster einen Namen zu geben. Für explizit gestartete Browser kann dieser mittels des Attributs Name des Browser-Fensters[765] im Browser-Fenster öffnen[763] Knoten definiert werden. Der Umgang mit mehreren Fenstern wird in FAQ 25 genauer erläutert.

Variabel: Ja

Einschränkungen: Keine

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf.

Variabel: Ja

Einschränkungen: >= 0

Ladevorgang nach Ablauf der Wartezeit abbrechen

Ist dieses Attribut gesetzt, wird nach erfolglosem Verstreichen der Wartezeit der Ladevorgang abgebrochen und zwar abhängig vom Attribut Name des Browser-Fensters in allen Browsern oder nur dem angegebenen. Ein solcher Abbruch hat keinen Einfluss auf das Ergebnis, nach Ablauf der Wartezeit gilt die Operation in jedem Fall als gescheitert.

Variabel: Ja

Einschränkungen: Keine

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur(672), Abhängigkeit(630) oder Testfall(599) Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6(116).

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception

werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`(963) geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.8 Warten auf Ende des Downloads

Web

Dieser spezielle Knoten ist nur auf Webseiten anwendbar. Er kann verwendet werden, um auf das Ende eines Downloads zu warten, der von QF-Test aus angestoßen wurde. Dies kann nötig sein, wenn Sie den Inhalt einer heruntergeladenen Datei verifizieren oder die Zeit für den Download messen wollen.

Ist der Download nach Ablauf der Wartezeit nicht beendet, wird eine DownloadNotCompleteException geworden, sofern diese nicht mittels des Attributs Im Fehlerfall Exception werfen unterdrückt wird. In jedem Fall kann der Download durch Aktivieren des Attributs Download nach Ablauf der Wartezeit abbrechen vorzeitig abgebrochen werden, um einen weiteren Download für die selbe Zieldatei zu ermöglichen. Das Ergebnis der Operation kann in einer Variable abgelegt werden indem deren Name über das Attribut Variable für Ergebnis festgelegt wird.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die Zieldatei für den Download wird zum SUT geschickt, wo QF-Test auf das Ende des Downloads wartet. Verstreicht die Wartezeit erfolglos, wird eine DownloadNotCompleteException geworfen.

Attribute:

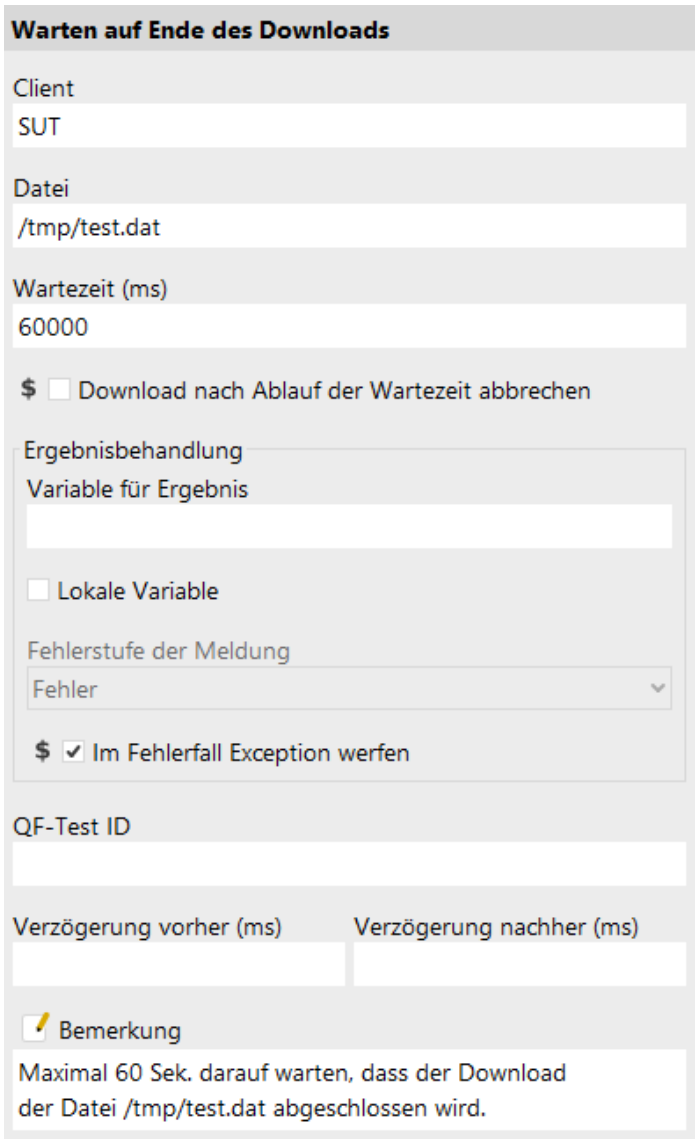


Abbildung 42.78: Warten auf Ende des Downloads Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, aus dem die Daten gelesen werden sollen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Datei

Die Zieldatei für den Download.

Variabel: Ja

Einschränkungen: Gültiger Dateiname

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf.

Variabel: Ja

Einschränkungen: >= 0

Download nach Ablauf der Wartezeit abbrechen

Ist dieses Attribut gesetzt, wird der Download nach erfolglosem Verstreichen der Wartezeit abgebrochen.

Variabel: Ja

Einschränkungen: Keine

Variable für Ergebnis

Mit diesem optionalen Attribut können Sie den Namen einer Variable festlegen, die abhängig vom Ergebnis der Aktion auf 'true' (erfolgreich) oder 'false' (fehlgeschlagen) gesetzt wird.

Hinweis

Ist dieses Attribut gesetzt, wird das Attribut Fehlerstufe der Meldung ignoriert. Das Attribut Im Fehlerfall Exception werfen behält dagegen seine Funktion, so dass es möglich ist, eine Ergebnisvariable zu erhalten und trotzdem eine Exception zu werfen.

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, wird die Variable in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur(672), Abhängigkeit(630) oder Testfall(599) Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6(116).

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Fehlerstufe der Meldung

Über dieses Attribut legen Sie die Fehlerstufe der Meldung fest, die in das Protokoll geschrieben wird, wenn die Aktion nicht erfolgreich ist. Zur Auswahl stehen Nachricht, Warnung und Fehler.

Hinweis

Dieses Attribut ist ohne Bedeutung, falls eines der Attribute Im Fehlerfall Exception werfen oder Variable für Ergebnis gesetzt ist.

Variabel: Nein

Einschränkungen: Keine

Im Fehlerfall Exception werfen

Ist dieses Attribut gesetzt, wird bei einem Scheitern der Aktion eine Exception geworfen. Für 'Check...'-Knoten wird eine `CheckFailedException`(963) geworfen, für 'Warten auf...'- Knoten eine spezifische Exception für diesen Knoten.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.9 Ressourcen laden

Hiermit laden Sie Daten aus einem `ResourceBundle` und stellen diese für die erweiterte Variablensyntax `${Gruppe:Name}` (vgl. Abschnitt 6.7[126]) zur Verfügung. Informationen über `ResourceBundles` finden Sie in der Beschreibung des ResourceBundle[891] Attributs.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Das `ResourceBundle` wird geladen und unter dem Gruppennamen[891] für spätere Zugriffe gebunden.

Attribute:

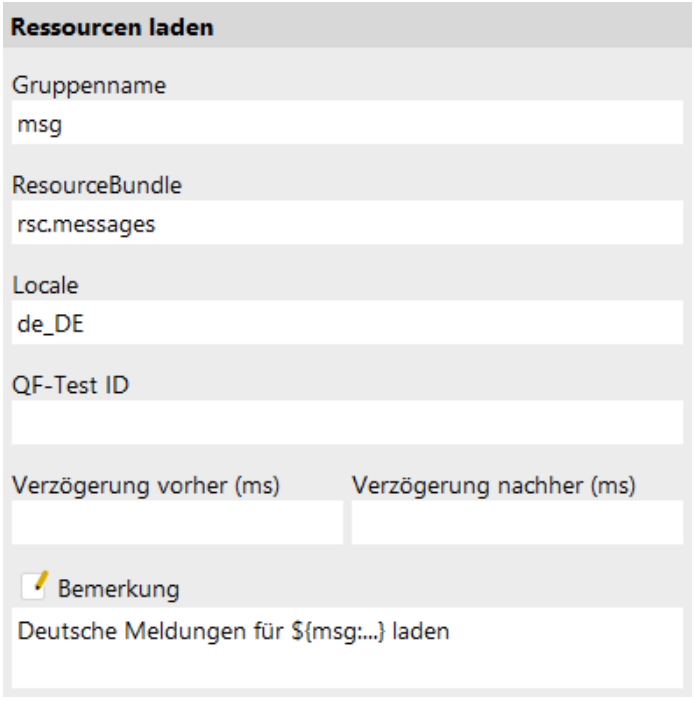


Abbildung 42.79: Ressourcen laden Attribute

Gruppenname

Der Gruppenname unter dem das `ResourceBundle` abgelegt wird. Der Wert einer Definition der Form `Name=Wert` aus diesem `ResourceBundle` kann dann mittels `${`*`Gruppenname`*`:`*`Name`*`}` abgerufen werden.

Variabel: Ja

Einschränkungen: Darf nicht leer sein und sollte keine Sonderzeichen enthalten.

ResourceBundle

Gibt das `ResourceBundle` an, das geladen werden soll. Hierfür ist etwas Java-Verständnis nötig, ggf. müssen Sie sich bei einem Entwickler erkundigen, wo die Ressourcen für das SUT abgelegt sind.

Die Ressourcen werden mit Hilfe der Java-Funktion `ResourceBundle.getBundle()` geladen. Dazu muss eine passende Datei mit der Endung `.class` oder `.properties` im Klassenpfad zu finden sein. Geben Sie in diesem Attribut den vollen Packagenamen mit '.' als Trennzeichen an, sowie den Basisnamen des Bundles ohne Locale und Endung.

Beispiel: QF-Test selbst enthält unter anderem ein deutsches `ResourceBundle` in der Datei

de/qfs/apps/qftest/resources/properties/qftest_de.properties, die im Archiv qfshared.jar enthalten ist. Um darauf zuzugreifen, müssten Sie hier den Wert de.qfs.apps.qftest.resources.properties.qftest angeben und das Locale(892) auf de setzen.

Variabel: Ja

Einschränkungen: Muss ein ResourceBundle im Java-Klassenpfad bezeichnen.

Locale

Das Hauptanwendungsgebiet von ResourceBundles ist es, Daten in verschiedenen Sprachen bereitzustellen. Hiermit legen Sie fest, welche Version des Bundles geladen werden soll. Dabei wird eine dem ISO Standard entsprechende Angabe der Form *Sprache_Land_Variante* erwartet. *Sprache* ist ein aus zwei Kleinbuchstaben bestehendes Kürzel, z.B. de für Deutsch oder en für Englisch. *Land* legt mit zwei Großbuchstaben die landesspezifische Ausprägung fest, z.B. en_UK für britisches und en_US für amerikanisches Englisch. Mit der *Variante* kann noch feiner unterschieden werden, was aber selten benötigt wird.

QF-Test stützt sich zum Laden des Bundles auf den in der Java-Dokumentation für ResourceBundle.getBundle() beschriebenen Mechanismus, der vereinfacht dargestellt wie folgt arbeitet:

Um ein ResourceBundle namens res für das Locale de_DE zu laden, durchsucht Java den Klassenpfad zunächst nach einer Datei namens res_de_DE.class oder res_de_DE.properties, dann nach res_de.class oder res_de.properties und schließlich nach res.class und res.properties. Dabei werden die weniger spezifischen Dateien auch dann geladen, wenn die spezifischeren bereits gefunden wurden. Aus diesen werden aber nur Werte übernommen, die nicht in der spezifischen Datei definiert wurden. Dadurch können Sie z.B. in res_de.properties alle deutschen Ressourcen ablegen. Wenn Sie einige davon für die Schweiz anders definieren wollen, erstellen Sie die Datei res_de_CH.properties und geben dort nur die abweichenden Definitionen an.

Java hat allerdings ein ”Feature”, das was zu einem überraschenden Ergebnis führen kann: Wenn keine Datei außer der Basisdatei res.properties gefunden wurde, sucht Java noch einmal nach spezifischen Dateien, diesmal für das Standard Locale, in dem die VM gerade läuft. Wenn Sie also mit dem deutschen QF-Test arbeiten und englische Ressourcen laden wollen, die direkt in res.properties ohne weiteres res_en.properties enthalten sind und eine deutsche Version in res_de.properties liegt, wird Java, selbst wenn Sie en als Locale angeben, die deutsche Version laden. Dies können Sie unterbinden,

indem Sie einen einzelnen Unterstrich '_' als Locale angeben. In diesem Fall wird definitiv nur `res.properties` geladen.

Wenn Sie dieses Attribut leer lassen, wird das Standard Locale verwendet in dem QF-Test gerade läuft.

Variabel: Ja

Einschränkungen: Leer oder gültige Locale Bezeichnung

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.10 Properties laden

Hiermit laden Sie Daten aus einer `Properties` Datei und stellen diese für die erweiterte Variablensyntax `${Gruppe:Name}` (vgl. Abschnitt 6.7[126]) zur Verfügung.

`Properties` sind einfacher zu handhaben als ein `ResourceBundle`, da Sie die Datei direkt angeben können, bieten dafür aber weniger Möglichkeiten. Das Format einer `Properties` Datei ist einfach: Zeilen der Form `Name=Wert` mit beliebigem Leerraum um das '=' Zeichen. Komplexe, auch mehrzeilige Definitionen sind möglich. Näheres entnehmen Sie bitte der Java-Dokumentation oder befragen Sie Ihre Entwickler.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die `Properties` werden geladen und unter dem Gruppennamen[894] für spätere Zugriffe gebunden.

Attribute:

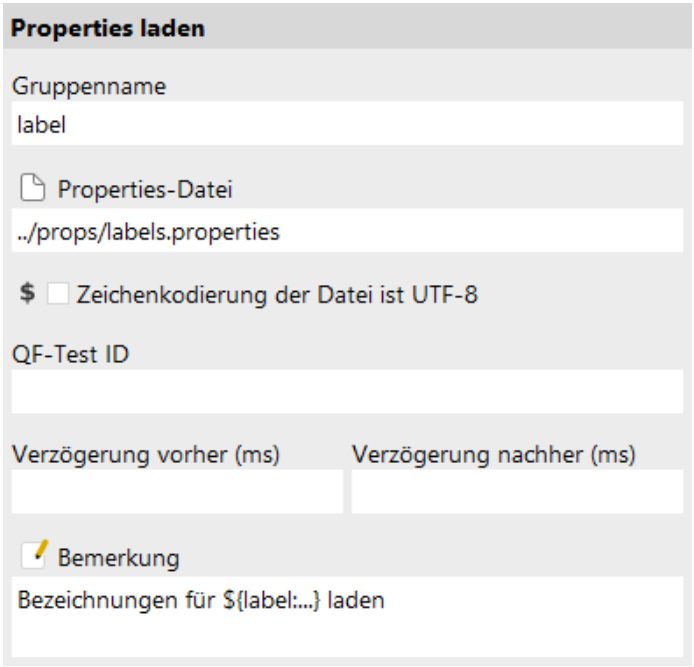


Abbildung 42.80: Properties laden Attribute

Gruppenname

Der Gruppenname unter dem die `Properties` abgelegt werden. Der Wert einer Definition der Form `Name=Wert` aus diesen `Properties` kann dann mittels `${Gruppenname:Name}` abgerufen werden.

Variabel: Ja

Einschränkungen: Darf nicht leer sein und sollte keine Sonderzeichen enthalten.

Properties-Datei

Legt die Datei fest, aus der die Properties geladen werden. Diese kann entweder mit absolutem Pfad angegeben werden, oder relativ zur aktuellen Suite. Als Trennzeichen für Verzeichnisse sollten Sie immer '/' angeben, QF-Test setzt es dann für das aktuelle Betriebssystem um.

Der "..." Button öffnet einen Dialog, in dem Sie die Datei direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet.

Variabel: Ja

Einschränkungen: Muss eine existierende Properties Datei bezeichnen.

Zeichenkodierung der Datei ist UTF-8

Bis Java 8 mussten Properties-Dateien für die Klasse `java.util.Properties` immer ISO-Latin-1 kodiert sein. Ab Java 9 ist die Standardkodierung hierfür UTF-8. QF-Test unterstützt beides und verwendet die UTF-8-Kodierung, falls dieses Attribut gesetzt ist, andernfalls ISO-Latin-1.

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung[551] aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.11 Unit-Test

Hiermit können JUnit-Tests ausgeführt werden.

`JUnit`-Tests sind für Komponenten und Integrationstests geeignet. Also für einfache Tests, welche oft wiederholt werden. Unit-Tests können entweder innerhalb eines SUT-Skripts definiert werden, mit dem SUT oder aus Java-Klassen geladen werden.

Enthalten in: Alle Arten von Sequenzen[599].

Kinder: Keine

Ausführung: Die benötigten Ressourcen und Injections werden geladen und vorbereitet. Dann werden nacheinander die einzelnen Test-Klassen ausgeführt.

Attribute:

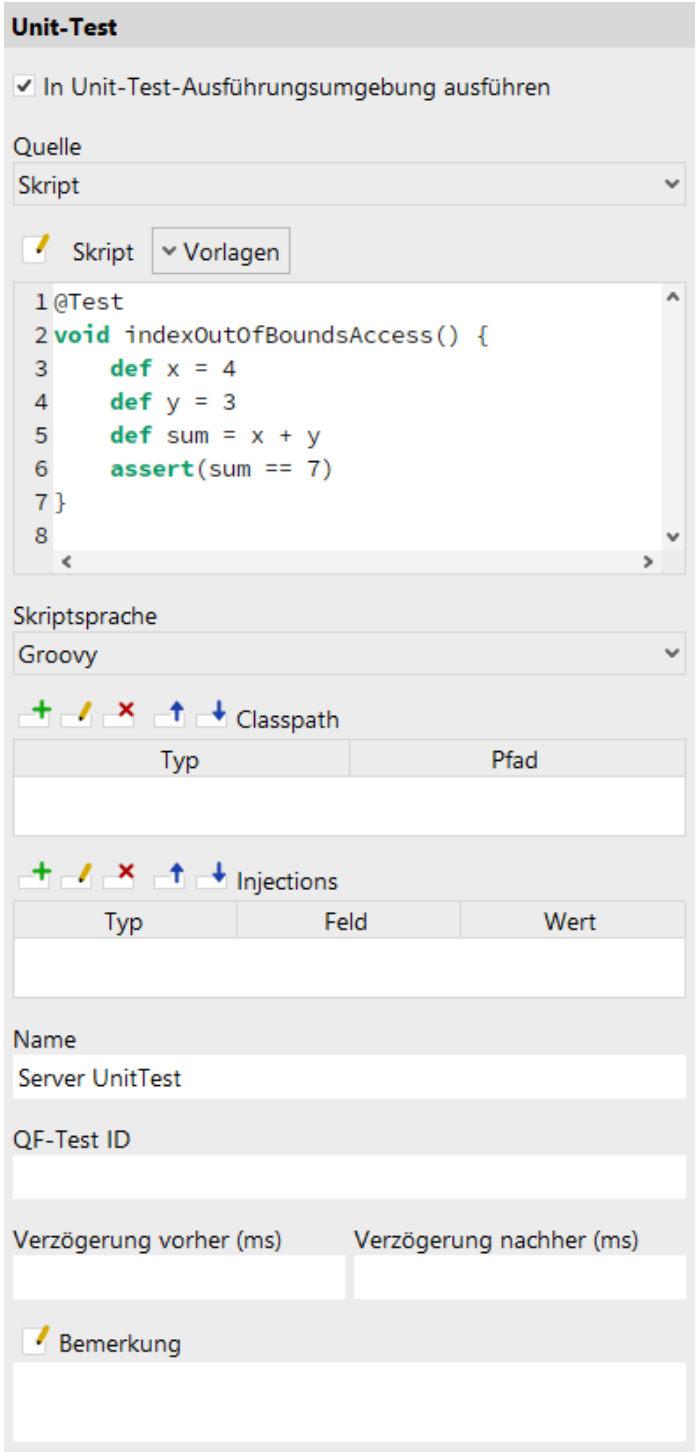


Abbildung 42.81: Unit-Test aus einem Skript ohne Verwendung eines Clients

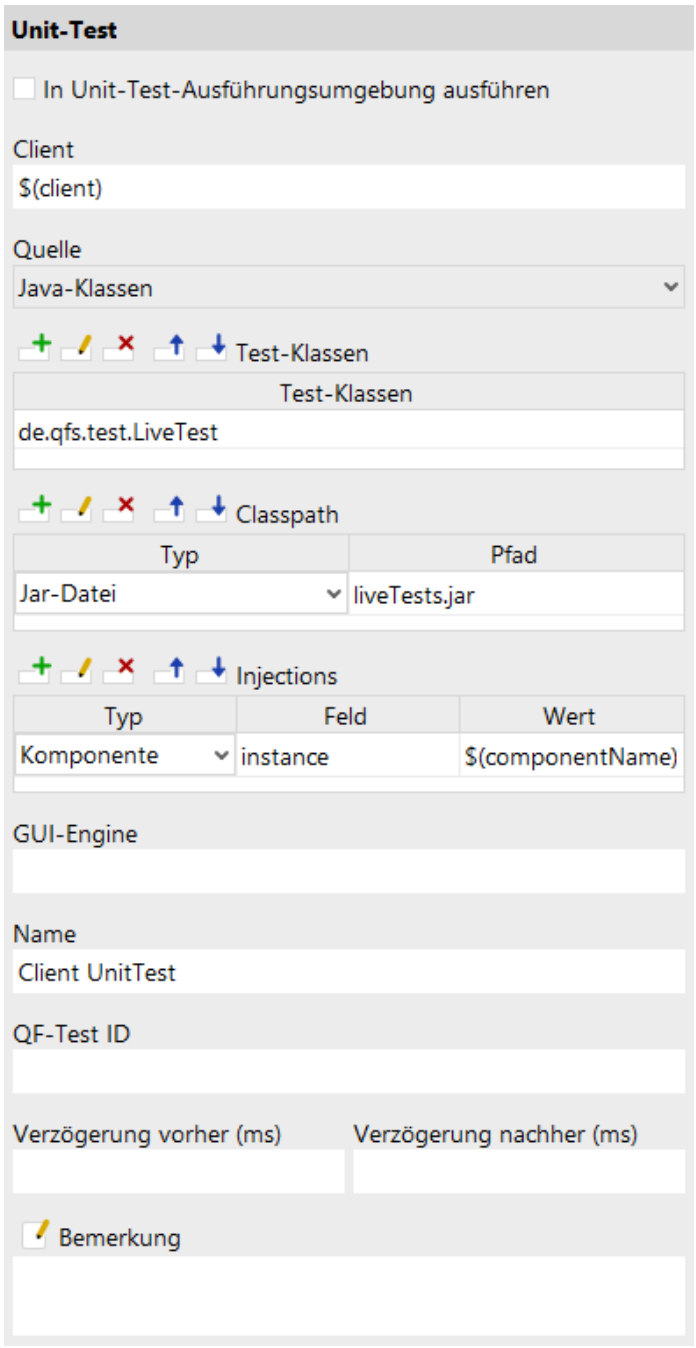


Abbildung 42.82: Unit-Test aus Java-Klassen mit Verwendung eines Clients

In Unit-Test-Ausführungsumgebung ausführen

Ob die Unit-Tests in einer eigenen Umgebung ausgeführt werden sollen. Falls diese Option deaktiviert ist, muss ein Client angegeben werden, in welchem die

Tests ausgeführt werden.

Variabel: Ja

Einschränkungen: Keine.

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, in dem das Skript ausgeführt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Quelle

Die Quelle aus der die JUnit-Tests ausgeführt werden können. Diese kann entweder aus einem SUT-Skript bestehen oder aus Java-Klassen, welche in das SUT geladen werden.

Skript

Das Skript, das ausgeführt werden soll.

Hinweis

In Jython-Skripten können QF-Test Variablen der Form $(var) oder ${Gruppe:Name} verwendet werden. Diese werden expandiert bevor das Skript an den Interpreter übergeben wird. Dies kann zu unerwünschten Effekten führen. Stattdessen sollte dafür die Methode `rc.getStr` verwendet werden, die in allen Skriptsprachen zur Verfügung steht (vgl. Abschnitt 11.3.3(192)).

Hinweis

Trotz Syntax-Highlighting und automatischer Einrückung ist dieses Textfeld womöglich nicht der geeignete Ort, um komplexe Skripte zu schreiben. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons das Skript komfortabel bearbeitet werden kann. Außerdem können komplexe Skripte in separate Module ausgelagert werden, die dann in QF-Test Skripte importiert werden. (vgl. Kapitel 50(1028)).

Variabel: Ja

Einschränkungen: Gültige Syntax

Vorlagen

Dieses Dropdown-Menü enthält eine Liste nützlicher Vorlagenskripte. Die verfügbaren Vorlagen unterscheiden sich je nach gewähltem Skripttyp und Skriptsprache.

Wenn Sie eine dieser Vorlagen auswählen, wird der aktuelle Inhalt Ihres Skripts ersetzt.

Sie können Ihre eigenen Vorlagen zu diesem Menü hinzufügen, indem Sie ”Benutzervorlagen-Verzeichnis öffnen” wählen und Ihre Vorlagendateien dort ablegen. Die folgenden Dateitypen sind gültig:

- **[Verzeichnis]:** Wird als Untermenü angezeigt.
- **.py:** Eine Jython-Vorlage.
- **.groovy:** Eine Groovy-Vorlage.
- **.js:** Eine JavaScript-Vorlage.

Skriptsprache

Dieses Attribut legt den Interpreter fest, in dem das Skript ausgeführt wird, oder in anderen Worten die Skriptsprache. Mögliche Werte sind ”Jython”, ”Groovy” und ”JavaScript”.

Variabel: Nein

Einschränkungen: Keine

Test-Klassen

Hier werden die Test-Klassen angegeben, welche über in den angegebenen Classpaths geladen wurden und ausgeführt werden sollen. Diese werden als Testschritte ausgeführt.

Anstatt vollständiger Klassennamen können auch Reguläre Ausdrücke angegeben werden.

Test-Klassen werden gefunden, wenn sie die JUnit 4 Test-Annotation besitzen, von der JUnit 3 Klasse `unit.org.TestCase` abgeleitet sind oder eine RunWith-Annotation besitzen.

Folgende Reguläre Ausdrücke sind möglich:

Regulärer Ausdruck	Bedeutung
**.MainTest	Alle MainTest-Klassen in allen Packages.
de.qfs.test.*	Alle Test-Klassen im package de.qfs.test.
de.qfs.**.*	Alle Test Klassen in allen Packages unter de.qfs.

Tabelle 42.33: Mögliche Reguläre Ausdrücke

Hinweis

Beim Suchen der Test-Klassen werden alle Klassen, die sich in den angegeben Verzeichnissen befinden, geladen. Der Ausdruck `**.*` lädt alle Klassen im Classpath und somit auch deren statische Felder. Daher sollte er nur mit Vorsicht verwendet werden.

Variabel: Ja

Einschränkungen: Die Klasse muss vorhanden sein.

Classpath

Hier können Dateien bzw. Ordner angegeben werden aus welchen die Test-Klassen geladen werden.

Variabel: Ja

Einschränkungen: Der angegebene Pfad muss gültig sein.

Injections

Mit den Injections können Objekte von QF-Test in die Unit-Tests übertragen werden, um darin mit ihnen zu arbeiten.

Typ	Beschreibung
String	QF-Test Variablen oder direkte Werte.
Komponente	Komponenten auf QF-Test.
WebDriver	WebDriver Objekte des aktuellen Browsers.

Tabelle 42.34: Arten von Injections

Hinweis

Es muss kein Wert für ”Feld” angegeben werden. In diesem Fall wird als Standardwert `instance` verwendet.

Variabel: Ja

Einschränkungen: Die Objekte müssen vorhanden sein.

GUI-Engine

Die GUI-Engine in der das Skript ausgeführt werden soll. Nur relevant für SUTs mit mehr als einer GUI-Engine wie in Kapitel 45(998) beschrieben.

Variabel: Ja

Einschränkungen: Siehe Kapitel 45(998)

Name

Der Name eines Unit-Tests ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion des Skripts aussagen.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.11.12 CustomWebResolver installieren

Mit diesem Knoten können Sie den CustomWebResolver konfigurieren.

Die Konfiguration des CustomWebResolver installieren Knotens ist in Der CustomWebResolver installieren Knoten(1082) im Detail beschrieben.

Enthalten in: Alle Arten von von Sequenzen(599).

Kinder: Abhängigkeit(630), SUT-Skripte(720), Prozeduraufruf(675) und Kommentar(851).

Ausführung: Der CustomWebResolver wird gemäß der angegebenen Konfiguration installiert oder aktualisiert.

Anschließend werden sämtliche enthaltenen Kindknoten nacheinander ausgeführt. Dabei werden innerhalb des CustomWebResolver die Variablen `$(client)` und `$(guiengine)` auf die Werte der Attribute Client und GUI-Engine gesetzt.

Falls der CustomWebResolver einen Vorbereitung-Knoten enthält, so wird dieser bereits *vor* der Anwendung der Konfiguration ausgeführt. Ein enthaltener Aufräumen-Knoten wird erst während der Deinstallation des CustomWebResolver ausgeführt.

Attribute:

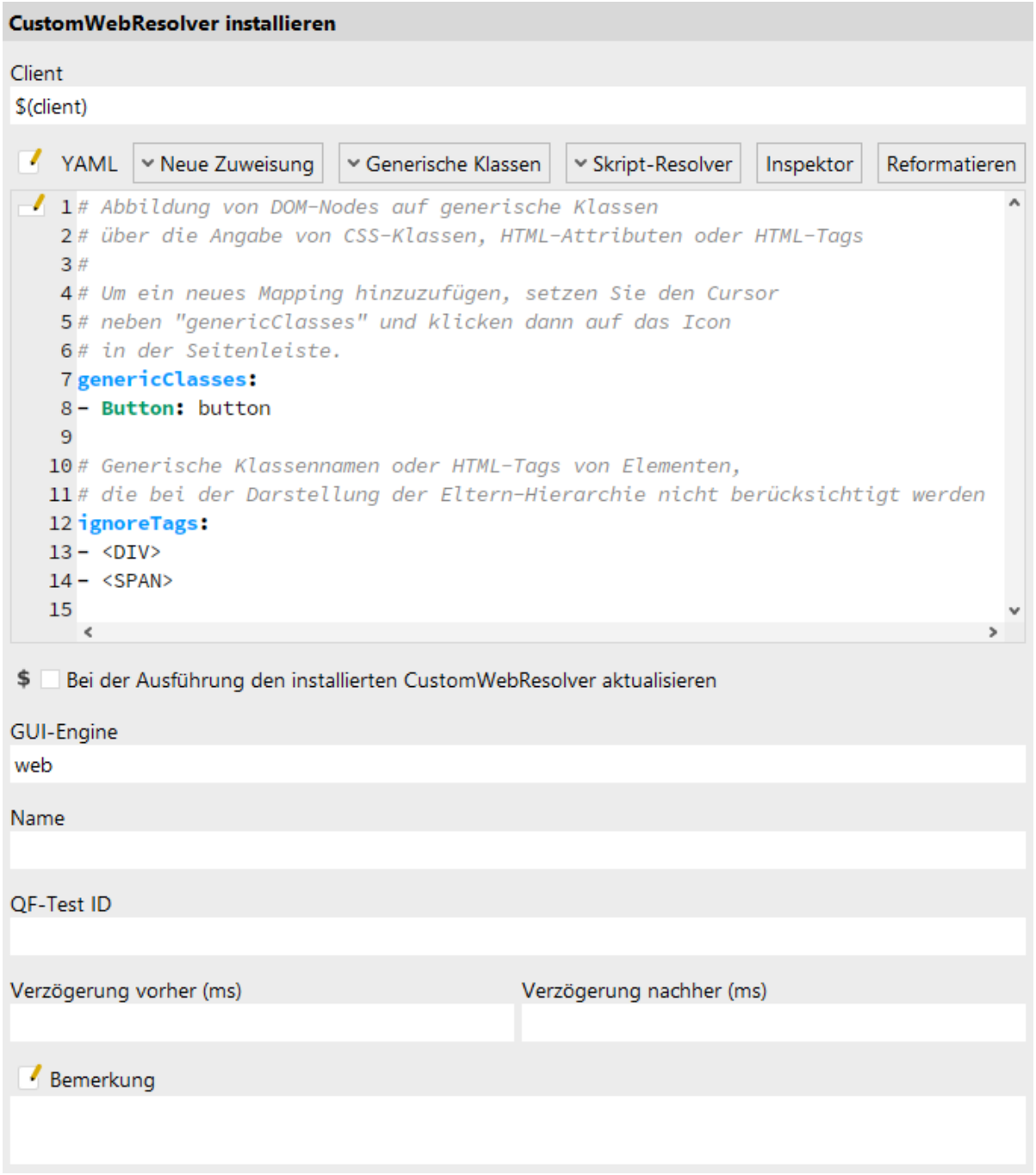


Abbildung 42.83: CustomWebResolver installieren Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, für das der CustomWebResolver gelten soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

YAML

Die Konfigurationsanweisungen für den CustomWebResolver. Diese sind in Abschnitt 51.1.2[1082] beschrieben. Es muss die dort beschriebene Syntax eingehalten werden.

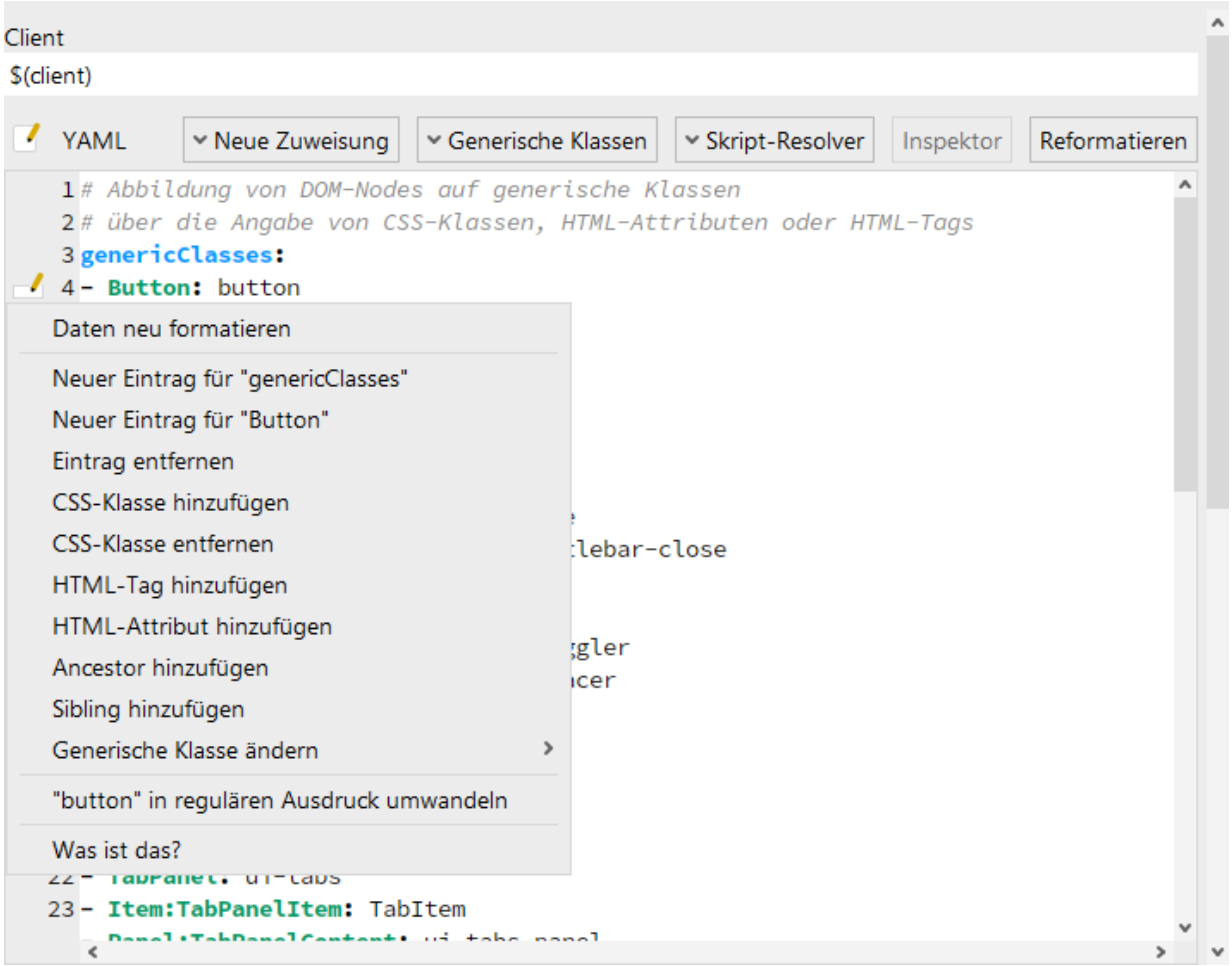


Abbildung 42.84: CustomWebResolver Konfigurationsvorlagen

Falls die Syntax bekannt ist, kann die YAML-Konfiguration direkt editiert werden. Bei einer ungültigen Konfiguration werden beim Ausführen oder Reformatieren entsprechende Fehlermeldungen ausgelöst.

Variabel: Ja

Einschränkungen: Gültige Syntax

Editier-Menü

Dieses Menü dient zur Vereinfachung der Bearbeitung der YAML-Konfiguration. Es bietet je nach Position im Dokument unterschiedliche Aktionen an.

Im Editor kann es auch jederzeit über Ctrl-Leertaste aktiviert werden.

Es können unter anderem die folgenden Aktionen zur Verfügung stehen:

Name	Beschreibung
Daten neu formatieren	Formatiert die angegebenen Daten in möglichst kompakter Form. Bei ungültiger Konfiguration wird stattdessen ein Dialog mit allen Konfigurationsfehlern angezeigt.
Neuer Eintrag für "..."	Legt eine neue Zuweisung für die Kategorie oder generische Klasse an.
HTML-Tag hinzufügen/entfernen	Steuert, ob die Zuweisung vom Namen des HTML-Tag des Elements abhängig ist.
CSS-Klasse hinzufügen/entfernen	Steuert, ob die Zuweisung von einer CSS-Klasse des Elements abhängig ist.
HTML-Attribut hinzufügen/entfernen	Steuert, ob die Zuweisung von einem HTML-Attribut des Elements abhängig ist.
Generische Klasse ändern	Steuert die generische Klasse, die dem Element zugewiesen wird.
Ancestor hinzufügen/entfernen	Steuert, ob die Zuweisung von einem Container des Elements abhängig ist.
"..." in Regex umwandeln/Regex entfernen	Steuert, ob der Wert als regulärer Ausdruck interpretiert wird.
Konfigurationsfehler anzeigen	Öffnet einen Dialog, der sämtliche in der aktuellen Konfiguration enthaltenen Probleme auflistet. Solange die Konfiguration fehlerhaft ist, kann der CustomWebResolver installieren nicht ausgeführt werden.

Tabelle 42.35: Aktionen des Editier-Menüs

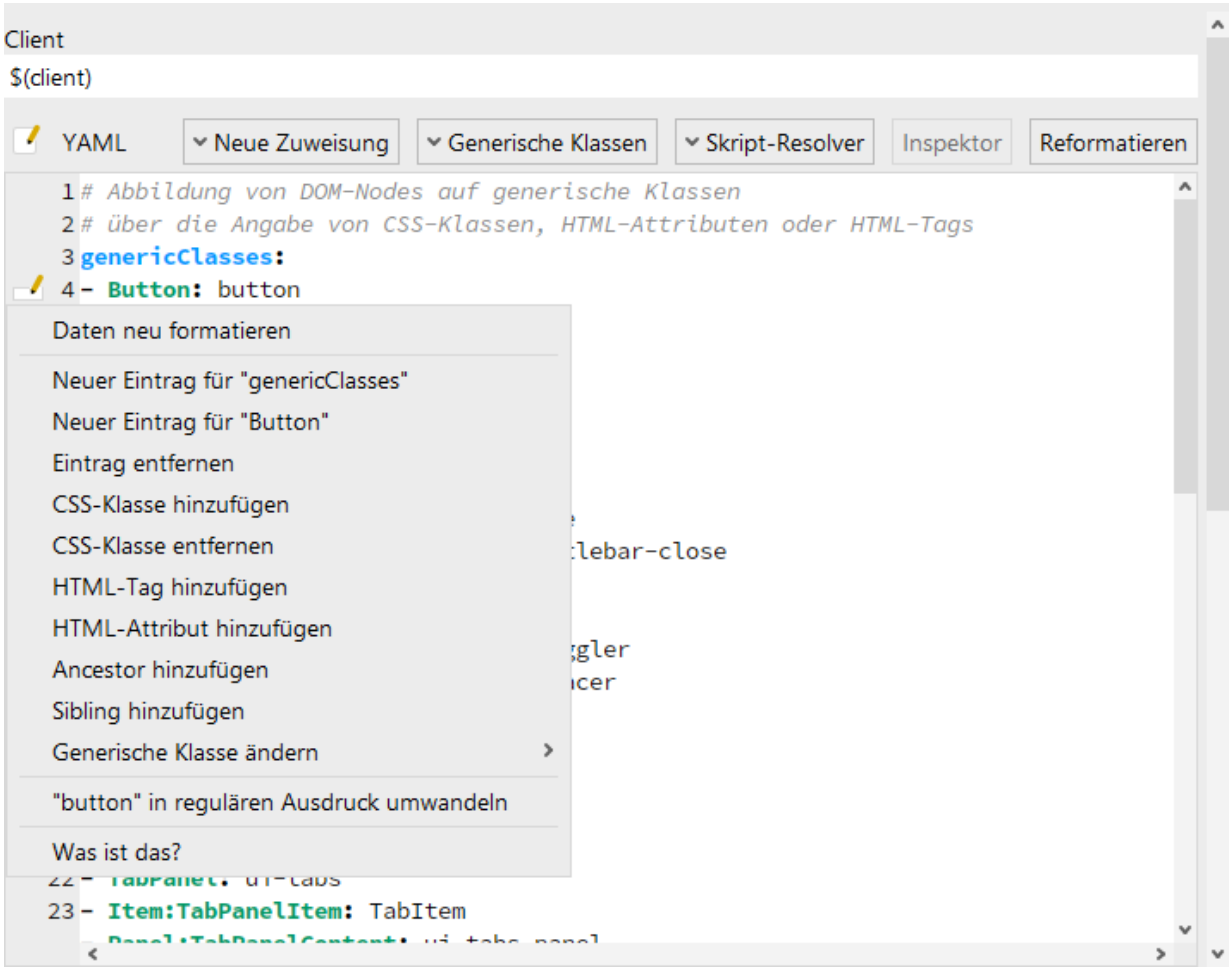


Abbildung 42.85: CustomWebResolver Editier-Menü

Weitere Informationen zu den Möglichkeiten der Konfigurations-Syntax finden Sie in Abschnitt 51.1.2(1082).

Neue Zuweisung

Beim Klick auf diesen Button öffnet sich eine Liste der verfügbaren Konfigurationskategorien. Wenn Sie einen Eintrag auswählen, wird in der passenden Kategorie ein Eintrag angelegt, bei dem Sie mögliche Platzhalter durch die gewünschten Werte ersetzen müssen.

Generische Klassen

Beim Klick auf diesen Button öffnet sich eine Liste, über die Sie eine neue Zuweisung für die jeweilige generische Klasse erzeugen können. Die Eigenschaften, die einer Klasse zugeordnet sind, können Sie in Generische Klassen(1329) nachlesen.

Skript-Resolver

Beim Klick auf diesen Button öffnet sich eine Liste über die Sie einen Template für einen der in Das `resolvers` Modul[1154] beschriebenen Resolver auswählen können. Das Template wird als eigener SUT-Skript[720] Knoten innerhalb des CustomWebResolver Knoten angelegt. Falls der Text-Cursor auf einer Zuweisung in `genericClasses` steht, wird der Resolver für die jeweilige generische Klasse registriert.

Inspektor

Beim Klick auf diese Schaltfläche öffnet sich der UI-Inspektor, siehe UI-Inspektor[108].

Sie sollten den UI-Inspektor verwenden, um Ihre Anwendung auf für den CustomWebResolver geeignete Merkmale zu durchsuchen und um die Wirkung des CustomWebResolver auf die Komponentenstruktur Ihrer Anwendung zu prüfen.

Die Schaltfläche ist nur verfügbar, wenn ein Web-Client aktiv ist.

Reformatieren

Beim Klick auf diesen Button wird der vorhandene YAML-Code gemäß der in Abschnitt 51.1.2[1082] beschriebenen Syntax in möglichst kompakter Form reformatiert. Hierdurch können auch Syntaxfehler erkannt werden.

Diese Aktion wird auch implizit jedes mal durchgeführt, wenn die Konfiguration z.B. über das Editier-Menü verändert wird.

Bei der Ausführung den installierten CustomWebResolver aktualisieren

Ist dieses Attribut gesetzt, so wird bei der Ausführung des Knotens der aktuell installierte CustomWebResolver nicht durch einen neuen ersetzt, sondern die Zuordnungen des installierten CustomWebResolver werden um die in diesem Knoten angegebenen Werte ergänzt. Ist kein CustomWebResolver installiert, so wird eine `TestException`[958] geworfen.

Variabel: Ja

Einschränkungen: Keine

GUI-Engine

Die GUI-Engine, in der der CustomWebResolver installiert oder aktualisiert werden soll. Nur relevant für SUTs mit mehr als einer GUI-Engine wie in Kapitel 45[998] beschrieben.

Variabel: Ja

Einschränkungen: Siehe Kapitel 45[998]

Name

Der Name ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Rolle des Knotens aussagen.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.12 HTTP-Requests

Dieser Abschnitt befasst sich mit der Möglichkeit HTTP-Requests mittels QF-Test abzusenden.

42.12.1 Server-HTTP-Request

Web

Dieser sehr spezielle Knoten bietet eine einfache Möglichkeit, einen Web-Request direkt via HTTP/HTTPS an einen Webserver abzusetzen. Die Verwendung ist bei Lasttests oder Szenarien mit Verarbeitung von Massendaten (z.B. Ausfüllen von Formularen) sehr hilfreich, da beim Abspielen die Simulation von Nutzereingaben im SUT und die damit verbundene Abhängigkeit von den Ladezeiten des SUT entfallen. Die Verwendung von Requests stellt somit eine Ergänzung der in Kapitel 33(437) und Abschnitt 42.4(646) beschriebenen Funktionalität für Lasttests und datengetriebenes Testen dar.

Wenn der vom Server zurückgegebene Statuscode 400 oder größer ist, wird eine Exception geworfen. Die Fehlerstufe kann mit dem Attribut Fehlerstufe bei HTTP-Statuscode >= 400 verändert werden. Detailinformationen zu den verschiedenen Statuscodes können sie unter http://www.w3.org/Protocols/HTTP/HTRESP.html nachlesen. Zusätzlich können Sie die Antwort des Servers einer Variablen zuweisen und über das Attribut Antwort des Servers in Protokoll schreiben diese Antwort auch im Protokoll ablegen.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Der Request wird via HTTP/HTTPS direkt von QF-Test aus an die angegebene URL geschickt. Wenn der vom Server zurückgegebene Statuscode größer/gleich 400 ist wird eine Exception geworfen. Dieses Verhalten kann über das Attribut Fehlerstufe bei HTTP-Statuscode >= 400 beeinflusst werden.

Attribute:

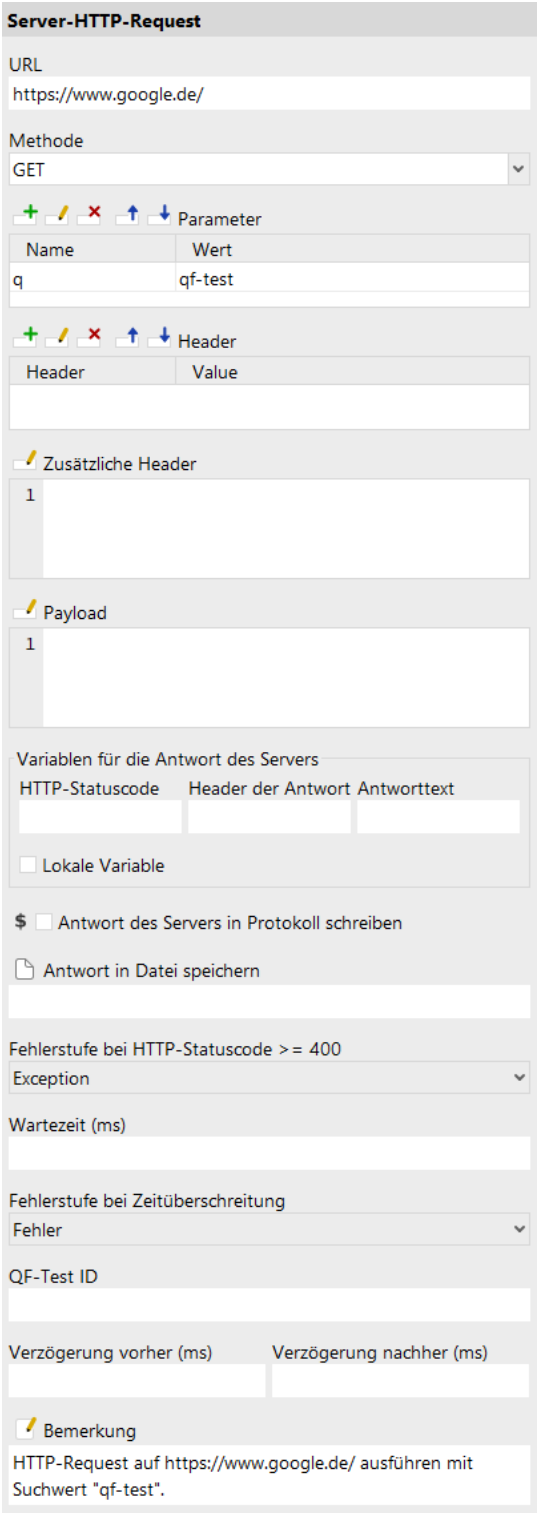


Abbildung 42.86: Server-HTTP-Request Attribute

URL

Die URL, an die der Request gesendet werden soll, exklusive der Parameter. Als Protokoll sind sowohl HTTP als auch HTTPS zulässig.

Internationalisierte Domainnamen (IDN) werden in der URL ebenso wie Pfade beginnend mit 'file:///' nicht unterstützt.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Methode

Über dieses Attribut legen Sie die Methode des Requests fest: GET, POST, OPTIONS, PUT, DELETE, HEAD oder TRACE.

Variabel: Ja

Einschränkungen: Keine

Parameters

Hier können Sie die Parameter für den Request definieren. Die Parameter werden bei der Ausführung kodiert übertragen (URL-Kodierung). Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20].

Variabel: Ja

Einschränkungen: Keine

Header

Mit dieser Tabelle können benutzerdefinierte Header spezifiziert werden. Dazu muss der Name des Header-Felds und dessen Wert angegeben werden. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20].

Variabel: Ja

Einschränkungen: Keine

Zusätzliche Header

Alternativ oder in Ergänzung zur Headers-Tabelle, können hier weitere Header als Text definiert werden. Damit ist es einfacher, Variablen zu nutzen und ggf. auch Header wegzulassen. Anzugeben ist jeweils ein Header pro Zeile im Format Header: Wert.

Variabel: Ja

Einschränkungen: Keine

Payload

Für die Methoden POST, PUT, DELETE bzw. HEAD, kann zusätzlich Payload in Form von XML, JSON oder Text angegeben

werden. Dafür muss der Content-Type Header an das entsprechende Format angepasst werden. Die Informationen dazu sind hier zu finden: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Type.

Variabel: Ja

Einschränkungen: Keine

Variable für den HTTP-Statuscode

Der Name der Variable welcher der HTTP-Statuscode als Wert zugewiesen wird (vgl. Kapitel 6[116]).

Variabel: Ja

Einschränkungen: Keine

Variable für den Header der Antwort

Der Name der Variable welcher die Header der Antwort als Wert zugewiesen werden (vgl. Kapitel 6[116]).

Variabel: Ja

Einschränkungen: Keine

Variable für den Antworttext

Der Name der Variable welcher der Body des Servers (HTTP-Response) als Wert zugewiesen wird (vgl. Kapitel 6[116]).

Variabel: Ja

Einschränkungen: Keine

Lokale Variable

Ist dieses Attribut nicht gesetzt, werden die Variablen in den globalen Definitionen gebunden. Andernfalls wird - sofern vorhanden - jeweils die oberste aktuelle Definition der Variablen überschrieben, sofern diese innerhalb des aktuellen Prozedur[672], Abhängigkeit[630] oder Testfall[599] Knotens liegt. Gibt es keine solche Definition, wird eine neue Definition im aktuellen Prozedur, Abhängigkeit oder Testfall Knoten angelegt, oder, falls kein solcher existiert, im obersten Knoten auf dem Variablen-Stapel mit Fallback auf die globalen Definitionen. Eine Erläuterung dieser Begriffe und weitere Details zu Variablen finden Sie in Kapitel 6[116].

Über die Option Attribut 'Lokale Variable' standardmäßig aktivieren[593] kann der Wert voreingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Antwort des Servers in Protokoll schreiben

Ist dieses Attribut gesetzt, wird zusätzlich zum Status-Code die Antwort des Servers (HTTP-Response) in das Protokoll geschrieben.

Variabel: Ja

Einschränkungen: Keine

Antwort in Datei speichern

Hier kann eine Datei angegeben werden, in welche die Antwort des Servers geschrieben wird. Dadurch können Dateien heruntergeladen werden.

Variabel: Ja

Einschränkungen: QF-Test muss in die Datei schreiben können.

Fehlerstufe bei HTTP-Statuscode >= 400

Hier kann die Fehlerstufe von HTTP-Statuscodes, die größer/gleich 400 sind, eingestellt werden.

Variabel: Nein

Einschränkungen: Keine

Wartezeit

Zeit in Millisekunden, die maximal verstreichen darf, bis der HTTP-Request erfolgreich durchgeführt wurde. Lassen Sie das Attribut leer wenn Sie unbegrenzt auf die Durchführung warten wollen.

Variabel: Ja

Einschränkungen: Darf nicht negativ sein.

Fehlerstufe bei Zeitüberschreitung

Dieses Attribut legt fest, was bei Überschreitung des Zeitlimits passiert. Ist der Wert ”Exception”, wird eine `CheckFailedException`(963) geworfen. Andernfalls wird eine Meldung mit der entsprechenden Fehlerstufe in das Protokoll geschrieben.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen ’\’, ’#’, ’$’, ’@’, ’&’, oder ’%’ enthalten und nicht mit einem Unterstrich (’_’) beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.12.2 Browser-HTTP-Request

Web

Dieser sehr spezielle Knoten bietet eine einfache Möglichkeit, einen GET oder POST Request direkt via HTTP/HTTPS an einen Webserver abzusetzen. Die Verwendung ist bei Lasttests oder Szenarien mit Verarbeitung von Massendaten (z.B. Ausfüllen von Formularen) sehr hilfreich, da beim Abspielen die Simulation von Nutzereingaben im SUT und die damit verbundene Abhängigkeit von den Ladezeiten des SUT entfallen. Die Verwendung von Requests stellt somit eine Ergänzung der in Kapitel 33(437) und Abschnitt 42.4(646) beschriebenen Funktionalität für Lasttests und datengetriebenes Testen dar.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Keine

Ausführung: Der GET/POST Request wird via HTTP/HTTPS direkt vom Browser aus an die angegebene URL geschickt. Die Ergebnisse des Requests werden im Browser angezeigt.

Attribute:

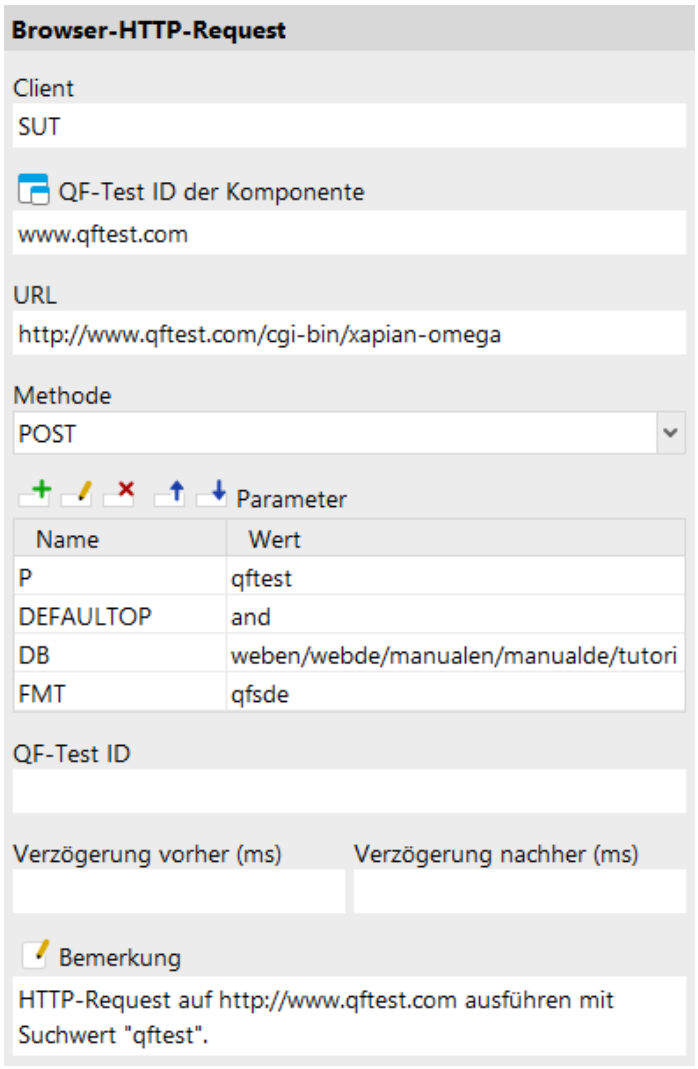


Abbildung 42.87: Browser-HTTP-Request Attribute

Client

Der Name unter dem der Java-Prozess des SUT gestartet wurde, in dem der Request ausgeführt werden soll.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

QF-Test ID der Komponente

Die QF-Test ID der Webseite[925] in welcher der Request abgespielt werden soll.

Der "Komponente auswählen" Button  öffnet einen Dialog, in dem Sie die Komponente (siehe Kapitel 5[47]) direkt auswählen können. Diesen erreichen Sie auch mittels Shift-Return oder Alt-Return, sofern sich der Fokus im Textfeld befindet. Alternativ können Sie den gewünschten Knoten mittels Strg-C bzw. Bearbeiten→Kopieren kopieren und seine QF-Test ID durch drücken von Strg-V in das Textfeld einfügen.

Dieses Attribut unterstützt ein spezielles Format, das es erlaubt, Komponenten in anderen Testsuiten zu referenzieren (siehe Abschnitt 26.1[359]). Des weiteren können Unterelemente von Knoten direkt angegeben werden, ohne dass ein eigener Knoten dafür vorhanden sein muss (siehe Abschnitt 5.9[92]). Bei der Verwendung von SmartIDs können Sie ein GUI-Element direkt über seine Wiedererkennungsmerkmale adressieren. Weitere Informationen hierzu finden Sie in SmartID[81] und Komponente-Knoten versus SmartID[51].

Variabel: Ja

Einschränkungen: Darf nicht leer sein

URL

Die URL, an die der Request gesendet werden soll, exklusive der Parameter. Als Protokoll sind sowohl HTTP als auch HTTPS zulässig.

Internationalisierte Domainnamen (IDN) werden in der URL ebenso wie Pfade beginnend mit 'file:///' nicht unterstützt.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Methode

Über dieses Attribut legen Sie die Methode des Requests fest, GET oder POST.

Variabel: Ja

Einschränkungen: Keine

Programm-Parameter

Hier können Sie die Parameter für den Request definieren. Die Parameter werden bei der Ausführung kodiert Übertragung (URL-Kodierung). Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20].

Variabel: Ja

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.13 Fenster, Komponenten und Elemente

Fenster(919), Komponenten(930) und Elemente(936) sind das Fundament, auf dem eine Testsuite aufbaut. Sie bilden mit ihrer Struktur und ihren Attributen die Fenster des SUT, sowie die darin enthaltenen Komponenten ab.

Andere Elemente der Suite, insbesondere Events(775) und Checks(805), müssen sich immer auf eine Komponente oder ein Fenster beziehen. Diese Beziehung wird über einen Ver-

weis auf die QF-Test ID des Fensters oder der Komponente hergestellt. Näheres dazu finden Sie bei der Beschreibung der entsprechenden Knoten.

Es gibt einen eigenen Bereich innerhalb einer Suite, der ausschließlich für Fenster, Komponenten und deren Unterelemente vorgesehen ist: Den Knoten Fenster und Komponenten(942), der seinen festen Platz am Ende einer Suite hat.

Weitere Informationen zur Arbeit mit Fenstern und Komponenten finden Sie in den Kapiteln Komponenten(47) sowie im "Best Practices" Unterkapitel Wie erreicht man eine robuste Komponentenerkennung?(54).

42.13.1 Fenster

Dieses Element repräsentiert ein Fenster im SUT. Events(775) und Checks(805) beziehen sich über seine QF-Test ID(920) darauf.

Enthalten in: Fenstergruppen(939), Fenster und Komponenten(942).

Kinder: Komponentengruppe(940), Komponente(930).

Ausführung: Kann nicht ausgeführt werden.

Attribute:

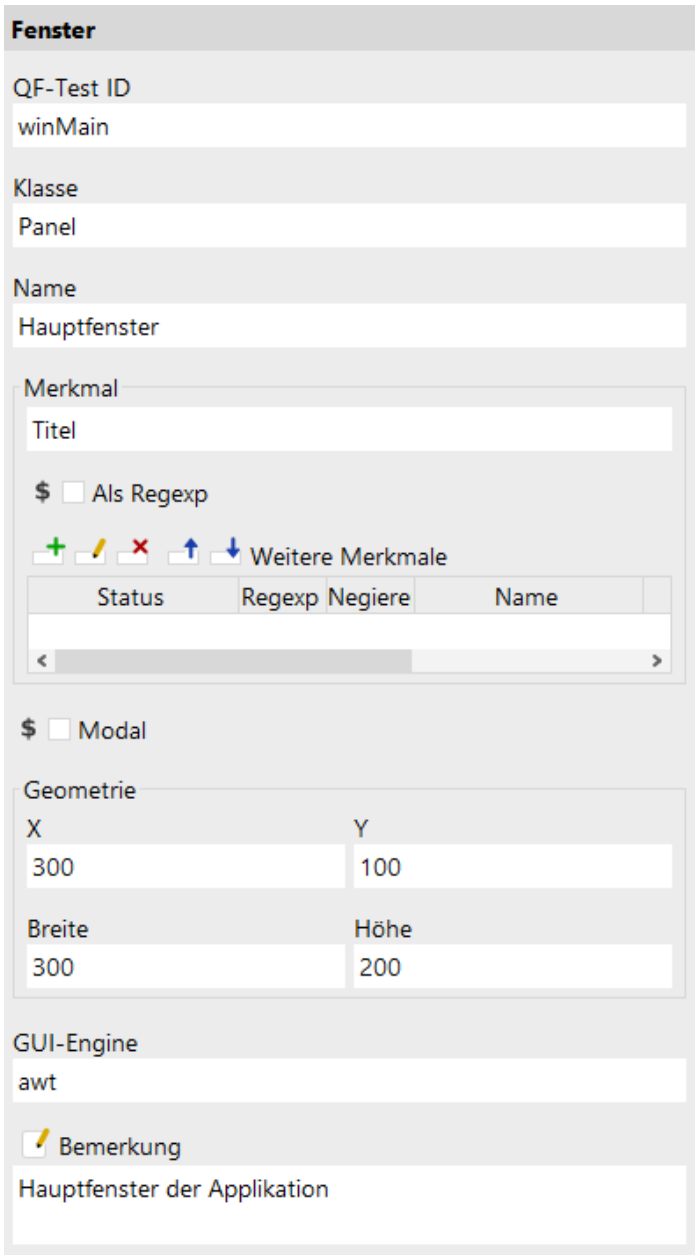


Abbildung 42.88: Fenster-Attribute

QF-Test ID

Mit Hilfe der QF-Test ID kann das Fenster von anderen Elementen referenziert werden, z.B. von einem Fensterevent(787), der sich auf dieses Fenster bezieht. Es ist daher wichtig, Fenstern und Komponenten möglichst prägnante QF-Test IDs zu geben, an die Sie sich gut erinnern können. Außerdem dürfen Sie jede QF-Test ID pro Suite nur einmal vergeben.

Variabel: Nein

Einschränkungen: Darf nicht leer sein, keines der Zeichen '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Klasse

Der vollständige Name der Java-Klasse des Fensters, oder eine ihrer Superklassen.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Name

Der Name des Fensters, der von den Entwicklern des SUT mittels `setName` vergeben werden sollte. An Hand dieses Namens kann QF-Test das Fenster eindeutig identifizieren.

Variabel: Ja

Einschränkungen: Keine

Merkmal

Hat ein Fenster keinen Namen, versucht QF-Test es an Hand eines typischen Merkmals zu identifizieren. Für *Frames* und *Dialoge* ermittelt QF-Test hierzu den Titel.

4.0+ Mittels Rechtsklick und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke in diesem Text mit '\' schützen.

Variabel: Ja

Einschränkungen: Keine

Als Regexp

Ist dieses Attribut gesetzt, handelt es sich beim Merkmal um einen regulären Ausdruck (vgl. Abschnitt 49.3(1023)).

Variabel: Ja

Einschränkungen: Keine

Weitere Merkmale

Neben dem primären Merkmal kann eine Komponente zusätzliche Merkmale in der Form von Name/Wert Paaren haben. Welche Merkmale aufgezeichnet werden, hängt von der Art der Komponente ab. Ein solches Merkmal kann einen von drei Zuständen haben:

Ignorieren (Hinweis für die Suche: aktuell dargestellt als 0)
Dieses zusätzliche Merkmal dient lediglich zur Information. Es hat keinen Einfluss auf die Wiedererkennung so lange dieser Status nicht manuell geändert wird.

Sollte übereinstimmen (Hinweis für die Suche: aktuell dargestellt als 1)
Zielkomponenten, die mit diesem Merkmal übereinstimmen haben bei der Wiedererkennung eine höhere Chance als solche, die nicht dazu passen.

Muss übereinstimmen (Hinweis für die Suche: aktuell dargestellt als 2)
Die Zielkomponenten muss mit diesem Feature übereinstimmen. Eine Komponente die nicht dazu passt, kommt als Kandidat für die Wiedererkennung nicht in Frage.

Weitere Spalten erlauben den Vergleich mit Hilfe eines regulären Ausdrucks sowie die Negation der Aussage. So lässt sich z.B. definieren, dass das ”class” Attribut eines DOM Knoten nicht ”dummy” sein darf. Das Fehlen eines Merkmals kann durch Vergleich mit einem leeren Wert ausgedrückt werden. Sie können in diesen Spalten übrigens auch Variablen verwenden, in dem Sie per Doppelklick auf diese Spalten einen Texteditor öffnen und die entsprechenden Variablen eintragen.

4.0+ Mittels Rechtsklick in die Wert Spalte und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke im Wert mit ’\’ schützen.

QF-Test fügt automatisch einige Einträge zu den Weiteren Merkmalen hinzu.

Name	Engine	Bedeutung
columns	Web	Anzahl der Spalten TABLE Komponenten
imagehash	Swing, SWT	Zeigt den Hashwert eines Icons eines Buttons oder MenuItems
qfs:class	Alle	Die konkrete Klasse der Komponente, z.B. de.qfs.QfsTextField.
qfs:genericclass	Alle	Die generische Klasse der Komponente, z.B. TextField.
qfs:item	Web	Zeigt den Elementindex eines DomNode, wenn dieser ein Element einer komplexen GUI-Komponente ist. Dies kann passieren, wenn Kindkomponenten aufgezeichnet werden, die als ebenfalls relevant betrachtet werden. Solche Elemente können in Bäumen, Listen, Reitern und Tabellen vorkommen.
qfs:label*-Varianten	Alle	Bis QF-Test Version 6: qfs:label enthält die beste Beschriftung einer Komponente. Dies kann ein Text eines Buttons oder der Text der am nächsten gelegenen Label Komponente sein. Wenn weder ein eigener Text noch ein Label gefunden werden kann, wird versucht Tooltips bzw. Icon Beschreibungen auszulesen. Ab QF-Test Version 7.0: Ein oder mehrere Einträge für Beschriftungen, die für die betrachtete Komponente in Frage kommen - die am besten bewertete mit dem Status ”Sollte übereinstimmen”, die anderen mit ”Ignorieren”. Zum Beispiel `qfs:labelText` für den Text der Komponente selbst oder`qfs:labelLeft` für eine Beschriftung links der Komponente. Weitere Informationen finden Sie unter `qfs:label*`-Varianten(74).
qfs:matchindex	Alle	Index von Komponenten mit demselben Namen. Wird ggf. automatisch zugewiesen, wenn die Option Wiedererkennung von Komponenten bei der Aufnahme validieren(519) aktiviert ist.
qfs:modal	Web	Zeigt an, ob die Komponente der Klasse ”Window” ein modales Fenster ist.
qfs:originalid	Web	Zeigt die originale ID des Knotens an, so wie diese wirklich im Dom steht.
qfs:systemclass	Alle	Die toolkit-spezifische Systemklasse, z.B. javax.swing.JTextField.
qfs:text	Alle	Enthält den Text der Komponente. Wird standardmäßig nicht aufgenommen, kann aber in SmartIDs oder durch manuelles Anlegen im Komponente Knoten für die Wiedergabe beliebig verwendet werden.
qfs:type	Alle	Der generische Typ der Komponente, z.B. TextField:PasswordField.

Tabelle 42.36: Weitere Merkmale, die von QF-Test gesetzt werden

Weitere Informationen finden Sie im Kapitel Weitere Merkmale(73).

Variabel: Ja

Einschränkungen: Name darf nicht leer sein

Modal

Gibt an, ob es sich bei dem Fenster um einen modalen Dialog handelt.

Variabel: Ja

Einschränkungen: Keine

Geometrie

Die X/Y Koordinaten, Breite und Höhe des Fensters bilden die Ausgangsbasis für die Wiedererkennung(1015), spielen dabei aber eine untergeordnete Rolle.

Für Fenster, deren Position oder Größe stark variiert, sollten Sie die entsprechenden Felder löschen.

Hinweis Sind keine Werte angegeben, startet der Mechanismus zur Wiedererkennung mit perfekter Übereinstimmung der Geometrie für alle Kandidaten. Um falsch-positive Treffer auf Basis der Geometrie zu verhindern, können Sie die Erkennung auf Basis der Geometrie unterdrücken, indem sie für diese Werte ein '-'-Zeichen eintragen.

Variabel: Ja

Einschränkungen: Breite und Höhe dürfen nicht negativ sein.

GUI-Engine

Die GUI-Engine zu der das Fenster und alle seine Komponenten gehören. QF-Test zeichnet `awt` für AWT/Swing und `swt` für SWT auf. Nur relevant für SUTs mit mehr als einer GUI-Engine wie in Kapitel 45(998) beschrieben.

Variabel: Ja

Einschränkungen: Siehe Kapitel 45(998)

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.13.2 Webseite

Web

Eine Webseite ist eine Variante des Fenster[919] Knotens speziell für den Test von Web-Anwendungen. Sie repräsentiert das oberste Dokument in einem Browser. Verschachtelte Dokumente innerhalb von FRAME Knoten werden als Komponenten[930] abgebildet.

Im Gegensatz zu einem Fenster hat eine Webseite keine Klasse, Modal, Geometrie oder GUI-Engine-Attribute, da diese entweder implizit definiert oder redundant sind.

Enthalten in: Fenstergruppe[939], Fenster und Komponenten[942].

Kinder: Komponentengruppe[940], Komponente[930].

Ausführung: Kann nicht ausgeführt werden.

Attribute:

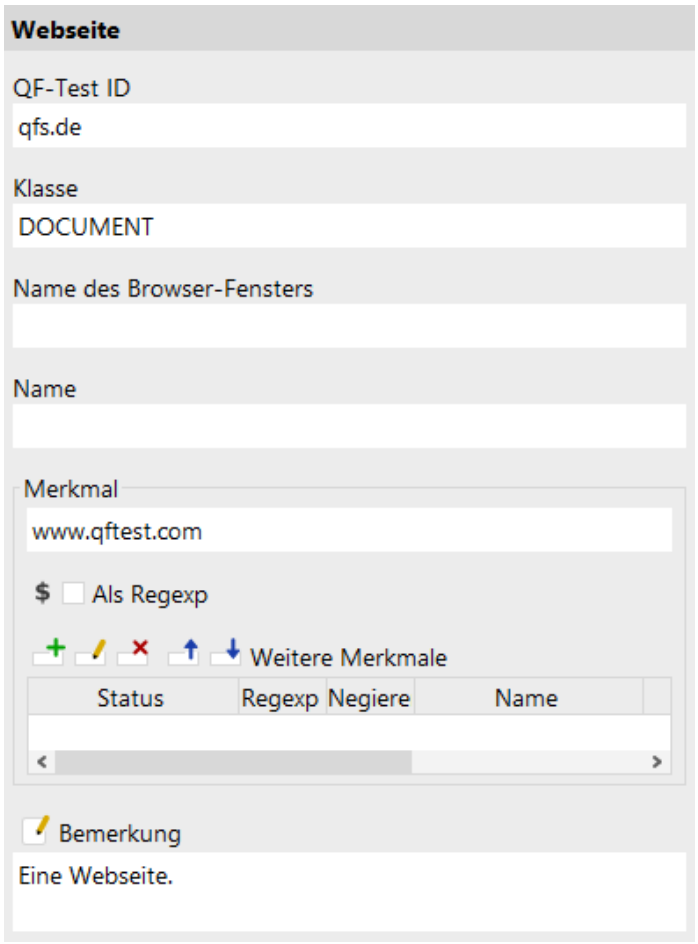


Abbildung 42.89: Webseite-Attribute

QF-Test ID

Mit Hilfe der QF-Test ID kann diese Seite von anderen Elementen referenziert werden, z.B. von einem Fensterevent(787), der sich auf diese Seite bezieht. Es ist daher wichtig, Fenstern und Komponenten möglichst prägnante QF-Test IDs zu geben, an die Sie sich gut erinnern können. Außerdem dürfen Sie jede QF-Test ID pro Suite nur einmal vergeben.

Variabel: Nein

Einschränkungen: Darf nicht leer sein, keines der Zeichen '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Name des Browser-Fensters

Dieses Attribut können Sie ignorieren, sofern Sie nicht eine Web-Anwendung mit mehreren offenen Browser-Fenstern testen müssen, die das selbe Dokument

darstellen. In diesem Fall kann das Attribut Name des Browser-Fensters dazu dienen, die Browser-Fenster zu unterscheiden. Der Name eines Browser-Fensters kann mittels des Attributs Name des Browser-Fensters[765] eines Browser-Fenster öffnen[763] Knotens festgelegt werden. Der Umgang mit mehreren Fenstern wird in FAQ 25 genauer erläutert.

Variabel: Ja

Einschränkungen: Keine

Name

Eine Webseite hat nur dann einen Namen, wenn dieser mittels eines NameResolvers implementiert wird. Näheres zum API für NameResolver finden Sie in Abschnitt 54.1.7[1163].

Variabel: Ja

Einschränkungen: Keine

Merkmal

Das primäre Merkmal einer Webseite ist ihre URL, wobei Parameter entfernt werden. Falls die Option URL-Merkmal von 'Webseite'-Knoten auf Host bzw. Datei beschränken[568] gesetzt ist, wird die URL sogar auf den Rechner- oder Dateinamen reduziert.

4.0+ Mittels Rechtsklick und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke in diesem Text mit '\' schützen.

Variabel: Ja

Einschränkungen: Keine

Als Regexp

Ist dieses Attribut gesetzt, handelt es sich beim Merkmal um einen regulären Ausdruck (vgl. Abschnitt 49.3[1023]).

Variabel: Ja

Einschränkungen: Keine

Weitere Merkmale

Neben dem primären Merkmal kann eine Komponente zusätzliche Merkmale in der Form von Name/Wert Paaren haben. Welche Merkmale aufgezeichnet werden, hängt von der Art der Komponente ab. Ein solches Merkmal kann einen von drei Zuständen haben:

Ignorieren (Hinweis für die Suche: aktuell dargestellt als 0)
Dieses zusätzliche Merkmal dient lediglich zur Information. Es hat keinen

Einfluss auf die Wiedererkennung so lange dieser Status nicht manuell geändert wird.

Sollte übereinstimmen (Hinweis für die Suche: aktuell dargestellt als 1)
Zielkomponenten, die mit diesem Merkmal übereinstimmen haben bei der Wiedererkennung eine höhere Chance als solche, die nicht dazu passen.

Muss übereinstimmen (Hinweis für die Suche: aktuell dargestellt als 2)
Die Zielkomponenten muss mit diesem Feature übereinstimmen. Eine Komponente die nicht dazu passt, kommt als Kandidat für die Wiedererkennung nicht in Frage.

Weitere Spalten erlauben den Vergleich mit Hilfe eines regulären Ausdrucks sowie die Negation der Aussage. So lässt sich z.B. definieren, dass das ”class” Attribut eines DOM Knoten nicht ”dummy” sein darf. Das Fehlen eines Merkmals kann durch Vergleich mit einem leeren Wert ausgedrückt werden. Sie können in diesen Spalten übrigens auch Variablen verwenden, in dem Sie per Doppelklick auf diese Spalten einen Texteditor öffnen und die entsprechenden Variablen eintragen.

4.0+ Mittels Rechtsklick in die Wert Spalte und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke im Wert mit ’\’ schützen.

QF-Test fügt automatisch einige Einträge zu den Weiteren Merkmalen hinzu.

Name	Engine	Bedeutung
columns	Web	Anzahl der Spalten TABLE Komponenten
imagehash	Swing, SWT	Zeigt den Hashwert eines Icons eines Buttons oder MenuItems
qfs:class	Alle	Die konkrete Klasse der Komponente, z.B. de.qfs.QfsTextField.
qfs:genericclass	Alle	Die generische Klasse der Komponente, z.B. TextField.
qfs:item	Web	Zeigt den Elementindex eines DomNode, wenn dieser ein Element einer komplexen GUI-Komponente ist. Dies kann passieren, wenn Kindkomponenten aufgezeichnet werden, die als ebenfalls relevant betrachtet werden. Solche Elemente können in Bäumen, Listen, Reitern und Tabellen vorkommen.
qfs:label*-Varianten	Alle	Bis QF-Test Version 6: qfs:label enthält die beste Beschriftung einer Komponente. Dies kann ein Text eines Buttons oder der Text der am nächsten gelegenen Label Komponente sein. Wenn weder ein eigener Text noch ein Label gefunden werden kann, wird versucht Tooltips bzw. Icon Beschreibungen auszulesen. Ab QF-Test Version 7.0: Ein oder mehrere Einträge für Beschriftungen, die für die betrachtete Komponente in Frage kommen - die am besten bewertete mit dem Status ”Sollte übereinstimmen”, die anderen mit ”Ignorieren”. Zum Beispiel `qfs:labelText` für den Text der Komponente selbst oder`qfs:labelLeft` für eine Beschriftung links der Komponente. Weitere Informationen finden Sie unter `qfs:label*`-Varianten(74).
qfs:matchindex	Alle	Index von Komponenten mit demselben Namen. Wird ggf. automatisch zugewiesen, wenn die Option Wiedererkennung von Komponenten bei der Aufnahme validieren(519) aktiviert ist.
qfs:modal	Web	Zeigt an, ob die Komponente der Klasse ”Window” ein modales Fenster ist.
qfs:originalid	Web	Zeigt die originale ID des Knotens an, so wie diese wirklich im Dom steht.
qfs:systemclass	Alle	Die toolkit-spezifische Systemklasse, z.B. javax.swing.JTextField.
qfs:text	Alle	Enthält den Text der Komponente. Wird standardmäßig nicht aufgenommen, kann aber in SmartIDs oder durch manuelles Anlegen im Komponente Knoten für die Wiedergabe beliebig verwendet werden.
qfs:type	Alle	Der generische Typ der Komponente, z.B. TextField:PasswordField.

Tabelle 42.37: Weitere Merkmale, die von QF-Test gesetzt werden

Weitere Informationen finden Sie im Kapitel Weitere Merkmale(73).

Variabel: Ja

Einschränkungen: Name darf nicht leer sein

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.13.3 Komponente

Komponenten entsprechen den *Components* der Java Oberfläche. Ebenso wie Fenster(919) werden sie von Events(775) und Checks(805) über ihre QF-Test ID(931) referenziert. Allgemeine Informationen zu Komponenten finden Sie im Kapitel Komponenten(47).

Komponenten werden innerhalb von QF-Test hierarchisch angeordnet, entsprechend den Parent-Child Beziehungen im SUT. Dabei werden allerdings nicht alle Zwischenkomponenten aufgezeichnet, sondern nur die ”interessanten”.

Enthalten in: Komponentengruppe(940), Fenster(919), Komponente.

Kinder: Komponentengruppe(940), Komponente, Element(936).

Ausführung: Kann nicht ausgeführt werden.

Attribute:

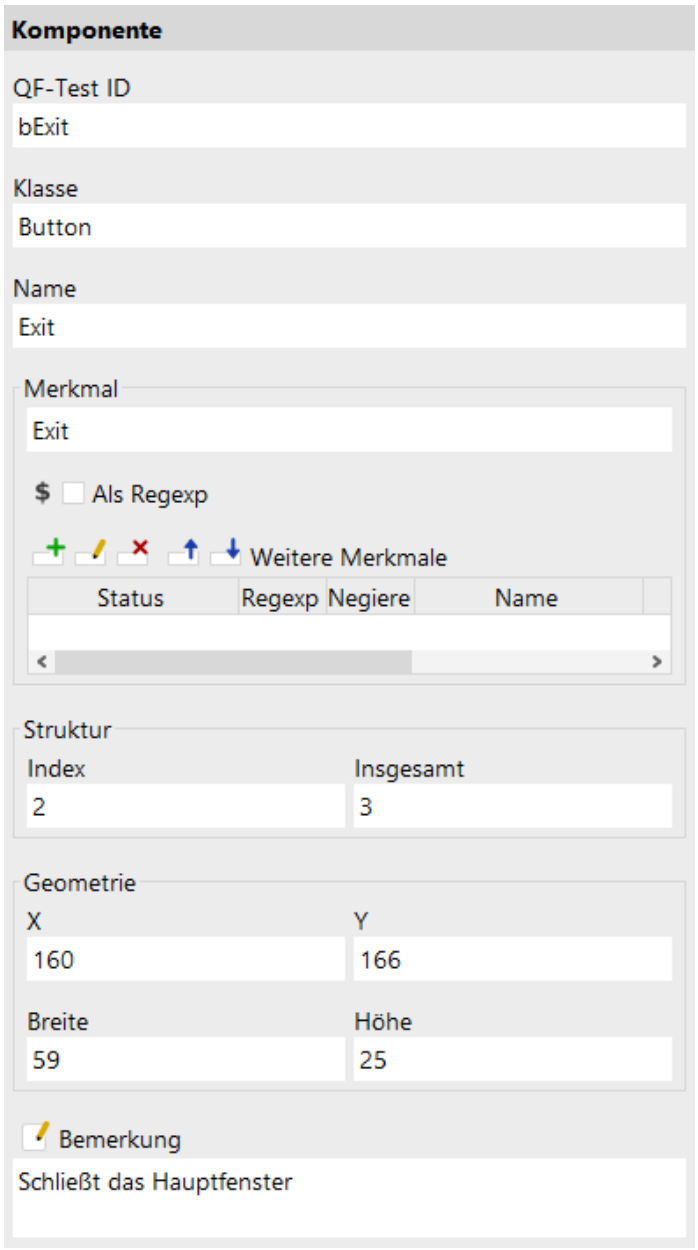


Abbildung 42.90: Komponente-Attribute

QF-Test ID

Mit Hilfe der QF-Test ID kann die Komponente von anderen Elementen referenziert werden, z.B. von einem Mausevent(775), der sich auf dieses Komponente bezieht. Es ist daher wichtig, Fenstern und Komponenten möglichst prägnante QF-Test IDs zu geben, an die Sie sich gut erinnern können. Außerdem

dürfen Sie jede QF-Test ID pro Suite nur einmal vergeben. Siehe auch Generierung der QF-Test ID der Komponente(1017).

Variabel: Nein

Einschränkungen: Darf nicht leer sein, keines der Zeichen '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Klasse

Für SWT und Swing-Anwendungen ist dies der vollständige Name der Java-Klasse der Komponente oder eine ihrer Superklassen. Bei Web-Anwendungen gibt es hingegen eine Pseudoklassenhierarchie, die in Abschnitt 54.10(1253) beschrieben ist. Allgemeine Informationen finden Sie im Kapitel Klasse(62).

Die Klasse, die von QF-Test für eine Komponente aufgezeichnet wird, hängt von der Option Nur Systemklassen aufnehmen(519) ab sowie von eventuell registrierten `ClassNameResolvern` (vgl. Abschnitt 54.1.9(1166)). Bei der Wiedergabe wird die Übereinstimmung der Klasse basierend auf der tatsächlichen bzw. der Pseudoklassenhierarchie ermittelt, so dass Sie dieses Attribut auch manuell auf eine Basisklasse des Elements setzen können.

Variabel: Ja

Einschränkungen: Darf nicht leer sein.

Name

Der Name der Komponente, der von den Entwicklern des SUT mittels `setName` vergeben werden sollte. An Hand dieses Namens kann QF-Test das Fenster eindeutig identifizieren. Allgemeine Informationen finden Sie im Kapitel Name(64).

Variabel: Ja

Einschränkungen: Keine

Merkmal

Hat eine Komponente keinen Namen, versucht QF-Test sie an Hand eines typischen Merkmals zu identifizieren. Für *Buttons*, *MenuItems* und *Labels* ermittelt QF-Test z.B. deren Text.

Allgemeine Informationen finden Sie im Kapitel Merkmal(70).

4.0+ Mittels Rechtsklick und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke in diesem Text mit '\' schützen.

Variabel: Ja

Einschränkungen: Keine

Als Regexp

Ist dieses Attribut gesetzt, handelt es sich beim Merkmal um einen regulären Ausdruck (vgl. Abschnitt 49.3(1023)).

Variabel: Ja

Einschränkungen: Keine

Weitere Merkmale

Neben dem primären Merkmal kann eine Komponente zusätzliche Merkmale in der Form von Name/Wert Paaren haben. Welche Merkmale aufgezeichnet werden, hängt von der Art der Komponente ab. Ein solches Merkmal kann einen von drei Zuständen haben:

Ignorieren (Hinweis für die Suche: aktuell dargestellt als 0)
Dieses zusätzliche Merkmal dient lediglich zur Information. Es hat keinen Einfluss auf die Wiedererkennung so lange dieser Status nicht manuell geändert wird.

Sollte übereinstimmen (Hinweis für die Suche: aktuell dargestellt als 1)
Zielkomponenten, die mit diesem Merkmal übereinstimmen haben bei der Wiedererkennung eine höhere Chance als solche, die nicht dazu passen.

Muss übereinstimmen (Hinweis für die Suche: aktuell dargestellt als 2)
Die Zielkomponenten muss mit diesem Feature übereinstimmen. Eine Komponente die nicht dazu passt, kommt als Kandidat für die Wiedererkennung nicht in Frage.

Weitere Spalten erlauben den Vergleich mit Hilfe eines regulären Ausdrucks sowie die Negation der Aussage. So lässt sich z.B. definieren, dass das ”class” Attribut eines DOM Knoten nicht ”dummy” sein darf. Das Fehlen eines Merkmals kann durch Vergleich mit einem leeren Wert ausgedrückt werden. Sie können in diesen Spalten übrigens auch Variablen verwenden, in dem Sie per Doppelklick auf diese Spalten einen Texteditor öffnen und die entsprechenden Variablen eintragen.

4.0+ Mittels Rechtsklick in die Wert Spalte und Auswahl von Text in regulären Ausdruck konvertieren, können Sie Sonderzeichen regulärer Ausdrücke im Wert mit ’\’ schützen.

QF-Test fügt automatisch einige Einträge zu den Weiteren Merkmalen hinzu.

Name	Engine	Bedeutung
columns	Web	Anzahl der Spalten TABLE Komponenten
imagehash	Swing, SWT	Zeigt den Hashwert eines Icons eines Buttons oder MenuItems
qfs:class	Alle	Die konkrete Klasse der Komponente, z.B. de.qfs.QfsTextField.
qfs:genericclass	Alle	Die generische Klasse der Komponente, z.B. TextField.
qfs:item	Web	Zeigt den Elementindex eines DomNode, wenn dieser ein Element einer komplexen GUI-Komponente ist. Dies kann passieren, wenn Kindkomponenten aufgezeichnet werden, die als ebenfalls relevant betrachtet werden. Solche Elemente können in Bäumen, Listen, Reitern und Tabellen vorkommen.
qfs:label*-Varianten	Alle	Bis QF-Test Version 6: qfs:label enthält die beste Beschriftung einer Komponente. Dies kann ein Text eines Buttons oder der Text der am nächsten gelegenen Label Komponente sein. Wenn weder ein eigener Text noch ein Label gefunden werden kann, wird versucht Tooltips bzw. Icon Beschreibungen auszulesen. Ab QF-Test Version 7.0: Ein oder mehrere Einträge für Beschriftungen, die für die betrachtete Komponente in Frage kommen - die am besten bewertete mit dem Status ”Sollte übereinstimmen”, die anderen mit ”Ignorieren”. Zum Beispiel `qfs:labelText` für den Text der Komponente selbst oder`qfs:labelLeft` für eine Beschriftung links der Komponente. Weitere Informationen finden Sie unter `qfs:label*`-Varianten(74).
qfs:matchindex	Alle	Index von Komponenten mit demselben Namen. Wird ggf. automatisch zugewiesen, wenn die Option Wiedererkennung von Komponenten bei der Aufnahme validieren(519) aktiviert ist.
qfs:modal	Web	Zeigt an, ob die Komponente der Klasse ”Window” ein modales Fenster ist.
qfs:originalid	Web	Zeigt die originale ID des Knotens an, so wie diese wirklich im Dom steht.
qfs:systemclass	Alle	Die toolkit-spezifische Systemklasse, z.B. javax.swing.JTextField.
qfs:text	Alle	Enthält den Text der Komponente. Wird standardmäßig nicht aufgenommen, kann aber in SmartIDs oder durch manuelles Anlegen im Komponente Knoten für die Wiedergabe beliebig verwendet werden.
qfs:type	Alle	Der generische Typ der Komponente, z.B. TextField:PasswordField.

Tabelle 42.38: Weitere Merkmale, die von QF-Test gesetzt werden

Weitere Informationen finden Sie im Kapitel Weitere Merkmale(73).

Variabel: Ja

Einschränkungen: Name darf nicht leer sein

Struktur

Zur Wiedererkennung(1015) notiert QF-Test charakteristische strukturelle Merkmale der Komponenten. Die Anzahl der Komponenten im SUT die die selbe Klasse haben und dem selben Parent angehören, wird im Attribut Insgesamt eingetragen. Die Position der jeweiligen Komponente bei dieser Zählung landet im Attribut Index. Die erste Komponente erhält den Index 0. Unsichtbare Komponenten werden hierbei mitgezählt.

Hinweis

Es kann auch nur eines der beiden Attribute angegeben werden, das andere wird dann ignoriert.

Allgemeine Informationen finden Sie im Kapitel Index(77).

Variabel: Ja

Einschränkungen: Nicht negativ.

Geometrie

Die X/Y Koordinaten, Breite und Höhe des Fensters bilden die Ausgangsbasis für die Wiedererkennung(1015), spielen dabei aber eine untergeordnete Rolle. Für Komponenten, deren Position oder Größe zur Laufzeit typischerweise stark variieren, werden die entsprechenden Werte nicht aufgezeichnet.

Hinweis

Sind keine Werte angegeben, startet der Mechanismus zur Wiedererkennung mit perfekter Übereinstimmung der Geometrie für alle Kandidaten. Um falsch-positive Treffer auf Basis der Geometrie zu verhindern, können Sie die Erkennung auf Basis der Geometrie unterdrücken, indem sie für diese Werte ein '-'-Zeichen eintragen.

Allgemeine Informationen finden Sie im Abschnitt Geometrie(77).

Variabel: Ja

Einschränkungen: Breite und Höhe dürfen nicht negativ sein.

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.13.4 Element

Mausevents(775) und Checks(805) können sich bei einer komplexen Swing Komponente auch auf ein Unterelement beziehen. Dieses Unterelement wird über einen Index, den Primärindex(937) identifiziert. Dieser kann auf drei Arten festgelegt werden: als Text, als Zahl oder als Regexp. Text und Regexp definieren dabei ein Unterelement mit entsprechender Darstellung, während eine numerische Interpretation des Index ein Unterelement an Hand seiner Position bestimmt. Wie in Java üblich, hat das erste Unterelement die Position 0.

Für die Klasse `JTable` kann neben dem Primärindex, der die Tabellenspalte festlegt, auch der Sekundärindex(938) zur Bestimmung der Zeile angegeben werden.

Für die Knoten von `JTree` Komponenten gibt es zwei Darstellungen: Flach wie in einer Liste oder hierarchisch mittels Pfadangaben. Ein Knoten names `tmp` unterhalb eines Knotens namens `usr` würde im besten Fall einfach als `tmp` dargestellt, im zweiten Fall als `/usr/tmp`. Welche Form bei der Aufnahme verwendet wird, entscheidet die Option Knoten im Baum als Pfad darstellen(525).

Für folgende komplexe Swing Komponenten sind Elemente definiert:

Klasse	Primärindex	Sekundärindex
`JComboBox`	Listenelement	-
`JEditorPane`	Strukturelement (experimentell)	-
`JList`	Listenelement	-
`JTabbedPane`	Tab	-
`JTable`	Tabellenspalte	Zeile
`JTableHeader`	Tabellenspalte	-
`JTextArea`	Zeile	-
`JTree`	Knoten	-

Tabelle 42.39: Unterelemente komplexer Swing Komponenten

Enthalten in: Komponente(930).

Kinder: Keine.

Ausführung: Kann nicht ausgeführt werden.

Attribute:

Abbildung 42.91: Element-Attribute

QF-Test ID

Mit Hilfe der QF-Test ID kann ein Element z.B. von einem Mausevent(775) referenziert werden, der sich auf dieses Element bezieht. Es ist daher wichtig, Elementen ebenso wie Fenstern und Komponenten möglichst prägnante QF-Test IDs zu geben, an die Sie sich gut erinnern können. Außerdem dürfen Sie jede QF-Test ID pro Suite nur einmal vergeben.

Variabel: Nein

Einschränkungen: Darf nicht leer sein, keines der Zeichen '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Primärindex

Identifiziert das Unterelement. Je nachdem, ob "als Text", "als Zahl" oder "als Regexp" gewählt ist, wird das Unterelement durch direkten Textvergleich, über seinen Index, oder durch Vergleich mit der Regexp ermittelt.

Es ist möglich, einen leeren Primärindex anzugeben, z.B. für eine Tabellenspalte ohne Überschrift.

Variabel: Ja

Einschränkungen: Muss ggf. eine gültige Zahl oder eine gültige Regexp sein.

Sekundärindex

Die Klasse `JTable` unterstützt zwei Arten von Unterelementen: Ist nur der Primärindex angegeben, ist eine ganze Spalte gemeint. Ein zusätzlich vorhandener Sekundärindex wählt in dieser Spalte eine Zelle aus.

Der Sekundärindex muss zunächst über die Checkbox aktiviert sein, bevor sein Wert festgelegt werden kann. Nur dadurch kann ein leerer Index von einem nicht vorhandenen unterschieden werden. Ein leerer Sekundärindex definiert eine Zelle mit leerem Inhalt.

Variabel: Ja

Einschränkungen: Muss ggf. eine gültige Zahl oder eine gültige Regexp sein.

Als Text

Der Primär- bzw. Sekundärindex wird als Text interpretiert. Dieser Text wird mit dem dargestellten Text der Unterelemente der komplexen Komponente verglichen.

Variabel: Nein

Einschränkungen: Keine

Als Zahl

Der Primär- bzw. Sekundärindex wird als Zahl interpretiert. Diese legt die Position des Unterelements fest, beginnend bei 0.

Variabel: Nein

Einschränkungen: Keine

Als Regexp

Der Primär- bzw. Sekundärindex wird als Regexp interpretiert. Diese Regexp wird mit dem dargestellten Text der Unterelemente verglichen.

Variabel: Nein

Einschränkungen: Keine

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der

Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.13.5 Fenstergruppe

Eine Fenstergruppe dient dazu, bei größeren Programmen mit vielen Fenstern eine bessere Strukturierung zu ermöglichen.

Enthalten in: Fenster und Komponenten[942], Fenstergruppe[939].

Kinder: Fenstergruppe[939], Fenster[919].

Ausführung: Kann nicht ausgeführt werden.

Attribute:

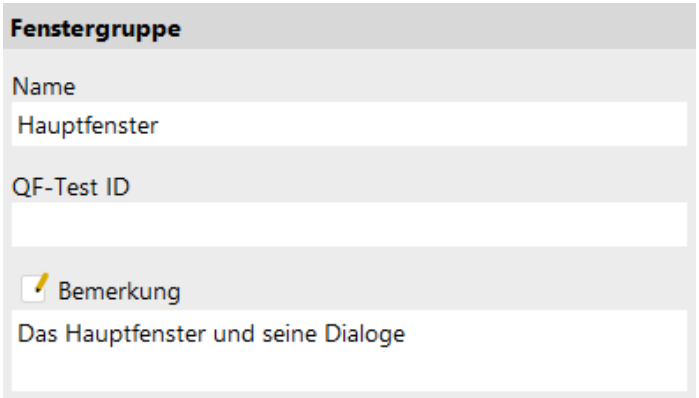


Abbildung 42.92: Fenstergruppe-Attribute

Name

Der Name ist frei wählbar. Er wird im Baum angezeigt und soll Ihnen helfen, die Übersicht über die Fenster zu behalten.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.13.6 Komponentengruppe

Eine Komponentengruppe dient dazu, bei größeren Fenstern mit vielen Komponenten eine bessere Strukturierung zu ermöglichen.

Enthalten in: Fenster[919], Komponentengruppe[940].

Kinder: Komponente[930], Komponentengruppe[940].

Ausführung: Kann nicht ausgeführt werden.

Attribute:

Abbildung 42.93: Komponentengruppe-Attribute

Name

Der Name ist frei wählbar. Er wird im Baum angezeigt und soll Ihnen helfen, die Übersicht über die Komponenten eines Fensters(919) zu behalten.

Variabel: Nein

Einschränkungen: Keine

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.13.7 Fenster und Komponenten

Der Fenster und Komponenten Knoten ist der Platz in der Testsuite, an dem die Fenster und Komponenten untergebracht sind.

Enthalten in: Wurzelknoten

Kinder: Fenster[919], Fenstergruppe[939].

Ausführung: Kann nicht ausgeführt werden.

Attribute:

Abbildung 42.94: Fenster und Komponenten-Attribute

QF-Test ID

Die QF-Test ID ist für diesen Knoten zur Zeit ohne Bedeutung.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor[498] kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags[1360].

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.14 Historische Knoten

Die folgenden Knoten wurden im Laufe der Entwicklungszeit von QF-Test angeboten, werden aber nicht mehr weitergepflegt. Aus Gründen der Rückwärtskompatibilität können diese aber noch ausgeführt werden.

42.14.1 Test

Hinweis

Vor QF-Test Version 2 war der Test Knoten einer der zentralen Bausteine einer Testsuite. Er hat allerdings einige Schwachpunkte. So ist zum Beispiel seine Struktur für Vorbereitungs- und Aufräumarbeiten für komplexe Szenarios zu unflexibel. Außerdem war nie klar, ob ein Test Knoten einen logischen Testfall repräsentiert oder nur zur Implementierung einer einfachen Sequenz dient. Daher wurden Test Knoten durch Testfallsatz[606] und Testfall[599] Knoten abgelöst.

Ein Test ist eine besondere Sequenz, die es erlaubt, vor und nach der Ausführung jedes ihrer Childknoten spezielle Maßnahmen zu treffen, um einen gesicherten Ausgangszustand herzustellen. Dazu kann der Test neben seinen normalen Children als erstes Child einen Vorbereitung[638] und als letztes Child einen Aufräumen[641] Knoten enthalten. Die Vorbereitung wird vor, das Aufräumen nach jedem normalen Childknoten ausgeführt.

Mit seinem Attribut Implizit Exceptions fangen[947] bietet ein Test eine besondere Möglichkeit zur Behandlung von Exceptions um den Abbruch eines gesamten Testlaufs durch eine einzelne Exception in einem Test zu verhindern.

Für spezielle Fälle von datengetriebenem Testen kann der Test auch einen Datentreiber[646] enthalten. Im Regelfall wird dies jedoch mit Hilfe von Testfallsätze[606] realisiert, wie in Kapitel 23[319] beschrieben. Dieser Anwendungsfall kann auch per Testschritt Knoten abgedeckt werden.

Aus Gründen der Rückwärtskompatibilität und um den Übergang von den alten Test Knoten zu den modernen Testfallsatz und Testfall Knoten zu erleichtern, behandelt QF-Test Test Knoten im Report analog zu Testfallsatz oder Testfall Knoten, sofern deren

Position in der Hierarchie dies zulässt. In einigen Fällen konnte ein Test Knoten auch wie ein Testschritt behandelt werden, um z.B. datengetriebene Testschritte zu erlauben.

Alte Testsuiten, deren Struktur noch auf Test Knoten basiert, können migriert werden, um die neuen Features von Testfallsätze und Testfälle zu nutzen. Klicken Sie hierzu mit der rechten Maustaste auf einen Knoten, um dessen Kontextmenü anzuzeigen. Wenn eine Transformation erlaubt ist, wird QF-Test anbieten, den Test Knoten in einen Testfallsatz, Testfall oder Testschritt Knoten umzuwandeln.

3.0+ Um eine ganze Hierarchie von Test Knoten zu transformieren müssen Sie von oben nach unten arbeiten oder die Knotentransformation "Testfallsatz rekursiv" anwenden.

Hinweis Sowohl Testfallsatz als auch Testfall Knoten können aus Gründen der Rückwärtskompatibilität Vorbereitung und Aufräumen Knoten enthalten. Bei einem Testfallsatz verhalten sich diese genau wie bei einem Test, d.h. Vorbereitung und Aufräumen Knoten werden vor und nach jedem im Testfallsatz enthalten Test ausgeführt. Bei einem Testfall werden hingegen Vorbereitung und Aufräumen nur einmal ganz zu Beginn und Ende ausgeführt. Enthält ein Testfallsatz oder Testfall Knoten sowohl Abhängigkeit als auch Vorbereitung/Aufräumen Knoten wir die Abhängigkeit zuerst aufgelöst. Vorbereitung und Aufräumen haben keinen Einfluss auf den in Abschnitt 8.6.3(163) beschriebenen Stapel von Abhängigkeiten.

Enthalten in: Alle Arten von Sequenzen(599).

Kinder: Ein optionaler Datentreiber(646) gefolgt von einer optionalen Vorbereitung(638) am Anfang, dann beliebige ausführbare Knoten und ein optionales Aufräumen(641) am Ende.

Ausführung: Die Variablendefinitionen(946) des Tests werden gebunden. Ist ein Datentreiber(646) Knoten vorhanden, wird dieser ausgeführt, um einen entsprechenden Datenkontext zu erzeugen und einen oder mehrere Daten Knoten zu binden mit dem Zweck, über die ermittelten Datensätze zu iterieren (vgl. Kapitel 23(319)). Für jeden normalen Childknoten des Tests wird zunächst die Vorbereitung ausgeführt, dann das Child, dann der Aufräumen Knoten. Zuletzt werden die Variablen des Tests wieder gelöscht.

Attribute:

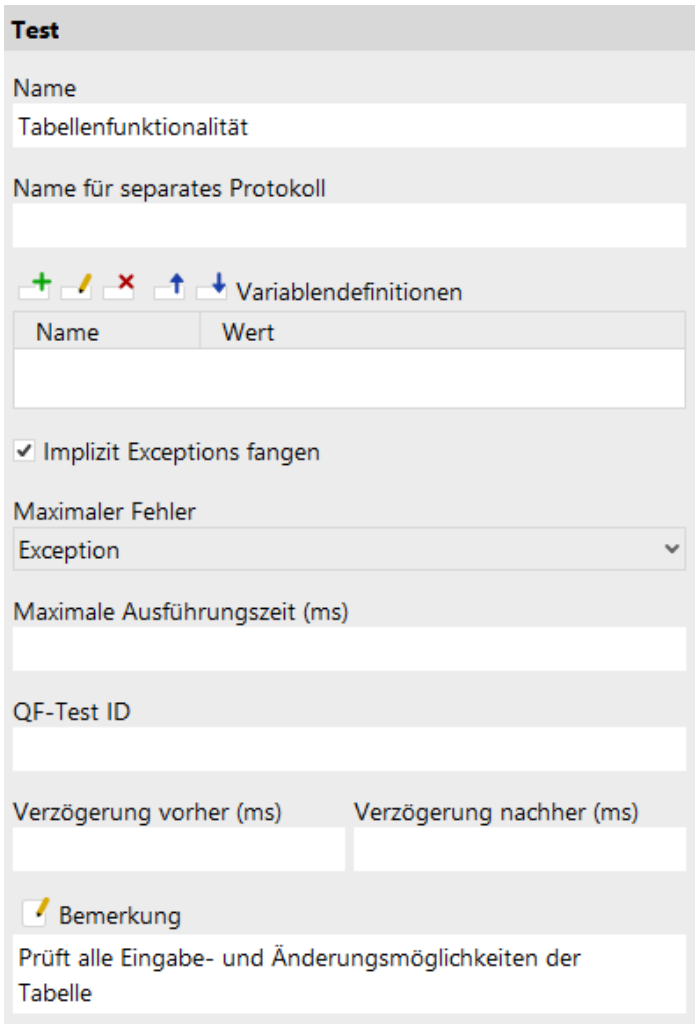


Abbildung 42.95: Test Attribute

Name

Der Name einer Sequenz ist eine Art Kurzkommentar. Er wird in der Baumdarstellung der Testsuite angegeben und sollte etwas über die Funktion der Sequenz aussagen.

Variabel: Nein

Einschränkungen: Keine

Name für separates Protokoll

Mit diesem Attribut kann ein Knoten als Bruchstelle zum Abteilen eines Protokolls markiert werden. Es legt den Dateinamen für das abgeteilte Protokoll fest. Nach Durchlaufen des Knotens wird das zugehörige Protokoll aus dem Hauptprotokoll

entfernt und als eigenständiges Protokoll gespeichert. Diese Operation ist vollständig transparent, da das Hauptprotokoll eine Referenz auf das abgeteilte Protokoll erhält und somit vollständig navigierbar bleibt. Näheres zu geteilten Protokollen finden Sie in Abschnitt 7.1.6[143].

Dieses Attribut hat keinen Effekt, wenn die Option Geteilte Protokolle erzeugen[582] deaktiviert ist oder geteilte Protokolle durch explizite Angabe von `-splitlog`[990] im Batchmodus ausgeschaltet werden.

Es ist nicht nötig für Eindeutigkeit der Dateinamen für abgeteilte Protokolle zu sorgen. Wo nötig hängt QF-Test eine Zahl an den Dateinamen an, um Konflikte zu vermeiden. Der Dateiname darf Verzeichnisse enthalten und es können - analog zur Angabe des Dateinamens für das Protokoll im Batchmodus - folgende Platzhalter in Kombination mit einem '%' oder '+' Zeichen verwendet werden:

Zeichen	Bedeutung
%	'%'-Zeichen.
+	'+'-Zeichen.
i	Die aktuelle Runid wie mit `-runid [<ID>]`[989] angegeben.
r	Die Fehlerstufe des abgeteilten Protokolls.
w	Die Anzahl der Warnungen im abgeteilten Protokoll.
e	Die Anzahl der Fehler im abgeteilten Protokoll.
x	Die Anzahl der Exceptions im abgeteilten Protokoll.
t	Der Threadindex zu dem das abgeteilte Protokoll gehört (für Tests mit parallelen Threads).
y	Das aktuelle Jahr (2 Ziffern).
Y	Das aktuelle Jahr (4 Ziffern).
M	Der aktuelle Monat (2 Ziffern).
d	Der aktuelle Tag (2 Ziffern).
h	Die aktuelle Stunde (2 Ziffern).
m	Die aktuelle Minute (2 Ziffern).
s	Die aktuelle Sekunde (2 Ziffern).

Tabelle 42.40: Platzhalter für das Attribut Name für separates Protokoll

Variabel: Ja

Einschränkungen: Keine, Zeichen die für Dateinamen nicht zulässig sind werden durch '_' ersetzt.

Variablendefinitionen

Hier können Sie Werte für Variablen definieren, die während der Ausführung der Children der Sequenz Gültigkeit besitzen. Näheres zur Arbeit mit der Tabelle finden Sie in Abschnitt 2.2.5[20]. Eine detaillierte Erklärung zur Definition und Verwendung von Variablen finden Sie in Kapitel 6[116].

Variabel: Namen der Variablen nein, Werte ja

Einschränkungen: Keine

Implizit Exceptions fangen

Wenn in einem der Children des Tests eine Exception auftritt, die nicht abgefangen wird, ist der Test normalerweise beendet. Das ist oft nicht wünschenswert, da man keine Information über den Testausgang der weiteren Children erhält.

Wenn Vorbereitung und Aufräumen so beschaffen sind, dass Sie den Ausgangszustand auch dann sicher stellen können, wenn eine Exception in einem normalen Child des Tests auftritt, können Sie hiermit die Exception implizit abfangen.

Anstatt dann bei einer Exception abzubrechen, protokolliert der Test diese und fährt mit dem Aufräumen fort. Anschließend wird der nächste Childknoten ganz normal mit Vorbereitung und Aufräumen ausgeführt.

Exceptions, die während der Vorbereitung oder des Aufräumen auftreten, können auf diesem Weg nicht abgefangen werden.

Variabel: Nein

Einschränkungen: Keine

Maximaler Fehler

Wenn beim Ablauf des Tests innerhalb der Sequenz eine Warnung, ein Fehler oder eine Exception auftritt, wird dieser Status im Protokoll normalerweise an die übergeordneten Knoten weitergeleitet. Mit diesem Attribut können Sie den Fehlerstatus, den das Protokoll für diese Sequenz erhält, beschränken.

Hinweis

Dieser Wert beeinflusst ausschließlich den Status des Protokolls und damit den Rückgabewert von QF-Test falls es im Batchmodus läuft (vgl. Abschnitt 1.7(13)). Auf die Behandlung von Exceptions hat er keinen Einfluss.

Auch für die Erstellung kompakter Protokolle (vgl. Kompakte Protokolle erstellen(590)), hat dieser Wert keinen Einfluss. Eine Sequenz, in der eine Warnung oder ein Fehler auftritt, wird nicht aus einem kompakten Protokoll entfernt, selbst wenn über dieses Attribut der Fehlerstatus auf ”Keinen Fehler” zurückgesetzt wird.

Variabel: Nein

Einschränkungen: Keine

Maximale Ausführungszeit

Zeit in Millisekunden, die der Knoten maximal ausgeführt werden soll. Nach Ablauf dieser Zeit wird der Knoten abgebrochen.

Variabel: Ja

Einschränkungen: >= 0

QF-Test ID

Bei der Ausführung von Tests im Batchmodus kann beim Kommandozeilenargument `-test <Index>|<ID>`(992) alternativ zum qualifizierten Namen die QF-Test ID des Knotens angegeben werden.

Variabel: Nein

Einschränkungen: Darf keines der Zeichen '\', '#', '$', '@', '&', oder '%' enthalten und nicht mit einem Unterstrich ('_') beginnen.

Verzögerung vorher/nachher

Mit diesen Parametern kann vor oder nach der Ausführung eine Verzögerung bewirkt werden. Sind sie nicht gesetzt, wird die Standardverzögerung(551) aus den Optionen verwendet.

Variabel: Ja

Einschränkungen: Leer oder >0

Bemerkung

Hier können Sie einen beliebigen Kommentar eintragen.

Hinweis

Für die ausführliche Dokumentation, insbesondere von Testfallsatz, Testfall oder Prozedur Knoten, ist dieses Textfeld womöglich nicht der geeignete Ort. Es gibt hervorragende Editoren, die wesentlich besser dafür geeignet sind. Mittels der Option Kommando für externen Editor(498) kann ein externer Editor festgelegt werden, in dem nach Drücken von Alt-Eingabe oder Klicken des  Buttons der Kommentar komfortabel bearbeitet werden kann.

Für einige Knoten können Sie ein spezielles Verhalten per Doctags konfigurieren, siehe Doctags(1360).

Falls bei Komponenten Knoten in dem Bemerkungsfeld eine Eintragung gemacht wurde, wird der Knoten bei der Suche bzw. dem Löschen von ungenutzten Komponenten übersprungen.

Variabel: Ja

Einschränkungen: Keine

42.14.2 Prozedur `installCustomWebResolver`

Bis QF-Test 7 erfolgte die Zuordnung der funktionalen Komponenten zu HTML Objekten im Normalfall über die Prozedur `qfs.web.ajax.installCustomWebResolver` aus

der Standardbibliothek `qfs.qft`. Sie wurde durch den CustomWebResolver installieren[902] Knoten ersetzt.

Abbildung 42.96: Aufruf des CustomWebResolvers im Vorbereitung Knoten des Schnellstart-Assistenten

Die Erläuterung der Parameter finden Sie in `installCustomWebResolver` - Parameter[949], die der Parametersyntax in `installCustomWebResolver` - Parametersyntax[954]. Beispiele für die Verwendung der Prozedur Sie daran anschließend.

`installCustomWebResolver` - Parameter

Die Parameter sind entsprechend ihrer Relevanz geordnet. Da die Komponentenerkennung sehr oft auf der Zuordnung von CSS Klassen oder anderen Attributwerten zu generischen Klassen von QF-Test basiert, sind somit die Parameter `genericClasses` und `attributesToGenericClasses` an erster Stelle zu finden.

`installCustomWebResolver`

Konfiguration der Komponentenerkennung für Web-Anwendungen.

Parameter

`resolver`

Kurzname des Frameworks für das die Komponentenerkennung erweitert werden soll:

- `autodetect` (Standardwert), für die automatische Erkennung des verwendeten Frameworks.

- `custom`, wenn die Anwendung mit keinem der von QF-Test unterstützten

Webframeworks erstellt wurde.

- Der Kurzname des Frameworks, z.B. `zk` oder `vaadin`.
Den Kurzname des jeweiligen Frameworks können Sie der Tabelle Tabelle 51.7(1123) entnehmen.
Wenn Sie die Startsequenz über den Schnellstartassistenen erstellen lassen und dort ein Framework angeben, wird der Kurzname direkt hier eingetragen.

`version`
Die Resolver-Version, z.B. `1` oder `1.0` oder `1.1.1`.
Wenn eine Beschränkung angegeben wird, wird innerhalb dieser die letzte verfügbare Version verwendet. Also würde bei `1.0` die letzte verfügbare `1.0.x` verwendet. Wird nichts angegeben, wird die letzte verfügbare Version verwendet.
Bei automatischer Erkennung den Parameter entweder aus dem Prozeduraufruf löschen oder leer lassen.

`genericClasses`
(Optional) Eine Liste von Zuweisungen, die CSS Klassen auf generische QF-Test Klassen mappen.
Dieser Parameter nutzt nur das `class` Attribut des GUI-Elements. Im `class` Attribut können mehrere css-Klassen genannt werden. Diese sind durch Leerzeichen getrennt. Es kann für das Mapping eines GUI-Elements eine dieser css-Klassen angegeben werden.
Kann durch Einträge in `attributesToGenericClasses` überschrieben werden.
z.B. `css-button=Button,ui-table=Table`.
Beispiel für mehrere css-Klassen im GUI-Element:
`class="button css-button active"`. Im obigen Beispiel wurde für das Mapping die css-Klasse `css-button` gewählt.
(verwendet node.getAttribute(`class`)).

`attributesToGenericClasses`
(Optional) Eine Liste von Zuweisungen, die GUI-Elemente mit den angegebenen Attributwerten QF-Test Komponenten der jeweiligen generischen Klasse zuordnen,
Es wird der komplette Wert des angegebenen Attributs geprüft. Zuweisungen über diesen Parameter können Zuweisungen im Parameter `genericClasses` überschreiben.
z.B. `id=table=Table,name=%.*combo.*=ComboBox`.
Beispiel zum Überschreiben des Parameters `genericClasses` mit mehreren css-Klassen:
`class=button css-button active=Button`.

`tagsToGenericClasses`

(Optional) Eine Liste von Zuweisungen, die GUI-Elemente mit dem angegebenen Tag QF-Test zu Komponenten der angegebenen generischen Klasse zuordnen.
z.B. `LI=ListItem`.
Tags müssen in Großbuchstaben angegeben werden.

ignoreTags
(Optional) Eine Liste von Klassennamen oder Tags, für deren Komponenten keine Knoten in der Komponentenhierarchie erstellt werden, solange diese nicht durch andere Anweisungen gemappt werden. Tags müssen in Großbuchstaben angegeben werden.
z.B. werden durch den Eintrag `DIV,TBODY` alle `DIV` und `TBODY` Elemente, die nicht anderweitig gemappt wurden, bei der Erstellung des Komponentenbaums ignoriert.

ignoreByAttributes
(Optional) Eine Liste von Attributwerten, für deren Komponenten keine Knoten in der Komponentenhierarchie erstellt werden,
z.B. `id=container,id=header`.

autoIdPatterns
(Optional) Eine Liste von Mustern, aus denen abgeleitet werden kann, ob Ids automatisch über das Framework generiert wurden. Falls das `id` Attribut dem Muster entspricht, wird der Wert nicht für das Name Attribut der Komponente verwendet,
z.B. `myAutoId,%auto.*`.

customIdAttributes
(Optional) Eine Liste von Attributnamen, deren Werte als Ids für die Komponente verwendet werden können,
z.B. bewirkt `myid,qft-id`, dass die Attribute `myid` und `qft-id` als Ids interpretiert werden.

interestingByAttributes
(Optional) Eine Liste von Attributwerten, die angeben, ob eine Komponente für die Wiedererkennung interessant ist und somit ein Knoten dafür in der Komponentenhierarchie erstellt werden soll,
z.B. `id=container,id=header`.

attributesToQftFeature
(Optional) Eine Liste von Attributnamen, deren Werte für das Merkmal Attribut der Komponente verwendet werden sollen.

documentJS
(Optional) Javascript Code, der in die Webseite eingefügt (injected) werden soll. Kann verwendet werden um benutzerspezifische Javascript Funktionen einzufügen.

attributesToQftName
(Optional) Eine Liste von Attributnamen, der Werte für das Name Attribut der Komponente verwendet werden sollen.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.

nonTrivialClasses
(Optional) Eine Liste von CSS Klassen, für die die zugehörigen Elemente von QF-Test nicht ignoriert werden sollen. Triviale Klassen sind I, FONT, BOLD etc. Wenn die zugehörigen Komponenten (z.B. ein Bild innerhalb eine Buttons) ansprechbar sein sollen, muss die entsprechende triviale Klasse einer generischen QF-Test Klasse zugewiesen werden.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.

allBrowsersSemihardClasses
(Optional) Eine Liste von Klassen, auf deren Komponenten semi-harte Events abgespielt werden sollen. Gilt für alle Browser. z.B. spielt der Eintrag `Button` semi-harte Klicks auf Buttons ab.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie die globale Option `Options.OPT_WEB_SEMI_HARD_EVENTS` auf ”true”, setzen, was sich auf alle Komponenten auswirkt.

chromeSemihardClasses
(Optional) Eine Liste von Klassen, auf deren Komponenten semi-harte Events abgespielt werden sollen. Gilt für Chrome. z.B. spielt der Eintrag `Button` semi-harte Klicks auf Buttons im Chrome Browser ab.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie die globale Option `Options.OPT_WEB_SEMI_HARD_EVENTS` auf ”true”, setzen, was sich auf alle Komponenten auswirkt.

ieSemihardClasses
(Optional) Eine Liste von Klassen, auf deren Komponenten semi-harte Events abgespielt werden sollen. Gilt für alle den Internet Explorer. z.B. spielt der Eintrag `Button` semi-harte Klicks auf Buttons im Internet Explorer ab.

Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie die globale Option `Options.OPT_WEB_SEMI_HARD_EVENTS` auf ”true”, setzen, was sich auf alle Komponenten auswirkt.

`mozSemihardClasses`
(Optional) Eine Liste von Klassen, auf deren Komponenten semi-harte Events abgespielt werden sollen. Gilt für Firefox. z.B. spielt der Eintrag `Button` semi-harte Klicks auf Buttons im Firefox Browser ab.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie die globale Option `Options.OPT_WEB_SEMI_HARD_EVENTS` auf ”true”, setzen, was sich auf alle Komponenten auswirkt.

`edgeSemihardClasses`
(Optional) Eine Liste von Klassen, auf deren Komponenten semi-harte Events abgespielt werden sollen. Gilt für Edge. z.B. spielt der Eintrag `Button` semi-harte Klicks auf Buttons in Edge ab.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie die globale Option `Options.OPT_WEB_SEMI_HARD_EVENTS` auf ”true”, setzen, was sich auf alle Komponenten auswirkt.

`allBrowsersHardClasses`
(Optional) Eine Liste von Klassen, auf deren Komponenten harte Events abgespielt werden sollen. Gilt für alle Browser. z.B. spielt der Eintrag `Button` harte Klicks auf Buttons ab.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie ”Als harten Event wiedergeben” in den Mausevent Knoten aktivieren.

`chromeHardClasses`
(Optional) Eine Liste von Klassen, auf deren Komponenten harte Events abgespielt werden sollen. Gilt für Chrome. z.B. spielt der Eintrag `Button` harte Klicks auf Buttons ab.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie ”Als harten Event wiedergeben” in den Mausevent Knoten aktivieren.

ieHardClasses
(Optional) Eine Liste von Klassen, auf deren Komponenten harte Events abgespielt werden sollen. Gilt für den Internet Explorer. z.B. spielt der Eintrag `Button` harte Klicks auf Buttons ab.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie "Als harten Event wiedergeben" in den Mausevent Knoten aktivieren.

mozHardClasses
(Optional) Eine Liste von Klassen, auf deren Komponenten harte Events abgespielt werden sollen. Gilt für Firefox. z.B. spielt der Eintrag `Button` harte Klicks auf Buttons ab.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie "Als harten Event wiedergeben" in den Mausevent Knoten aktivieren.

edgeHardClasses
(Optional) Eine Liste von Klassen, auf deren Komponenten harte Events abgespielt werden sollen. Gilt für Edge. z.B. spielt der Eintrag `Button` harte Klicks auf Buttons ab.
Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Support Team.
Alternativ können Sie "Als harten Event wiedergeben" in den Mausevent Knoten aktivieren.

`installCustomWebResolver` - Parametersyntax

Wenn für einen Parameter mehrere Einträge möglich sind, so müssen die einzelnen Zuweisungen oder Ausdrücke in den Parameterlisten durch Kommas voneinander getrennt werden. Nach dem Komma kann ein Zeilenumbruch eingefügt werden, aber **kein** Leerzeichen!

Die Parametersyntax umfasst folgende Ausdrücke, nach Relevanz geordnet:

%
`%` gibt an, das die folgende Zeichenkette als regulärer Ausdruck interpretiert werden soll.

`%list.*` bezeichnet alle Werte, die mit `list` beginnen.

Nutzbar für alle Parameter.

css-class=generic class
Ordnet einem GUI-Element mit der angegebenen CSS Klasse eine QF-Test Komponente der angegebenen generischen Klasse zu.

`css-button=Button` ordnet dem GUI-Element mit der CSS Klasse `css-button` eine QF-Test Komponente mit der generischen Klasse `Button` zu.

Nutzbar im Parameter `genericClasses`.

attribute=value=generic class
Ordnet einem GUI-Element mit dem angegebenen Attributwert eine QF-Test Komponente der angegebenen generischen Klasse zu.

`role=toggle=RadioButton` weist Elementen mit `role` Attributen, die den Wert `toggle` haben, QF-Test Komponenten der Klasse `RadioButton` zu.

Nutzbar im Parameter `attributesToGenericClasses`.

TAG=generic class
Ordnet einem GUI-Element mit dem angegebenen Tag eine QF-Test Komponente der angegebenen generischen Klasse zu. Tags müssen in Großbuchstaben angegeben werden.

`LI=ListItem` ordnet einem GUI-Element mit dem Tag `li` eine QF-Test Komponente mit der generischen Klasse `ListItem` zu.

Nutzbar im Parameter `tagsToGenericClasses`.

@::ancestor=classname oder TAG
Kann an einen Eintrag der Parameterliste angehängt werden. Der Eintrag wird nur dann ausgewertet, wenn eines der übergeordneten Objekte des GUI-Elements den angegebenen generischen Klassenamen oder das angegebene Tag (in Großbuchstaben) hat.

Nutzbar in allen Parametern.

Beispiel für den Parameter `tagsToGenericClasses`:

`LI=TableCell@::ancestor=TableRow` ordnet einem GUI-Element mit dem Tag `li` eine QF-Test Komponente der generischen Klasse `TableCell` zu, wenn ein übergeordnetes Element die Klasse `TableRow` hat.

...=TAG
Die vorangehende Zuweisung wird nur ausgewertet, wenn das GUI-Element das angegebene Tag hat. Muss bei Bedarf vor `@::` stehen.
Tags müssen in Großbuchstaben angegeben werden.

Nutzbar in allen Parametern.

Beispiel für den Parameter `genericClasses`:

`row=TableRow=SPAN` ordnet einem GUI-Element der CSS-Klasse `row` eine QF-Test Komponente mit der generischen Klasse `TableRow` zu, wenn das Tag den Wert `SPAN` hat.

Beispiel für den Parameter `genericClasses`:

`row=TableRow=SPAN@::ancestor=Table` ordnet einem GUI-Element der CSS-Klasse `row` eine QF-Test Komponente mit der generischen Klasse `TableRow` zu, wenn das Tag den Wert `SPAN` hat und wenn eines der übergeordneten Elemente die Klasse. `Table` hat.

Beispiel für den Parameter `interestingByAttributes`:

`myid=%.*=CONTAINER` nimmt GUI-Elemente mit dem Tag `container` auf, aber nur, wenn das Attribut den Wert `myid` hat.

@::parent=classname oder TAG

Kann an einen Eintrag der Parameterliste angehängt werden. Der Eintrag wird nur dann ausgewertet, wenn eines der übergeordneten Objekte des GUI-Elements den angegebenen generischen Klassenamen oder das angegebene Tag (in Großbuchstaben) hat.

Nutzbar in allen Parametern.

Beispiel für den Parameter `genericClasses`:

`css-data-row=TableRow@::parent=Table`
ordnet dem GUI-Element mit der CSS Klasse `css-data-row` eine QF-Test Komponente der generischen Klasse `TableRow` zu, wenn das direkt übergeordnete GUI-Element die Klasse `Table` hat.

@::parent<level>=classname oder TAG

Kann an einen Eintrag der Parameterliste angehängt werden. Der Eintrag wird nur dann ausgewertet, wenn eines der übergeordneten Objekte des GUI-Elements den angegebenen generischen Klassenamen oder das angegebene Tag (in Großbuchstaben) hat.

Die Ebene bezieht sich auf die von QF-Test aufgenommene Komponentenstruktur bzw. den generierten DomNode. Dies bedeutet, dass die Komponentenerkennung fehlschlagen kann, wenn sich die Webseite ändert oder Sie Resolver-Einträge ändern. Es könnte besser sein, statt dessen `@::ancestor` zu verwenden. Alternativ können Sie das entscheidende übergeordnete GUI-Element einer eigenen Klasse (oder Klassentyp) zuweisen und diese dann in `@::parent` oder `@::ancestor` einsetzen. Das Beispiel in CustomWebResolver – Baum (Tree)(1099) verdeutlicht diese Technik.

Nutzbar in allen Parametern.

Beispiel für den Parameter `genericClasses`:

`css-button=Button:ComboBoxButton@::parent<3>=ComboBox`
Nimmt einen Button mit dem Typ `ComboBoxButton` auf, wenn der übergeordnete Knoten in Ebene drei die Klasse `ComboBox` hat.

@::ancestor<level>=classname oder TAG
Kann an einen Eintrag der Parameterliste angehängt werden. Der Eintrag wird nur dann ausgewertet, wenn eines der übergeordneten Objekte des GUI-Elements den angegebenen generischen Klassenamen oder das angegebene Tag (in Großbuchstaben) hat. (Ebenenzählung wie bei @::parent.)

Nutzbar in allen Parametern.

Beispiel für den Parameter `genericClasses`:

`cbx=CheckBox:ListItemCheckBox@::ancestor<3>=List` Nimmt eine CheckBox mit dem Typ `ListItemCheckBox` auf, wenn einer der Knoten in den drei übergeordneten Ebenen die Klasse `List` hat.

attribute=value
Nutzbar in den Parametern `ignoreByAttributes` und `interestingByAttributes`.

Im Parameter `ignoreByAttributes` bewirkt die Zuweisung, dass GUI-Elemente mit dem angegebenen Attributwert in der Baumhierarchie nicht angelegt werden.

Im Parameter `interestingByAttributes` bewirkt die Zuweisung, dass im Komponentenbaum Knoten für Elemente angelegt werden, die den angegebenen Attributwert aufweisen.

Beispiel für den Parameter `ignoreByAttributes`:

`type=container` bewirkt, dass im Komponentenbaum keine Knoten für Elemente angelegt werden, bei denen das Attribut `type` den Wert `container` hat.

Beispiel für den Parameter `interestingByAttributes`:

`type=splitpane` bewirkt, dass im Komponentenbaum Knoten für Elemente angelegt werden, bei denen das Attribut `type` den Wert `splitpane` hat.

Für Interessierte:
@::ancestor=`class` nutzt intern die Objekt-Methode obj.getAncestorOfClass(`class`).
@::ancestor<`level`>=`class` nutzt intern die Objekt-Methode obj.getAncestorOfClass(`class, level`)
@::parent= nutzt intern die Objekt-Methode obj.getParent()
@::parent<`level`>= nutzt intern die Objekt-Methode obj.getNthParent(`level`)
Details zu den Methoden finden Sie in Pseudo DOM API für Web-Anwendungen[1253].

Kapitel 43

Exceptions

Es gibt viele Exceptions, die bei der Ausführung eines Tests mit QF-Test geworfen werden können. Dieses Kapitel führt alle Exceptions mit ihren typischen Fehlermeldungen und einer kurzen Erklärung auf.

Wenn Sie mit Exceptions in Skripten arbeiten wollen, werfen Sie bitte ein Blick in Abschnitt 50.10(1074).

`TestException`

Dies ist die Basisklasse aller QF-Test Exceptions. Eine konkrete Exception ist nie von dieser, sondern immer von einer davon abgeleiteten Klasse. Ein Catch(707) Knoten, dessen Exception-Klasse(708) auf `TestException` gesetzt ist, fängt alle möglichen Exceptions. Wie in Java selbst sollten Sie einen solchen Catch(707) nur in Ausnahmefällen verwenden, da dadurch Exceptions verschleiert werden können, mit denen zunächst nicht gerechnet wurde.

`ComponentNotFoundException`

Diese Exception wird geworfen, wenn die Zielkomponente eines Events(775) oder Checks(805) nicht ermittelt werden kann. Auch das Scheitern eines Warten auf Komponente(876) Knotens führt zu einer `ComponentNotFoundException`, sofern dessen Warten auf Verschwinden(878) Attribut nicht gesetzt ist.

`ScopeNotFoundException`

Diese Exception wird geworfen, wenn die Zielkomponente für den expliziten Scope eines Events(775) oder Checks(805) nicht ermittelt werden kann. Auch das Scheitern eines Warten auf Komponente(876) Knotens mit einem expliziten

Scope führt zu einer `ScopeNotFoundException`, sofern dessen Warten auf Verschwinden(878) Attribut nicht gesetzt ist.

`DocumentNotLoadedException`

Diese Exception ist eine Variante der `ComponentNotFoundException` und wird speziell beim Fehlschlagen eines Warten auf Laden des Dokuments(880) Knotens geworfen.

`PageNotFoundException`

Diese Exception ist eine Variante der `ComponentNotFoundException` und wird speziell beim Fehlschlagen eines Auswahl(793) Knotens geworfen, wenn in einem PDF-Dokument die anzuzeigende Seite nicht gefunden wurde.

`ComponentFoundException`

Dies ist das Gegenstück zur `ComponentNotFoundException` und wird geworfen, wenn ein Warten auf Komponente(876) mit gesetztem Warten auf Verschwinden(878) Attribut scheitert.

`ModalDialogException`

Diese Exception wird geworfen, wenn ein Event(775) von einem modalen Dialog blockiert wird. Siehe Option Auf modale Dialoge prüfen(543).

`ComponentCannotGetFocusException`

Hinweis

Diese Ausnahme ist veraltet und sollte nicht mehr auftreten.

Diese Exception wird geworfen, wenn die Zielkomponente für einen Tastaturevent(780) oder Texteingabe(784) Knoten ein Textfeld ist, das aus irgendwelchen Gründen den Tastaturfokus nicht erhalten kann. Mit JDK 1.4 führt dies dazu, dass der Event nicht an die Komponente übermittelt werden kann.

`DisabledComponentException`

Diese Exception wird geworfen, wenn die Zielkomponente für einen Mausevent(775), Tastaturevent(780) oder Texteingabe(784) Knoten deaktiviert ist. In diesem Fall würden die Events einfach ignoriert und damit höchstwahrscheinlich zu Fehlern im weiteren Verlauf des Tests führen.

Aus Gründen der Rückwärtskompatibilität kann diese Exception durch Deaktivieren der Option DisabledComponentExceptions werfen(544) unterdrückt werden.

DisabledComponentStepException

Diese Exception wird geworfen, wenn die Zielkomponente für einen Mausevent[775], Tastaturevent[780] oder Texteingabe[784] Knoten unter Fenster und Komponenten deaktiviert ist.

BadItemException

Diese Exception wird von Item-Resolvern geworfen, wenn der Typ des Elements und des Items nicht passen.

ExecutionTimeoutExpiredException

Diese Exception wird geworfen, wenn die angegebene maximale Ausführungszeit eines Knotens abgelaufen ist.

BusyPaneException

Diese Exception wird geworfen, wenn die Zielkomponente für einen Mausevent[775], Tastaturevent[780] oder Texteingabe[784] Knoten von einer GlassPane mit einem 'busy' Mauszeiger verdeckt ist. In diesem Fall würden die Events normalerweise von der GlassPane abgefangen werden und damit höchstwahrscheinlich zu Fehlern im weiteren Verlauf des Tests führen. Die Option Warten bei 'busy' GlassPane (ms)[556] legt fest, wie lange QF-Test auf das Verschwinden der GlassPane wartet, bevor die Exception geworfen wird.

InvisibleDnDTargetException

Diese Exception wird geworfen, wenn der Zielpunkt in der Komponente für einen Mausevent[775] vom Typ `DRAG_FROM`, `DRAG_OVER` oder `DROP_TO` für eine Drag&Drop Operation nicht sichtbar ist und nicht durch Scrollen der Zielkomponente sichtbar gemacht werden kann.

InvisibleTargetComponentException

Diese Exception wird geworfen, wenn die Zielkomponente unsichtbar ist.

InvisibleTargetItemException

Diese Exception wird geworfen, wenn zwar die umgebende Zielkomponente sichtbar ist, aber das Zielelement darin unsichtbar.

DeadlockTimeoutException

Diese Exception wird geworfen, wenn das SUT für einen vorgegebenen Zeitraum nicht reagiert. Dieser Zeitraum wird durch die Option Erkennen von Deadlocks (s)[554] festgelegt.

DownloadNotCompleteException

Diese Exception wird geworfen, wenn auf das Herunterladen eines Dokuments gewartet wird, dies aber nicht abgeschlossen wurde.

DownloadStillActiveException

Diese Exception wird geworfen, wenn versucht wird, eine Datei herunter zu laden, die noch durch einen anderen Download blockiert ist.

NoSuchDownloadException

Diese Exception wird geworfen, wenn auf einen Download gewartet wird, bei dem das Herunterladen nicht gestartet hat.

VariableException

Diese Exception wird nicht selbst geworfen, sondern ist die Basisklasse für verschiedene Exceptions, die im Zusammenhang mit Variablenexpansion geworfen werden können.

BadVariableSyntaxException

Diese Exception wird geworfen, wenn ein Ausdruck bei der Variablenexpansion keine gültige Syntax aufweist, z.B. weil die schließende Klammer fehlt.

MissingPropertiesException

Diese Exception wird geworfen, wenn keine Properties oder kein ResourceBundle für die Gruppe einer erweiterten Variablenexpansion der Form `${Gruppe:Name}` verfügbar sind (siehe auch Ressourcen laden[890] und Properties laden[893]).

MissingPropertyException

Diese Exception wird geworfen, wenn der Name bei einer erweiterten

Variablenexpansion der Form `${Gruppe:Name}` nicht verfügbar ist (siehe auch Ressourcen laden[890] und Properties laden[893]).

`ReadOnlyPropertyException`

9.0+ Diese Exception wird geworfen, wenn versucht wird, einen Wert in einer speziellen Gruppe zu ändern, bei dem dies nicht möglich ist. (siehe Spezielle Gruppen[127]).

`RecursiveVariableException`

Diese Exception wird geworfen, wenn die Expansion einer Variablen zu Rekursion führt, z.B. wenn Sie eine Variable namens `x` auf den Wert `$(y)` und die Variable namens `y` auf `$(x)` setzen und versuchen, den Wert `$(x)` zu expandieren.

`UnboundVariableException`

Diese Exception wird geworfen, wenn eine Variable bei einer Variablenexpansion nicht existiert.

`VariableNumberException`

Diese Exception wird geworfen, wenn das Ergebnis einer Variablenexpansion für ein numerisches Attribut keine Zahl darstellt.

`BadExpressionException`

Diese Exception wird geworfen, wenn ein Ausdruck der Form $[...] fehlerhaft ist (vgl. Abschnitt 11.2[189]).

`BadTestException`

Diese Exception wird geworfen, wenn die Bedingung[694] eines If[693] oder Elseif[697] Knotens eine ungültige Syntax aufweist.

`BadRegexpException`

Diese Exception wird geworfen, wenn ein regulärer Ausdruck keine gültige Syntax aufweist (vgl Abschnitt 49.3[1023]), z.B. bei einem Element[936] oder Check Text[806] Knoten.

`BadRangeException`

Diese Exception wird geworfen, wenn das Attribut Iterationsbereiche eines Daten Knotens ungültige Syntax aufweist oder einen Index außerhalb des Datenbereichs enthält.

`CannotExecuteException`

Diese Exception wird geworfen, wenn der Start eines Prozesses durch einen Java-SUT-Client starten(724) Knoten fehlschlägt.

`InvalidDirectoryException`

Diese Exception wird geworfen, wenn das Verzeichnis(726) Attribut eines Java-SUT-Client starten(724) Knotens auf ein nicht existierendes Verzeichnis verweist.

`CheckFailedException`

Diese Exception wird geworfen, wenn ein Check(805) fehlschlägt, dessen Im Fehlerfall Exception werfen(810) Attribut gesetzt ist.

`CheckNotSupportedException`

Wie im Abschnitt über Checks(805) beschrieben, kann sich jeder Check nur auf bestimmte Komponenten beziehen. Diese Exception wird geworfen, wenn die Zielkomponente eines Checks für diesen keinen Sinn ergibt.

`OperationNotSupportedException`

Diese Exception wird geworfen, wenn eine Operation wie Text auslesen(839) für die angegebene Zielkomponente nicht möglich ist.

`BadComponentException`

Diese Exception wird geworfen, wenn eine Komponente für einen Event(775) keinen Sinn ergibt, z.B. etwas anderes als ein Fenster für einen Fensterevent(787).

`IndexFormatException`

Diese Exception wird geworfen, wenn der Index eines Unterelements in einem ungültigen Format angegeben ist (vgl. Abschnitt 5.9.1(95)).

IndexFoundException

Diese Exception wird geworfen, wenn ein Unterelement gefunden wird, obwohl bei der Ausführung eines Warten auf Komponente(876) Knotens auf dessen Abwesenheit geprüft wird.

IndexNotFoundException

Diese Exception wird geworfen, wenn kein Unterelement für einen angegebenen Index gefunden wurde.

IndexRequiredException

Diese Exception wird geworfen, wenn ein für eine Operation notwendiger Index nicht angegeben wurden, z.B. für einen Check Text(806) Knoten für einen `JTree`.

UnexpectedIndexException

Diese Exception wird geworfen, wenn ein Index für ein Unterelement angegeben ist, obwohl eine Operation keinen benötigt, z.B. für einen Check Elemente(818) Knoten für einen `JTree`.

ClientNotConnectedException

Diese Exception wird geworfen, wenn keine Verbindung zum angegebenen SUT Client für eine Operation besteht, obwohl im Gegensatz zur `NoSuchClientException`(964) ein Prozess unter diesem Namen verfügbar ist.

CannotAttachException

Diese Exception wird geworfen, die Verbindung zu der Windows-Anwendung fehlschlug.

ConnectionFailureException

Diese Exception wird geworfen, wenn die Verbindung zu einem Client fehlerhaft ist.

NoSuchClientException

Diese Exception wird geworfen, wenn der angegebene SUT Client für eine Operation nicht existiert.

`NoSuchEngineException`

Diese Exception wird geworfen, wenn eine Engine referenziert wird, die nicht existiert oder wenn versucht wird, auf eine Engine zurückzugreifen, die (noch) nicht mit QF-Test verbunden wurde Dies kann passieren, wenn die entsprechende Technologie in der Anwendung nicht oder erst zu einem späteren Zeitpunkt verwendet wird.

`DuplicateClientException`

Diese Exception wird geworfen, wenn versucht wird, mehr als einen Prozess gleichzeitig unter dem selben Namen zu starten.

`UnexpectedClientException`

Diese Exception wird geworfen, wenn im SUT eine unerwartete Exception beim Abspielen eines Events geworfen wird. Sofern es sich dabei nicht um einen Bug in QF-Test handelt, deutet dies auf ein echtes Problem im SUT hin.

`ExtensionException`

Diese Exception wird geworfen, wenn in einem Client eine unerwartete Exception auftritt.

`ClientNotTerminatedException`

Diese Exception wird geworfen, wenn bei der Ausführung eines Warten auf Programmende(771) Knotens der Prozess nicht terminiert.

`UnexpectedExitCodeException`

Diese Exception wird geworfen, wenn der Exitcode eines Prozesses nicht den Vorgaben des Attributs Erwarteter Exitcode(773) eines Warten auf Programmende(771) Knotens entspricht.

`BadExitCodeException`

Diese Exception wird geworfen, wenn das Attribut Erwarteter Exitcode(773) eines Warten auf Programmende(771) Knotens nicht der Spezifikation entspricht.

`ComponentIdMismatchException`

Diese Exception wird geworfen, wenn das QF-Test ID der Komponente(776) Attribut eines Knotens auf einen Knoten verweist, der nicht vom Typ Fenster(919), Komponente(930) oder Element(936) ist.

`UnresolvedComponentIdException`

Diese Exception wird geworfen, wenn das Ziel eines QF-Test ID der Komponente(776) Attributs nicht existiert.

`TestNotFoundException`

Diese Exception wird geworfen, wenn der Testfall(599) oder das Testfallsatz(606) für einen Testaufruf(614) nicht existiert.

`DependencyNotFoundException`

Diese Exception wird geworfen, wenn die Abhängigkeit(630) für einen Bezug auf Abhängigkeit(635) nicht existiert.

`InconsistentDependenciesException`

Diese Exception wird geworfen, wenn die Abhängigkeit(630) aufgrund inkonsistenter Bezüge nicht linearisiert werden können.

`RecursiveDependencyReferenceException`

Diese Exception wird geworfen, wenn die Abhängigkeit(630) aufgrund rekursiver Bezüge nicht linearisiert werden können.

`ProcedureNotFoundException`

Diese Exception wird geworfen, wenn die Prozedur(672) für einen Prozeduraufruf(675) nicht existiert.

`StackOverflowException`

Diese Exception wird geworfen, wenn die Verschachtelungstiefe von Prozeduraufrufen(675) den Wert der Option Größe des Callstacks(531) überschreitet, was auf eine endlose Rekursion von Prozeduraufrufen hindeutet.

`ValueCastException`

Diese Exception wird geworfen, wenn im Variable setzen(871) oder Return(678) Knoten der Wert nicht in den gewünschten Objekttyp konvertiert werden kann.

`UserException`

Diese Exception wird explizit durch einen Throw(714) Knoten geworfen.

`CannotRethrowException`

Diese Exception wird geworfen, wenn versucht wird, durch einen Rethrow(715) Knoten eine Exception erneut zu werfen, ohne dass vorher eine Exception durch einen Catch(707) Knoten abgefangen wurde.

`ScriptException`

Diese Exception wird geworfen, wenn ein Script einen Fehler liefert.

`AndroidSdkException`

Dies ist eine Basisklasse. Um Android-Anwendungen zu testen, benötigt QF-Test ein auf dem Computer installiertes Android SDK. Sollte es hierbei zu Problemen kommen, so wird eine Exception die von dieser Basisklasse abgeleitet ist geworfen.

`InvalidAndroidSdkPathException`

Eine Exceptions dieses Typs wird geworfen, wenn ein angegeber Pfad nicht auf ein gültiges SDK verweist.

`AndroidSdkNotFoundException`

Eine Exceptions dieses Typs wird geworfen wenn das Android SDK nicht auf dem Computer gefunden werden konnte.

`InvalidAndroidAdbPathException`

Diese Exception wird geworfen, wenn der angegebenen ADB-Klassenpfad ungültig ist.

`AndroidAdbNotFoundException`

Diese Exception wird geworfen, wenn kein Android ADB auf dem Rechner gefunden wurde.

`InvalidAndroidEmulatorPathException`

Diese Exception wird geworfen, wenn der angegebenen Pfad für den Android-Emulator ungültig ist.

`AndroidEmulatorNotFoundException`

Diese Exception wird geworfen, wenn kein Android-Emulator auf dem Rechner gefunden wurde.

`AndroidVirtualDeviceException`

Dies ist eine Basisklasse. Um Android-Anwendungen zu Testen sollte der Android-Emulator installiert sein. Dieser Emulator kann dann ein virtuelles Android-Gerät (AVD) ausführen. Wird kein solches virtuelles Android-Gerät gefunden, oder wenn das Testen des angegebenen Emulators nicht möglich ist, so wird eine Exception die von dieser Basisklasse abgeleitet ist geworfen.

`NoAndroidVirtualDeviceException`

Eine Exceptions dieses Typs wird geworfen wenn kein virtuelles Android-Gerät gefunden wird auf dem die Tests ausgeführt werden können.

`NoSupportedAndroidVirtualDeviceException`

Eine Exceptions dieses Typs wird geworfen wenn kein von QF-Test unterstütztes virtuelles Android-Gerät gefunden wird auf dem die Tests ausgeführt werden können.

`AndroidVirtualDeviceNotFoundException`

Eine Exceptions dieses Typs wird geworfen wenn das angegebene virtuelle Android-Gerät nicht gefunden werden konnte.

`AndroidVirtualDeviceNotSupportedException`

Eine Exceptions dieses Typs wird geworfen wenn das angegebene virtuelle Android-Gerät nicht unterstützt wird.

`AndroidVirtualDeviceParsingException`

Eine Exceptions dieses Typs wird geworfen wenn es für QF-Test nicht möglich ist die Zeichenkette zu parsen die den zu testenden Emulator spezifiziert.

`BreakException`

Dies ist keine normale TestException und sie kann nicht von einem Catch[707] Knoten gefangen werden. Sie wird von einem Break[691] Knoten geworfen um aus einer Schleife auszubrechen. In einem Skript erfüllt das Werfen einer `BreakException` den selben Zweck. Wird sie außerhalb einer Schleife geworfen, löst eine `BreakException` einen Fehler aus.

`ReturnException`

Dies ist keine normale TestException und sie kann nicht von einem Catch[707] Knoten gefangen werden. Sie wird von einem Return[678] Knoten geworfen um aus einer Prozedur[672] zurückzukehren. In einem Skript erfüllt das Werfen einer `ReturnException` den selben Zweck. Wird sie außerhalb einer Prozedur geworfen, löst eine `ReturnException` einen Fehler aus.

`TestOutOfMemoryException`

Dies ist eine spezielle Exception, die geworfen wird, wenn QF-Test feststellt, dass der für die Testausführung zur Verfügung stehende Speicher zu Ende geht. Die Exception führt zum sofortigen Abbruch des Testlaufs und kann nicht gefangen werden, da QF-Test ohne Speicher nicht viel tun kann, um sie zu behandeln. Allerdings versucht QF-Test, sich eine kleine Speicherreserve zu erhalten und zumindest das Protokoll zu speichern.

Teil IV

Technische Referenz

Kapitel 44

Kommandozeilenargumente und Rückgabewerte

44.1 Aufrufsyntax

Die Aufrufsyntax für interaktiven und Batchmodus unterscheidet sich deutlich, da einige Kommandozeilenargumente für den interaktiven, andere für den Batchmodus oder einen Sub-Modus davon spezifisch sind. Beachten Sie, dass alle Argumente sinnvolle Voreinstellungen besitzen und nur in besonderen Fällen angegeben werden müssen. Im Allgemeinen verwenden Sie lediglich `qftest [<Suite> | <Protokoll>]*` für den interaktiven Modus oder `qftest -batch [-runlog [<Datei>]] [-report <Verzeichnis>] <Suite>` für den Batchmodus.

5.2+ Für maximale Flexibilität wird bei den Namen der QF-Test Kommandozeilenargumente Groß-/Kleinschreibung ebenso ignoriert, wie enthaltene '-', '_', '.' oder ':'-Zeichen. Somit ist `-report.html` äquivalent zu `-reportHtml` oder `-report-html`. Letzteres ist die offiziell dokumentierte Form, da sie Konflikte mit der Windows PowerShell vermeidet.

Windows Das Programm `qftest.exe` ist eine Windows GUI-Anwendung. Wird diese von der Eingabeaufforderung gestartet, wartet sie nicht darauf, dass QF-Test beendet wird, sondern kehrt sofort zurück. Wenn also ein Test im Batchmodus ausgeführt wird, kann man nicht erkennen, ob QF-Test bereits beendet ist oder nicht (man könnte dem Abhelfen, indem man die Anweisung in eine `.bat` Datei packt). Außerdem werden bei Verwendung von `qftest.exe` keine Ausgaben von QF-Test in der Konsole angezeigt. Besser geeignet für die Ausführung von der Eingabeaufforderung ist daher die Konsolen-Anwendung `qftestc.exe`: Sie wartet, bis QF-Test sich beendet und wenn Server-Skripte(717) print-Ausgaben vornehmen, werden diese auf der Konsole angezeigt. Ansonsten gilt was in diesem Kapitel über `qftest.exe` gesagt wird auch für `qftestc.exe`.

Mac Falls die macOS App verwendet wird können diese Parameter auch direkt via Bearbeiten→Optionen unter Allgemein->Programmstart definiert werden (siehe auch

Aufruf von QF-Test[13]).

Interaktiver Modus

Die Aufrufsyntax für den interaktiven Modus lautet: `qftest [-dbg`[976]`] [-java <Programm> (abgekündigt)`[977]`] [-noconsole`[977]`] [-J<Java-Argument>]* [-allow-shutdown [<Shutdown-ID>]`[978]`] [-daemon`[980]`] [-daemonhost <Host>`[980]`] [-daemonport <Port>`[980]`] [-daemonrmiport <Port>`[980]`] [-dontkillprocesses`[980]`] [-engine <Engine>`[981]`] [-groovydir <Verzeichnis>`[981]`] [-help`[981]`] [-ipv6`[982]`] [-javascriptdir <Verzeichnis>`[982]`] [-jythondir <Verzeichnis>`[982]`] [-jythonport <Nummer>`[982]`] [-keybindings <Wert>`[982]`] [-keystore <Keystore-Datei>`[983]`] [-keypass <Kennwort>`[983]`] [-libpath <Pfad>`[983]`] [-license <Datei>`[983]`] [-license-waitfor <Sekunden>`[983]`] [-logdir <Verzeichnis>`[983]`] [-noplugins`[984]`] [-noupdatecheck`[984]`] [-option <Name>=<Wert>`[985]`] [-options <Datei>`[985]`] [-plugindir <Verzeichnis>`[986]`] [-port <Nummer>`[986]`] [-reuse`[988]`] [-run`[988]`] [-runlogdir <Verzeichnis>`[989]`] [-runtime`[989]`] [-serverhost <Host>`[990]`] [-shell <Programm>`[990]`] [-shellarg <Argument>`[990]`] [-splitlog`[990]`] [-suitesfile <Datei>`[991]`] [-systemcfg <Datei>`[992]`] [-systemdir <Verzeichnis>`[992]`] [-tempdir <Verzeichnis>`[992]`] [-test <Index>|<ID>`[992]`]* [-usercfg <Datei>`[994]`] [-userdir <Verzeichnis>`[994]`] [-variable <Name>=<Wert>`[994]`]* [-version`[995]`] [<Testsuite> | <Protokoll>]*`

Es gibt mehrere Sub-Modi für die Ausführung von QF-Test im Batchmodus. Standardfall ist die Durchführung von Tests aus einer oder mehreren Testsuiten. Alternativ können Test-Dokumentation aus Testsuiten oder Reports aus Protokollen generiert werden. Außerdem kann QF-Test im Daemon-Modus gestartet werden in dem es von außen gesteuert werden kann (siehe Kapitel 55[1276]). Und schließlich kann auch die Anzeige von Hilfs- oder Versionsinformationen als Sub-Modus angesehen werden.

Tests durchführen

Um eine oder mehrere Testsuiten auszuführen und dabei ein Protokoll und/oder einen Report zu erstellen, verwenden Sie: `qftest -batch [-run`[988]`] [-dbg`[976]`] [-java <Programm> (abgekündigt)`[977]`] [-noconsole`[977]`] [-J<Java-Argument>]* [-allow-shutdown [<Shutdown-ID>]`[978]`] [-clearglobals`[979]`] [-compact`[979]`] [-engine <Engine>`[981]`] [-exitcode-ignore-exception`[981]`] [-exitcode-ignore-error`[981]`] [-exitcode-ignore-warning`[981]`] [-groovydir <Verzeichnis>`[981]`] [-ipv6`[982]`] [-javascriptdir <Verzeichnis>`[982]`]`

[-jythondir <Verzeichnis>(982)] [-jythonport <Nummer>(982)]
[-keystore <Keystore-Datei>(983)] [-keypass <Kennwort>(983)]
[-libpath <Pfad>(983)] [-license <Datei>(983)]
[-license-waitfor <Sekunden>(983)] [-logdir <Verzeichnis>(983)]
[-nolog(984)] [-nomessagewindow(984)] [-noplugins(984)]
[-option <Name>=<Wert>(985)] [-options <Datei>(985)]
[-plugindir <Verzeichnis>(986)] [-port <Nummer>(986)]
[-report <Verzeichnis>(986)] [-report-checks(986)]
[-report-customdir <Verzeichnis>(986)]
[-report-doctags(987)] [-report-errors(987)]
[-report-exceptions(987)] [-report-html <Verzeichnis>(987)]
[-report-ignorenotimplemented(987)] [-report-ignoreskipped(987)]
[-report-junit <Verzeichnis>(987)] [-report-name <Name>(987)]
[-report-nodeicons(987)] [-report-passhtml(987)]
[-report-piechart(988)] [-report-include-suitename(988)]
[-report-scale-thumbnails <Prozent>(988)]
[-report-teststeps(988)] [-report-thumbnails(988)]
[-report-warnings(988)] [-report-xml <Verzeichnis>(988)]
[-runid [<ID>](989)] [-runlogdir <Verzeichnis>(989)]
[-runlog [<Datei>](989)] [-runtime(989)] [-serverhost <Host>(990)]
[-shell <Programm>(990)] [-shellarg <Argument>(990)]
[-sourcedir <Verzeichnis>(990)] [-suitesfile <Datei>(991)]
[-splitlog(990)] [-systemcfg <Datei>(992)]
[-systemdir <Verzeichnis>(992)] [-test <Index>|<ID>(992)]*
[-threads <Anzahl>(994)] [-userdir <Verzeichnis>(994)]
[-variable <Name>=<Wert>(994)]* [-verbose [<level>](994)]
<Testsuite>+

Ausführen eines Test durch einen QF-Test Daemon

Die folgenden Parameter steuern die Ausführung eines QF-Test Daemon-Tests:

qftest -batch -calldaemon(979) [-cleanup(979)] [-clearglobals(979)]
[-dbg(976)] [-java <Programm> (abgekündigt)(977)] [-noconsole(977)]
[-J<java-argument>]* [-daemonhost <Host>(980)]
[-daemonport <Port>(980)] [-exitcode-ignore-exception(981)]
[-exitcode-ignore-error(981)] [-exitcode-ignore-warning(981)]
[-ipv6(982)] [-keystore <Keystore-Datei>(983)]
[-keypass <Kennwort>(983)] [-nomessagewindow(984)]
[-ping(985)] [-options <Datei>(985)] [-runid [<ID>](989)]
[-runlogdir <Verzeichnis>(989)] [-runlog [<Datei>](989)]
[-startclean(991)] [-startsut(991)] [-stopclean(991)] [-stoprun(991)]
[-suitedir <Verzeichnis>(991)] [-systemdir <Verzeichnis>(992)]
[-terminate(992)] [-timeout <Millisekunden>(994)]
[-userdir <Verzeichnis>(994)] [-variable <Name>=<Wert>(994)]*

[-verbose [<level>](994)] <suite#test-case>

XML-Format bestehender Testsuiten ändern

qftest -batch -convertxml(979)
[-convertxml-indent <Anzahl>(980)]
[-convertxml-linelength <Anzahl>(980)]
[-convertxml-utf8 <true|false>(980)] (<Testsuite> |
<Verzeichnis>)+

Test-Dokumentation erstellen

Package oder Testfall Dokumentation kann für eine oder mehrere Testsuiten oder ganze Verzeichnisse in einem Durchgang erstellt werden. Dies wird in Kapitel 24(330) genauer beschrieben. Die Kommandozeilen Syntax lautet:
qftest -batch -gendoc(981) [-dbg(976)] [-java <Programm>
(abgekündigt)(977)] [-noconsole(977)] [-J<Java-Argument>]*
[-license <Datei>(983)] [-license-waitfor <Sekunden>(983)]
[-option <Name>=<Wert>(985)] [-options <Datei>(985)]
[-nomessagewindow(984)] [-pkgdoc <Verzeichnis>(985)]
[-pkgdoc-doctags(985)] [-pkgdoc-dependencies(985)]
[-pkgdoc-html <Verzeichnis>(985)] [-pkgdoc-includelocal(985)]
[-pkgdoc-nodeicons(985)] [-pkgdoc-passhtml(985)]
[-report-piechart(988)] [-report-include-suitename(988)]
[-pkgdoc-sortpackages(986)] [-pkgdoc-sortprocedures(986)]
[-pkgdoc-xml <Verzeichnis>(986)] [-sourcedir <Verzeichnis>(990)]
[-systemdir <Verzeichnis>(992)] [-testdoc <Verzeichnis>(992)]
[-testdoc-doctags(993)] [-testdoc-followcalls(993)]
[-testdoc-html <Verzeichnis>(993)] [-testdoc-nodeicons(993)]
[-testdoc-passhtml(993)] [-testdoc-sorttestcases(993)]
[-testdoc-sorttestsets(993)] [-testdoc-teststeps(994)]
[-testdoc-xml <Verzeichnis>(994)] (<Testsuite> |
<Verzeichnis>)+

Einen Report aus Protokollen erstellen

Um einen Report aus einem oder mehreren Protokollen oder ganzen Verzeichnissen zu erstellen verwenden Sie: qftest -batch -genreport(981) [-dbg(976)] [-java <Programm>
(abgekündigt)(977)] [-noconsole(977)] [-J<Java-Argument>]*
[-license <Datei>(983)] [-license-waitfor <Sekunden>(983)]
[-nomessagewindow(984)] [-option <Name>=<Wert>(985)]
[-options <Datei>(985)] [-report <Verzeichnis>(986)]
[-report-checks(986)] [-report-customdir <Verzeichnis>(986)]
[-report-doctags(987)] [-report-errors(987)]
[-report-exceptions(987)] [-report-html <Verzeichnis>(987)]
[-report-ignorenotimplemented(987)] [-report-ignoreskipped(987)]

[-report-junit <Verzeichnis>(987)] [-report-name <Name>(987)]
[-report-nodeicons(987)] [-report-passhtml(987)]
[-report-piechart(988)] [-report-include-suitename(988)]
[-report-scale-thumbnails <Prozent>(988)]
[-report-teststeps(988)] [-report-thumbnails(988)]
[-report-warnings(988)] [-report-xml <Verzeichnis>(988)]
[-runlogdir <Verzeichnis>(989)] [-systemdir <Verzeichnis>(992)]
(<Protokoll> | <Verzeichnis>)+

Daemon-Modus

Um QF-Test im Daemon-Modus wie in Kapitel 55(1276) beschrieben zu starten verwenden Sie: qftest -batch -daemon(980) [-dbg(976)]
[-java <Programm> (abgekündigt)(977)] [-noconsole(977)]
[-J<Java-Argument>]* [-daemonhost <Host>(980)]
[-daemonport <Port>(980)] [-daemonrmiport <Port>(980)]
[-engine <Engine>(981)] [-groovydir <Verzeichnis>(981)]
[-ipv6(982)] [-javascriptdir <Verzeichnis>(982)]
[-jythondir <Verzeichnis>(982)] [-jythonport <Nummer>(982)]
[-keystore <Keystore-Datei>(983)] [-keypass <Kennwort>(983)]
[-libpath <Pfad>(983)] [-license <Datei>(983)]
[-license-waitfor <Sekunden>(983)] [-logdir <Verzeichnis>(983)]
[-nolog(984)] [-nomessagewindow(984)] [-noplugins(984)]
[-option <Name>=<Wert>(985)] [-options <Datei>(985)]
[-plugindir <Verzeichnis>(986)] [-port <Nummer>(986)]
[-runtime(989)] [-serverhost <Host>(990)] [-shell <Programm>(990)]
[-shellarg <Argument>(990)] [-systemcfg <Datei>(992)]
[-systemdir <Verzeichnis>(992)] [-usercfg <Datei>(994)]
[-variable <Name>=<Wert>(994)]*

Eine Testsuite in eine andere importieren

qftest -batch -import(982) [-import-from <Testsuite>(982)]
[-import-into <Testsuite>(982)] [-import-components(982)]
[-import-procedures(982)] [-import-tests(982)]

Referenzen einer Testsuite analysieren

qftest -batch -analyze(978)
[-analyze-target <Verzeichnis>(978)]
[-suitedir <Verzeichnis>(991)] [-analyze-references(978)]
[-analyze-duplicates(978)] [-analyze-invalidchar(978)]
[-analyze-emptynodes(978)] [-analyze-components(978)]
[-analyze-procedures(978)] [-analyze-dependencies(978)]
[-analyze-tests(978)] [-analyze-packages(979)]
[-remove-unused-callables(979)] [-remove-unused-components(979)]

```
[-analyze-transitive(979)] [-analyze-followincludes(979)]
(<Testsuite> | <Verzeichnis>)+
```

Protokolle zusammenführen

```
  qftest -batch -mergelogs(983) [-mergelogs-mode [<Modus>](984)]
[-mergelogs-usefqn(984)] [-mergelogs-resultlog [<Datei>](984)]
[-mergelogs-masterlog [<Datei>](983)] (<Protokoll> |
<Verzeichnis>)+
```

Versionsinformationen ausgeben

```
qftest -batch -version(995)
```

Beendet eine spezifische QF-Test Instanz auf diesem System (siehe `-allow-shutdown [<Shutdown-ID>]`(978))

```
qftest -batch -shutdown <ID>(990) 4711
```

Pausiert den Testlauf auf diesem System

```
                qftest -batch -interrupt-running-instances(982)
[-timeout <Millisekunden>(994)]
```

Abbilder in einer bestehenden Testsuite komprimieren

```
qftest -batch -compress(979) <suite>+
```

Hilfe anfordern

```
qftest -batch -help(981)
```

44.2 Kommandozeilenargumente

Kommandozeilenargumente für QF-Test fallen in drei Kategorien, wobei es nicht auf die Reihenfolge ankommt.

44.2.1 Argumente für das Startskript

Diese Argumente werden direkt vom `qftest` Skript bzw. Programm ausgewertet und heben die Werte auf, die während der Installation von QF-Test ermittelt wurden. Unter Linux stehen diese in der Datei `launcher.cfg` in QF-Tests Systemverzeichnis, unter Windows heißt die Datei `launcherwin.cfg`.

`-batch`

Startet QF-Test im Batchmodus. Dadurch lädt QF-Test eine Testsuite, führt sie direkt aus und beendet sich mit einem Rückgabewert, der das Ergebnis des Testlaufs widerspiegelt.

-dbg
Schaltet Debug-Ausgaben für das Startskript ein. Den selben Effekt erreichen Sie, wenn Sie die Umgebungsvariable QFTEST_DEBUG auf einen nicht leeren Wert setzen. Sofern nicht zusätzlich -noconsole[977] angegeben ist, öffnet QF-Test unter Windows in diesem Fall ein Konsolenfenster, da die Ausgaben ansonsten nicht sichtbar wären. Falls der alte Verbindungsmechanismus zum SUT eingesetzt wird, schaltet diese Option gleichzeitig Debug-Ausgaben für das qfclient Programm und QF-Tests java Hülle ein (vgl. Kapitel 46[1000]).

-java <Programm> (abgekündigt)
Das Java-Programm, das die Ausführung von QF-Test übernimmt. Standard sind java unter Linux und javaw.exe unter Windows, sofern während der Installation nichts anderes angegeben wurde. Dieses Argument wird in einer zukünftigen Version von QF-Test entfernt.

-noconsole (nur Windows)
Verhindert das Öffnen eines Konsolenfensters unter Windows für den Fall, dass -dbg[976] ebenfalls angegeben ist.

44.2.2 Argumente für die Java-VM

Sie können durch das Startskript Argumente an die Java-VM weiterreichen, indem Sie diesen -J voranstellen, z.B. -J-Duser.language=en, um eine Systemvariable zu setzen. Um den classpath anzugeben, fügen Sie -J nur dem -cp oder -classpath Argument zu, nicht dem eigentlichen Wert, z.B. -J-classpath myclasses.jar. Auch wenn Sie den classpath auf diese Weise angeben, brauchen Sie QF-Tests eigene jar Archive nicht zu berücksichtigen.

44.2.3 Argumente für QF-Test

Die restlichen Argumente werden von QF-Test selbst ausgewertet, wenn es von der Java-VM ausgeführt wird. Diese Argumente können auch in eine Datei geschrieben werden. Das Format lautet <Name>=<Wert> für Argumente mit Parametern und <Name>=true oder <Name>=false, um ein einfaches Argument ein- oder auszuschalten. Normalerweise heißt diese Datei qftest.options, liegt im bin Verzeichnis von QF-Test und wird nur für interne Zwecke verwendet. Wenn Sie Änderungen an dieser Datei vornehmen, ist es sinnvoll, sie in das Systemverzeichnis von QF-Test zu kopieren, da diese Änderungen dadurch auch nach einem Update gültig bleiben. Alternativ können Sie über das -options <Datei>[985] Argument eine andere Datei angeben. Argumente auf der Kommandozeile haben Vorrang vor Argumenten aus der Datei. Für Argumente, die mehrfach angegeben werden können,

werden sowohl Angaben von der Kommandozeile, als auch aus der Datei übernommen.

-allowkilling
Abgekündigt, verwenden Sie stattdessen `-allow-shutdown` ohne Shutdown-ID.

-allow-shutdown [<Shutdown-ID>]
Erlaubt explizit, dass diese QF-Test Instanz über einen Batch-Aufruf mit dem Argument `-shutdown <ID>`(990) kontrolliert beendet werden darf. Optional kann als Argument eine Zeichenkette als Shutdown-ID angegeben werden, welche ein individuelles Beenden von QF-Test Prozessen ohne die Kenntnis deren Prozess-ID erlaubt. Die Shutdown-ID muss dabei mindestens ein nicht-numerisches Zeichen enthalten, damit sie von der rein numerischen Prozess-ID unterschieden werden kann. Wurde `-allow-shutdown [<Shutdown-ID>]` nicht angegeben, so kann die QF-Test Instanz nur mit Hilfe der Prozess-ID kontrolliert beendet werden. Das Argument `-allow-shutdown false` verbietet grundsätzlich das kontrollierte Beenden dieser QF-Test Instanz, selbst wenn `-shutdown <ID>`(990) mit der korrekten Prozess-ID aufgerufen wird.

-analyze (nur Batchmodus)
Modus, um statische Analyse von Testsuiten vorzunehmen. Das Ergebnis wird für Analyseschritte in eine angegebene Datei geschrieben.

-analyze-target <Verzeichnis> (nur Batchmodus)
Das Zielverzeichnis, in welches die Ergebnisdatei gespeichert wird.

-analyze-references (nur Batchmodus)
Schalter, um Referenzen zu analysieren.

-analyze-duplicates (nur Batchmodus)
Schalter, um Duplikate zu analysieren.

-analyze-invalidchar (nur Batchmodus)
Schalter, um auf ungültige Zeichen in Knoten zu prüfen.

-analyze-emptynodes (nur Batchmodus)
Schalter, um auf leere Knoten zu prüfen.

-analyze-components (nur Batchmodus)
Schalter, um Komponentenverweise zu analysieren.

-analyze-dependencies (nur Batchmodus)
Schalter, um Verweise auf Abhängigkeiten zu analysieren.

-analyze-procedures (nur Batchmodus)
Schalter, um Aufrufe von Prozeduren zu analysieren.

`-analyze-tests` (nur Batchmodus)
Schalter, um Aufrufe von Testfällen zu analysieren.

`-analyze-packages` (nur Batchmodus)
Schalter, um Packages zu analysieren.

`-remove-unused-callables` (nur Batchmodus)
Schalter, um ungenutzte aufrufbare Knoten zu löschen.

`-remove-unused-components` (nur Batchmodus)
Schalter, um ungenutzte Komponenten zu löschen.

`-analyze-transitive` (nur Batchmodus)
Schalter, um die gefundenen Referenzen transitiv zu analysieren, d.h. auch deren Inhalt und Verweise zu prüfen.

`-analyze-followincludes` (nur Batchmodus)
Schalter, um alle inkludierten Testsuiten der angegebenen Testsuite zu analysieren.

`-calldaemon` (nur Batchmodus)
Verbinden mit einem laufenden QF-Test Daemon, um einen Test(fall) auszuführen.

`-cleanup` (nur Calldaemon-Modus)
Mit diesem Argument werden alle TestRunDaemon-Objekte einer Daemon-Instanz aufgeräumt und alle Clients beendet bevor der Test ausgeführt wird.

`-clearglobals` (nur Batchmodus und Calldaemon-Modus)
Sind mehrere Testsuiten zur Ausführung im Batchmodus angegeben, führt dieses Argument dazu, dass vor der Ausführung jeder Testsuite die globalen Variablen und Ressourcen gelöscht werden. Es kann außerdem im Calldaemon-Modus verwendet werden, um die globalen Variablen vor Setzen der Kommandozeilen-Variablen und Start des Tests zu löschen und, in Kombination mit `-stopclean`(991), zum Löschen der globalen Variablen am Ende des Tests.

`-compact` (nur Batchmodus)
Erstellt ein kompaktes Protokoll, das nur die Äste und Knoten enthält, in denen Warnungen, Fehler, Exceptions oder für den Report relevante Informationen enthalten sind - analog zum Aktivieren der Option Kompakte Protokolle erstellen(590) im interaktiven Modus. Hat keine Bedeutung, wenn das Protokoll mittels `-nolog`(984) unterdrückt wird.

`-compress` (nur Batchmodus)
Komprimiert die Abbilder in einer bestehenden Testsuite verlustfrei.

`-convertxml` (nur Batchmodus)
Ausführung im Batchmodus zur automatischen Konvertierung des XML-Formats von Testsuiten gemäß nachfolgender Argumente bzw. Optionen.

`-convertxml-indent <Anzahl>` (Nur im Modus zur Konvertierung des XML-Formats)
Anzahl von Leerzeichen pro Einrückungsstufe. Falls nicht angegeben, wird der Wert der Option Anzahl der Leerzeichen für das Einrücken beim Speichern von Testsuiten[489] verwendet.

`-convertxml-linelength <Anzahl>` (Nur im Modus zur Konvertierung des XML-Formats)
Maximale Länge von Zeilen mit XML-Attributen. Falls nicht angegeben, wird der Wert der Option Zeilenlänge beim Speichern von Testsuiten[490] verwendet.

`-convertxml-utf8 <true|false>` (Nur im Modus zur Konvertierung des XML-Formats)
Entscheidet, ob die konvertierten Testsuite mit UTF-8 (Wert true) oder ISO-8859-1 (Wert false) kodiert gespeichert werden. Falls nicht angegeben, wird der Wert der Option Testsuiten UTF-8 kodiert speichern[489] verwendet.

`-daemon`
Startet QF-Test im Daemon-Modus. Weitere Informationen finden Sie in Kapitel 55[1276].

`-daemonhost <Host>` (nur Daemon-Modus oder Calldaemon-Modus)
Legt im Calldaemon-Modus den Rechner fest, auf dem nach einem laufenden QF-Test Daemon gesucht wird (Standard ist localhost). Beim Start des Daemon - interaktiv oder im Batchmodus - definiert dieser Parameter den Hostnamen oder die IP-Adresse, welche Daemon-Objekte auf RMI Ebene nutzen. Der Standard in diesem Fall wird von Java bestimmt, typischerweise die IP-Adresse der primären Netzwerkschnittstelle des Rechners.

`-daemonport <Port>`
Legt den Registry Port für den QF-Test Daemon fest. Standard ist 3543 oder der mittels `-port <Nummer>`[986] festgelegte Port.

`-daemonrmiport <Port>`
Legt den Port für die RMI Kommunikation des QF-Test Daemon fest. Nur sinnvoll, wenn der Daemon hinter einer Firewall betrieben wird. Beim ungeschützten Betrieb ohne SSL (vgl. Abschnitt 55.3[1294]) kann dies der selbe Port sein wie beim Argument `-daemonport <Port>`[980]. Wird SSL verwendet, sind zwei verschiedene Ports erforderlich.

`-dontkillprocesses` (nur Batchmodus)
Ist dieses Argument angegeben, beendet QF-Test vor dem Ende des Batchlaufs

die im Verlauf der Tests gestarteten Prozesse nicht explizit. Ob diese Unterprozesse das Ende von QF-Test überleben ist allerdings Systemabhängig.

`-engine <Engine>`
Legt fest, welche Engine-Lizenzen verwendet werden. Diese Option ist nur sinnvoll, wenn Ihre QF-Test Lizenz eine Mischung von Engine-Lizenzen in unterschiedlicher Anzahl unterstützt. In diesem Fall kann es nötig sein, die benötigte GUI-Engine zu spezifizieren, um so Lizenz-Konflikte mit Kollegen zu vermeiden, welche die selbe Lizenz nutzen. Mögliche Werte sind ”all” für alle unterstützten Lizenzen, ”ask” um einen Dialog zur Auswahl der Engine anzuzeigen sowie eine beliebige Kombination von ”awt” für AWT/Swing, ”fx” für JavaFX, ”swt” für SWT oder ”web” für Web, z.B. ”awt,web”. Weitere Informationen hierzu finden Sie in Abschnitt 41.1.9(505).

`-exitcode-ignore-exception` (Batch- und Calldaemon-Modus)
Exceptions, Fehler und Warnungen werden bei der Berechnung des Rückgabewertes von QF-Test ignoriert. Das bedeutet, dass bei einem Testlauf mit Exceptions, Fehlern oder Warnungen 0 geliefert wird. Diese Option ist hilfreich für die Integration von QF-Test mit Build-Tools, die abhängig vom Rückgabewert den Build als fehlgeschlagen interpretieren.

`-exitcode-ignore-error` (Batch- und Calldaemon-Modus)
Fehler und Warnungen werden bei der Berechnung des Rückgabewertes von QF-Test ignoriert. Das bedeutet, dass bei einem Testlauf mit nur Fehlern oder Warnungen 0 geliefert wird. Diese Option ist hilfreich für die Integration von QF-Test mit Build-Tools, die abhängig vom Rückgabewert den Build als fehlgeschlagen interpretieren.

`-exitcode-ignore-warning` (Batch- und Calldaemon-Modus)
Warnungen werden bei der Berechnung des Rückgabewertes von QF-Test ignoriert. Das bedeutet, dass bei einem Testlauf mit nur Warnungen 0 geliefert wird. Diese Option ist hilfreich für die Integration von QF-Test mit Build-Tools, die abhängig vom Rückgabewert den Build als fehlgeschlagen interpretieren.

`-gendoc` (nur Batchmodus)
Teilt QF-Test mit, dass dieser Batch-Lauf zur Erstellung von Test-Dokumentation aus Testsuiten dient.

`-genreport` (nur Batchmodus)
Teilt QF-Test mit, dass dieser Batch-Lauf zur Erstellung von Reports aus Protokollen dient.

`-groovydir <Verzeichnis>`
Mit diesem Argument kann das Verzeichnis für zusätzliche Groovy-Module

überschrieben werden. Dieses Verzeichnis heißt normalerweise `groovy` und liegt im Systemverzeichnis von QF-Test.

`-help`
Zeigt Hilfe zu den verfügbaren Kommandozeilenargumenten an.

`-import` (nur Batchmodus)
Importiert eine Testsuite in eine andere.

`-import-from <Testsuite>` (nur Batchmodus)
Die Testsuite, welche importiert werden soll.

`-import-into <Testsuite>` (nur Batchmodus)
Die Zieltestsuite, in die importiert werden soll.

`-import-components` (nur Batchmodus)
Schalter, um Komponenten zu importieren.

`-import-procedures` (nur Batchmodus)
Schalter, um Packages und Prozeduren zu importieren.

`-import-tests` (nur Batchmodus)
Schalter, um Testfallsätze und Testfälle zu importieren.

`-interrupt-running-instances` (nur Batchmodus)
Unterbricht den aktuellen Testlauf auf dem lokalen System und zeigt einen Dialog, der die Pausierung bzw. das Stoppen des Testlaufes erlaubt.

`-ipv6`
QF-Test kommuniziert normalerweise nur via IPv4, daher ist die IPv6-Unterstützung standardmäßig auf Java Ebene deaktiviert, was die Startzeit spürbar verkürzen kann. Fall Sie Unterstützung für IPv6 in QF-Test benötigen, z.B. in einem Plugin, verwenden Sie dieses Argument um diese zu aktivieren.

`-javascriptdir <Verzeichnis>`
Mit diesem Argument kann das Verzeichnis für zusätzliche JavaScript-Module überschrieben werden. Dieses Verzeichnis heißt normalerweise `javascript` und liegt im Systemverzeichnis von QF-Test.

`-jythondir <Verzeichnis>`
Mit diesem Argument kann das Verzeichnis für zusätzliche Jython-Module überschrieben werden. Dieses Verzeichnis heißt normalerweise `jython` und liegt im Systemverzeichnis von QF-Test.

`-jythonport <Nummer>`
Weist den eingebetteten Jython-Interpreter an, auf der angegebenen Portnummer auf TCP Verbindungen zu lauschen. Sie können dann `telnet`

verwenden, um sich mit diesem Port zu verbinden und eine interaktive Jython Kommandozeile zu erhalten.

-keybindings <Wert> (nur interaktiven Modus)
Aktuell nur unter macOS interessant, um zwischen den neuen Mac Standard-Kürzeln (Wert system) oder den alten, an Windows orientierten QF-Test Kürzeln (Wert classic) umzuschalten.

-keystore <Keystore-Datei>
Eine alternative Keystore-Datei zur Absicherung der Kommunikation mit dem Daemon über SSL. Details hierzu finden Sie in Abschnitt 55.3(1294). Um SSL zu deaktivieren, indem Sie keine Keystore-Datei angeben, verwenden Sie dieses Argument in der Form `-keystore=`.

-keypass <Kennwort>
Das Passwort für die Keystore-Datei, die zur Absicherung der Kommunikation mit dem Daemon über SSL verwendet wird. Details hierzu finden Sie in Abschnitt 55.3(1294).

-kill-running-instances
Abgekündigt, verwenden Sie stattdessen `-shutdown all`.

-libpath <Pfad>
Dieser Wert entspricht der Option Verzeichnisse mit Testsuite-Bibliotheken(503) und setzt diese außer Kraft. Die Verzeichnisse des Bibliothekspfads sollten durch das dem System entsprechende Trennzeichen getrennt werden, d.h. ';' für Windows und ':' für Linux. Das `include` Verzeichnis von QF-Test wird automatisch an das Ende des Bibliothekspfads gestellt.

-license <Datei>
Gibt den Ort der Lizenzdatei an, falls dieser vom Standard abweicht (vgl. Abschnitt 1.5(11)).

-license-waitfor <Sekunden>;
Legt eine Zeitspanne in Sekunden fest, die beim Start von QF-Test gewartet wird, falls gerade keine Lizenz verfügbar ist. Diese Zeitspanne findet auch Anwendung, wenn bei der Verlängerung einer Lease vom QF-Test Lizenzserver dieser temporär nicht verfügbar ist.

-logdir <Verzeichnis>
Mit diesem Argument kann der Ort für das Verzeichnis für überschrieben werden, in dem QF-Test seine internen Protokolle ablegt. Dieses Verzeichnis heißt normalerweise `log` und liegt im Systemverzeichnis von QF-Test.

-mergelogs (nur Batchmodus)
Teilt QF-Test mit, dass dieser Batch-Lauf zum Zusammenführen von Protokollen dient. Details hierzu finden Sie unter Abschnitt 7.1.9[145].

-mergelogs-masterlog [<Datei>] (nur Batchmodus)
Der Pfad zum Hauptprotokoll, welches beim Zusammenführen von Protokollen verwendet wird. Dieses Protokoll beinhaltet den gesamten Testlauf. Einzelne enthaltene Testfälle können mit den Ergebnissen des Nachtests ersetzt werden.

-mergelogs-mode [<Modus>] (nur Batchmodus)
Diese Option gibt den Modus an, wie Ergebnisse von Testfällen aus den neuen Protokollen in das Hauptprotokoll, das mit -mergelogs-masterlog [<Datei>][983] spezifiziert wurde, eingegliedert werden sollen. Hier stehen die Werte ”replace”, ”merge” und ”append” zur Verfügung. Mit ”replace” werden die bestehenden Testfälle ersetzt, mit ”merge” werden die neuen Ergebnisse hinzugefügt und mit ”append” wird das neue Protokoll an das Hauptprotokoll angehängt.

-mergelogs-resultlog [<Datei>] (nur Batchmodus)
Der Pfad zu einem Protokoll, welches nach der Zusammenführung von Protokollen das Ergebnis der Zusammenführung beinhaltet.

-mergelogs-usefqn (nur Batchmodus)
Diese Option gibt an, ob beim Zusammenführen von Protokollen auch die Testfallsatzhierarchie eines Testfalles miteinbezogen werden soll. Ohne diese Option wird nur der Name des Testfalles für das Zusammenführen herangezogen.

-nolog (nur Batchmodus)
Verhindert die automatische Erstellung eines Protokolls. Falls -runlog [<Datei>][989], -report <Verzeichnis>[986], -report-html <Verzeichnis>[987], -report-xml <Verzeichnis>[988] oder -report-junit <Verzeichnis>[987] angegeben ist, wird dieses Argument ignoriert. Diese Option ist nur noch aus Gründen der Rückwärtskompatibilität vorhanden. Um den Speicherverbrauch im Griff zu halten sollten geteilte Protokolle verwendet werden (vgl. -splitlog[990]).

-nomessagewindow (nur Batchmodus)
Wenn schwere Fehler im Batchmodus auftreten, gibt QF-Test eine Fehlermeldung in der Konsole aus und öffnet zur besseren Sichtbarkeit zusätzlich für ca. 30 Sekunden einen Fehlerdialog. Das Öffnen des Dialogs kann mit diesem Argument verhindert werden. Batch Kommandos die keine Anzeige benötigen, also alle Kommandos, die keine Tests ausführen, laufen im AWT Headless Modus falls dieses Argument angegeben wird.

-noplugins
Laden von Plugins in QF-Test und Clients verhindern. Dies kann nützlich sein, um Problemen mit Plugins auf die Spur zu kommen.

-noupdatecheck
Mit diesem Argument kann die automatische Suche nach Updates deaktiviert werden. Hierdurch werden die Update-Optionen (siehe Abschnitt 41.1.10(506)) überschrieben.

-option <Name>=<Wert>
Setzt Optionen. Mit `-option <Name>=<Wert>` wird die Option mit dem Name `<Name>` auf den Wert `<Wert>` gesetzt. Dieses Argument kann mehrfach angegeben werden, um mehrere Optionen zu setzen.

-options <Datei>
Legt die Datei fest, aus der weitere Argumente gelesen werden. Dieses Argument kann mehrfach angegeben werden, um Argumente aus mehreren Dateien zu lesen.

-ping (nur Calldaemon-Modus)
Prüft, ob ein Daemon erreichbar ist.

-pkgdoc <Verzeichnis> (nur Batchmodus)
Mit diesem Argument erstellt QF-Test HTML- und XML-Pkgdoc-Dokumentationen. Ist kein Verzeichnis angegeben wird der Name aus dem Namen der Testsuite gebildet.

-pkgdoc-dependencies (nur Batchmodus)
Legt fest, ob Abhängigkeiten in pgkdoc Dokumenten aufgeführt werden. Standard ist ja, mit `-pkgdoc-doctags=false` können diese ausgeschaltet werden.

-pkgdoc-doctags (nur Batchmodus)
Legt fest, ob die doctags Erweiterungen von QFS verwendet werden. Standard ist ja, mit `-pkgdoc-doctags=false` können die doctags ausgeschaltet werden.

-pkgdoc-html <Verzeichnis> (nur Batchmodus)
Mit diesem Argument erstellt QF-Test eine HTML-Pkgdoc-Dokumentation. Ist kein Verzeichnis angegeben, wird der Name aus dem Namen der Testsuite gebildet.

-pkgdoc-includelocal (nur Batchmodus)
Legt fest, ob lokale Packages und Prozeduren, d.h. solche, deren Name mit einem '_' beginnt, einbezogen werden. Standard ist nein.

-pkgdoc-nodeicons (nur Batchmodus)
Legt fest, ob Icons für Knoten in der Pkgdoc-Dokumentation angezeigt werden.

Standard ist ja, mit `-pkgdoc-nodeicons=false` können die Icons ausgeschaltet werden.

`-pkgdoc-passhtml` (nur Batchmodus)
Legt fest, ob HTML Tags in Kommentaren unverändert an die HTML-Dokumentation durchgereicht werden. Standard ist ja, mit `-pkgdoc-passhtml=false` können die Tags ausgeschaltet werden.

`-pkgdoc-sortpackages` (nur Batchmodus)
Legt fest, ob Packages alphabetisch sortiert werden. Standard ist ja, mit `-pkgdoc-sortpackages=false` kann die Sortierung ausgeschaltet werden.

`-pkgdoc-sortprocedures` (nur Batchmodus)
Legt fest, ob Prozeduren alphabetisch sortiert werden. Standard ist ja, mit `-pkgdoc-sortprocedures=false` kann die Sortierung ausgeschaltet werden.

`-pkgdoc-splitparagraph` (nur Batchmodus)
Legt fest, ob Kommentare an Leerzeilen in Absätze aufgeteilt werden. Standard ist ja, mit `-pkgdoc-splitparagraph=false` können Sie die Option ausschalten.

`-pkgdoc-stylesheet <Datei>` (nur Batchmodus)
Optionales XSLT Stylesheet für die zweite Stufe der Transformation.

`-pkgdoc-xml <Verzeichnis>` (nur Batchmodus)
Mit diesem Argument erstellt QF-Test eine XML-Pkgdoc-Dokumentation. Ist kein Verzeichnis angegeben, wird der Name aus dem Namen der Testsuite gebildet.

`-plugindir <Verzeichnis>`
Mit diesem Argument kann der Ort für das Plugin Verzeichnis für jar Dateien, auf die per Skript zugegriffen werden soll, überschrieben werden. Dieses Verzeichnis heißt normalerweise `plugin` und liegt im Systemverzeichnis von QF-Test. Weitere Informationen über Plugins finden Sie in Abschnitt 50.2(1029).

`-port <Nummer>`
Der TCP-Port auf dem QF-Test mit dem SUT kommuniziert. Normalerweise sucht sich QF-Test einen freien dynamischen Port, um dort seine eigene RMI-Registry anzulegen. Ein spezieller Port sollte nur dann angegeben werden, wenn es erforderlich ist, um das SUT zu starten.

`-report <Verzeichnis>` (nur Batchmodus)
Erstellt einen kombinierten XML/HTML-Report. Im Verzeichnisnamen können Platzhalter, wie in Abschnitt 44.2.4(995) beschrieben, angegeben werden.

`-report-checks` (nur Batchmodus)
Legt fest, ob Checks im Report aufgelistet werden. Standard ist nein. Bitte beachten Sie, dass dies nur für Checks greift, für deren Ergebnisbehandlung weder eine Variable gesetzt noch eine Exception geworfen wird. Weitere Informationen finden sie unter Abschnitt 24.1.2(333).

`-report-customdir <Verzeichnis>` (nur Batchmodus)
Verzeichnis für eigene CSS-Stylesheets und Icons für angepasste Reports.

`-report-doctags` (nur Batchmodus)
Legt fest, ob die doctags Erweiterungen von QFS verwendet werden. Standard ist ja, mit `-report-doctags=false` können die doctags ausgeschaltet werden.

`-report-errors` (nur Batchmodus)
Legt fest, ob Fehler im Report aufgelistet werden. Standard ist ja, mit `-report-errors=false` können die Fehler ausgeschaltet werden.

`-report-exceptions` (nur Batchmodus)
Legt fest, ob Exceptions im Report aufgelistet werden. Standard ist ja, mit `-report-exceptions=false` können die Exceptions ausgeschaltet werden.

`-report-html <Verzeichnis>` (nur Batchmodus)
Erstellt einen HTML-Report. Im Verzeichnisnamen können Platzhalter, wie in Abschnitt 44.2.4(995) beschrieben, angegeben werden.

`-report-ignorenotimplemented` (nur Batchmodus)
Legt fest, ob nicht implementierte Knoten im Report ignoriert werden sollen. In diesem Fall werden auch die Legende und die Spalten für nicht implementierte Tests nicht angezeigt. Standard ist nein, d.h. nicht implementierte Tests werden angezeigt.

`-report-ignoreskipped` (nur Batchmodus)
Legt fest, ob übersprungene Knoten im Report ignoriert werden sollen. In diesem Fall werden auch die Legende und die Spalten für übersprungene Tests nicht angezeigt. Standard ist nein, d.h. übersprungene Tests werden angezeigt.

`-report-junit <Verzeichnis>` (nur Batchmodus)
Erstellt einen Report im JUnit XML-Format, wie ihn viele Continuous Integration Tools verstehen. Im Verzeichnisnamen können Platzhalter, wie in Abschnitt 44.2.4(995) beschrieben, angegeben werden.

`-report-name <Name>` (nur Batchmodus)
Legt den Reportnamen fest (nicht den Dateinamen). Standard ist die Runid. Im Reportnamen können Platzhalter, wie in Abschnitt 44.2.4(995) beschrieben, angegeben werden.

-report-nodeicons (nur Batchmodus)
Legt fest, ob Icons für Knoten im Report angezeigt werden. Standard ist ja, mit `-report-nodeicons=false` können die Icons ausgeschaltet werden.

-report-passhtml (nur Batchmodus)
Legt fest, ob HTML-Tags in Kommentaren unverändert an den HTML-Report durchgereicht werden. Standard ist ja, mit `-report-passhtml=false` können die Tags ausgeschaltet werden.

-report-piechart (nur Batchmodus)
Legt fest, ob im Kopfbereich des HTML Reports ein Tortendiagramm angezeigt werden soll. Standard ist ja, mit `-report-piechart=false` kann das Erzeugen des Diagramms verhindert werden.

-report-include-suitename (nur Batchmodus)
Legt fest, ob für den Bezeichner einer Testsuite im HTML-Report der Wert des Name(596)-Attributs des Testsuite(595)-Knotens verwendet werden soll. Standard ist ja, mit `-report-include-suitename=false` wird der Dateiname verwendet.

-report-scale-thumbnails <Prozent> (nur Batchmodus)
Bestimmt die Skalierung von Miniaturbildern für Screenshots in den Fehlerübersichten von Reports. Ein einfacher Integer-Wert wird als Prozentangabe relativ zur Originalgröße interpretiert. Seit QF-Test Version 9.0 ist die bevorzugte Alternative ein kombinierter Wert in der Form <Breite>x<Höhe>. Hierbei werden Bilder proportional so skaliert, dass weder die angegebene Breite noch die Höhe überschritten werden. Der Standardwert ist 140x70.

-report-teststeps (nur Batchmodus)
Legt fest, ob Testschritte im Report aufgelistet werden. Standard ist ja, mit `-report-teststeps=false` können die Testschritte ausgeschaltet werden.

-report-thumbnails (nur Batchmodus)
Legt fest, ob Miniaturbilder für Screenshots in den Fehlerübersichten des Reports dargestellt werden. Standard ist nein.

-report-warnings (nur Batchmodus)
Legt fest, ob Warnungen im Report berücksichtigt werden. Standard ist ja.

-report-xml <Verzeichnis> (nur Batchmodus)
Erstellt einen XML-Report. Im Verzeichnisnamen können Platzhalter, wie in Abschnitt 44.2.4(995) beschrieben, angegeben werden.

-reuse (nur interaktiver Modus)
Dieses Argument kommt vor allem beim Start von QF-Test über ein Desktop Icon oder mittels einer Dateiverknüpfung aus dem Windows Explorer zum Tragen. Es veranlasst QF-Test zunächst nach bereits laufenden QF-Test Programmen zu

suchen und diese zu bitten, die angegebenen Dateien zu öffnen. Im Erfolgsfall wird die neu gestartete Version sofort wieder beendet und das bereits laufende Programm öffnet die Dateien in neuen Fenstern.

`-run` (interaktiver und Batchmodus)
Bei Verwendung im interaktiven Modus werden sofort nach dem Start die angegebene Testsuite bzw. die angegebenen Tests ausgeführt. Im Batchmodus teilt dieser Parameter QF-Test explizit mit, dass dieser Batch-Lauf zur Durchführung von Tests dient und nicht zur Erstellung von Test-Dokumentation oder Reports. Da dies der Standard-Fall ist kann dieses Kommandozeilenargument wegfallen.

`-runid [<ID>]` (Batch- und Calldaemon-Modus)
Legt die ID eines Testlaufs fest. In der ID können Platzhalter, wie in Abschnitt 44.2.4[995] beschrieben, angegeben werden. Sie dient selbst wiederum als Ersatz für den Platzhalter `%i`/`+i`.

`-runlog [<Datei>]` (Batch- und Calldaemon-Modus)
Schreibt das Protokoll in die angegebene Datei. Im optionalen Dateinamen können Platzhalter, wie in Abschnitt 44.2.4[995] beschrieben, angegeben werden. Ist keine Endung angegeben, wird automatisch `.qrz` angehängt und das Protokoll komprimiert geschrieben. Andernfalls entscheidet die Endung `.qrl` oder `.qrz` über die Kompression. Ist gar keine Datei angegeben, wird der Name aus dem Namen der Testsuite sowie dem aktuellen Datum und Uhrzeit gebildet. Ein Protokoll wird immer erstellt, sofern es nicht durch Angabe von `-nolog`[984] unterdrückt oder ein Report generiert wird. Im Calldaemon-Modus wird das Protokoll nur bei Angabe eines (lokalen) Dateinames gespeichert.

`-runlogdir <Verzeichnis>`
Im interaktiven Modus überschreibt dieses Argument die Option Verzeichnis für Protokolle[580] in einer speziellen Ebene für Optionen von der Kommandozeile, so dass das interaktive Ändern dieser Option keinen Effekt hat. Auf Skriptebene kann sie nach wie vor überschrieben werden. Im Batchmodus dient das Verzeichnis als Basis zum Speichern von Protokollen, sofern im Argument `-runlog [<Datei>]`[989] kein absoluter Pfad angegeben ist. Wird dieses Argument bei der Erstellung eines Reports angegeben, werden die Dateien im Report entsprechend der Struktur der Protokolle relativ zu diesem Verzeichnis angelegt. Im Verzeichnisnamen können Platzhalter, wie in Abschnitt 44.2.4[995] beschrieben, angegeben werden.

`-runtime`
Mit diesem Argument verwendet QF-Test ausschließlich Runtime Lizenzen. Im Batch Modus wird QF-Test normalerweise mit einer Runtime Lizenz gestartet (oder mehreren bei der Angabe von `-threads <Anzahl>`[994]). Sind nicht

ausreichend Runtime Lizenzen vorhanden, verwendet QF-Test stattdessen volle Entwicklerlizenzen. Dies wird durch die Angabe von `-runtime` verhindert, so dass sich QF-Test bei mangelnden Runtime Lizenzen mit einem Fehler beendet. Im interaktiven Modus verwendet QF-Test bei Angabe von `-runtime` eine Runtime statt einer Entwicklerlizenz. Damit können Tests normal oder unter Verwendung des Debuggers ausgeführt werden. Testsuite können allerdings nicht gespeichert werden, selbst wenn sie für experimentelle Tests verändert wurden.

`-shell <Programm>`
Das Shell Programm, das zur Ausführung eines Shell-Kommando ausführen(734) Knotens verwendet wird. Vorgegeben sind `/bin/sh` unter Linux und `COMMAND.COM` bzw. `cmd.exe` unter Windows.

`-shellarg <Argument>`
Das Argument, das der mittels `-shell <Programm>`(990) angegebenen Shell mitteilt, dass sie das darauf folgende Argument als Kommando ausführen soll. Standard für Linux Shells ist `-c`, während `COMMAND.COM` und `cmd.exe /c` erwarten. Wenn Sie Linux Tools unter Windows verwenden und z.B. `sh` oder `bash` als Shell angeben, dürfen Sie nicht vergessen, hier `-c` anzugeben.

`-shutdown <ID>` (nur Batchmodus)
Beendet kontrolliert die QF-Test Instanz mit der angegebenen Prozess-ID (nur Ziffern) oder Shutdown-ID auf dem aktuellen System, soweit dies erlaubt ist (siehe `-allow-shutdown [<Shutdown-ID>]`(978)). Im Batchmodus wird dabei der laufende Test abgebrochen, verbundene Clients werden beendet, das Protokoll wird geschrieben und der angegebene Prozess wird mit dem Exit-Code -12 terminiert. Bei einem interaktiven QF-Test werden zusätzlich alle Testsuiten ohne weitere Rückfrage geschlossen, etwaige Änderungen werden nicht gespeichert - dies kann in einigen Fällen dennoch nützlich sein, insbesondere wenn die automatische Speicherung sinnvoll konfiguriert ist (siehe Zeitabstand für automatische Speicherung (s)(502)). Die spezielle Shutdown-ID `all` beendet alle QF-Test Prozesse, die mit dem Argument `-allow-shutdown [<Shutdown-ID>]`(978) gestartet wurden, um dies explizit zu erlauben.

`-serverhost <Host>`
Legt den Hostnamen oder die IP-Adresse für die Kommunikation zwischen QF-Test und dem SUT fest. Eventuell müssen Sie diese angeben, wenn QF-Test und das SUT auf unterschiedlichen Rechnern laufen oder wenn Probleme mit der Namensauflösung auftreten. Ohne Angabe wird die Loopback Netzwerk-Schnittstelle verwendet. Um die primäre Netzwerk-Schnittstelle des lokalen Rechners zu verwenden, geben Sie `-serverhost=` mit leerem Wert an.

-sourcedir <Verzeichnis> (nur Batchmodus)
Wird dieses Argument bei der Erstellung eines Reports angegeben, werden die Dateien im Report entsprechend der Struktur der Testsuiten relativ zu diesem Verzeichnis angelegt, sofern nicht gleichzeitig -runlogdir <Verzeichnis>(989) angegeben wird. In jedem Fall wird das Verzeichnis einer Testsuite im Report nur dann aufgelistet, wenn dieses Argument angegeben ist und die Testsuite sich unterhalb dieses Verzeichnis befindet.

-splitlog (batch mode only)
Im Batchmodus sind geteilte Protokolle (vgl. Abschnitt 7.1.6(143)) standardmäßig aktiviert. Sie können durch Angabe von -splitlog=false ausgeschaltet werden. Wird -splitlog explizit ohne Parameter angegeben, ändert sich die Standard-Endung für Protokolle von .qrz in .qzp, um geteilte Protokolle im ZIP Format zu erzeugen. Unabhängig davon kann die Endung kann durch explizite Angabe beim Namen der Protokolls festgelegt werden.

-startclean (nur Calldaemon-Modus)
Mit diesem Argument werden alle Context-Objekte des gemeinsamen TestRunDaemons aufgeräumt bevor der Test ausgeführt wird.

-startsut (nur zum internen Gebrauch)
Dieses Argument wird zum Starten eines Clients auf einem entfernten Rechner verwendet. Sie sollten es nicht direkt verwenden, sondern ggf. auf die Prozedur qfs.daemon.startRemoteSUT aus der Standardbibliothek qfs.qft zurückgreifen.

-stopclean (nur Calldaemon-Modus)
Mit diesem Argument werden alle Context-Objekte des gemeinsamen TestRunDaemons aufgeräumt nachdem der Test ausgeführt wurde.

-stoprun (nur Calldaemon-Modus)
Stoppt ein laufenden Test auf dem Daemon mit dem angegebenen Hostnamen und Port. Dieses Argument kann mit -cleanup(979) oder -stopclean(991) kombiniert werden.

-suitedir <Verzeichnis> (nur Calldaemon-Modus)
Geben Sie ein Verzeichnis auf dem Daemon-Rechner an, wo der QF-Test Daemon nach Testsuiten sucht. Andernfalls kann beim auszuführenden Test auch ein absoluter Pfad angegeben werden.

-suitesfile <Datei> (interaktiver und Batchmodus)
Gibt eine Textdatei an, welche Testsuiten und ggf. Testfälle für die Ausführung beinhaltet. Hierzu können Sie pro Zeile einen Pfad zur Testsuite angeben. Einzelne Tests können wie beim Parameter -test <Index>|<ID> angegeben werden. In der unten stehenden Tabelle finden Sie weitere Beispiele.

Eintrag in Datei	Bedeutung
pfad/suite1.qft pfad/suite2.qft	Es werden die beiden Testsuiten ausgeführt.
pfad/suite1.qft pfad/suite2.qft#id-tc1	Es wird suite1.qft vollständig und der Testfall 'id-tc1' aus suite2.qft ausgeführt.
pfad/suite1.qft -test tc1 -test tc2	Es werden die Testfälle tc1 und tc2 aus suite1.qft ausgeführt.

Tabelle 44.1: Beispiele `-suitesfile <Datei>`

`-systemcfg <Datei>`
Legt die Konfigurationsdatei für Systemeinstellungen fest (siehe Abschnitt 1.6[12]).

`-systemdir <Verzeichnis>`
Überschreibt das Verzeichnis mit den systemspezifischen Konfigurationsdateien (siehe Abschnitt 1.6[12]) inklusive optionalen Plugins und Skript-Modulen. Falls die Argumente `-systemcfg <Datei>`[992], `-plugindir <Verzeichnis>`[986], `-jythondir <Verzeichnis>`[982], `-groovydir <Verzeichnis>`[981] oder `-javascriptdir <Verzeichnis>`[982] zusätzlich angegeben sind, haben diese Vorrang.

`-tempdir <Verzeichnis>` (nur interaktiver Modus)
Kann unter Windows benötigt werden, um temporäre Dateien für die kontextsensitive Hilfe anzulegen. Normalerweise werden die Umgebungsvariablen `TEMP` und `TMP` ausgewertet.

`-terminate` (nur Calldaemon-Modus)
Mit dieser Option wird der QF-Test Daemon (nach Ausführung des Tests) beendet.

`-test <Index>|<ID>` (interaktiver und Batchmodus)
Ohne diese Angabe werden die Tests der Suite einer nach dem anderen ausgeführt. Durch die Angabe von `-test <Index>|<ID>` können Sie gezielt einzelne Tests herauspicken. Einen beliebigen Knoten der Testsuite, der sich nicht auf der obersten Ebene befinden muss, können Sie über seine QF-Test ID[606] auswählen, einen Testfall[599] oder Testfallsatz[606] Knoten auch über seinen qualifizierten Namen. Die Tests auf der obersten Ebene sind zudem über einen numerischen Wert ansprechbar, wobei der erste Test den Index 0 hat. Sie können `-test <Index>|<ID>` beliebig oft angeben, auch mehrfach mit demselben Wert.

`-testdoc <Verzeichnis>` (nur Batchmodus)
Mit diesem Argument erstellt QF-Test HTML- und XML-Testdoc-Dokumentation.

Ist kein Verzeichnis angegeben wird der Name aus dem Namen der Testsuite gebildet.

-testdoc-doctags (nur Batchmodus)
Legt fest, ob die doctags Erweiterungen von QFS verwendet werden. Standard ist ja, mit `-testdoc-doctags=false` können die doctags ausgeschaltet werden.

-testdoc-followcalls (nur Batchmodus)
Normalerweise ignoriert QF-Test Testaufruf(614) Knoten bei der Testdoc-Generierung. Mit diesem Argument werden die referenzierten Ziele, also Testfall, Testfallsatz oder ganze Testsuite, so eingebunden, als wären sie Teil der Ausgangssuite. Hierdurch ist es möglich, partielle Testdoc Dokumente mit Hilfe einer dedizierten Testsuite zu erstellen, die Testaufrufe für die benötigten Teile enthält.

-testdoc-html <Verzeichnis> (nur Batchmodus)
Mit diesem Argument erstellt QF-Test eine HTML-Testdoc-Dokumentation. Ist kein Verzeichnis angegeben, wird der Name aus dem Namen der Testsuite gebildet.

-testdoc-nodeicons (nur Batchmodus)
Legt fest, ob Icons für Knoten in der Testdoc-Dokumentation angezeigt werden. Standard ist ja, mit `-testdoc-nodeicons=false` können die Icons ausgeschaltet werden.

-testdoc-passhtml (nur Batchmodus)
Legt fest, ob HTML-Tags in Kommentaren unverändert an die HTML-Dokumentation durchgereicht werden. Standard ist ja, mit `-testdoc-passhtml=false` können die Tags ausgeschaltet werden.

-testdoc-sorttestcases (nur Batchmodus)
Legt fest, ob Testfälle alphabetisch sortiert werden. Standard ist ja, mit `-testdoc-sorttestcases=false` kann die Sortierung ausgeschaltet werden.

-testdoc-sorttestsets (nur Batchmodus)
Legt fest, ob Testfallsätze alphabetisch sortiert werden. Standard ist ja, mit `-testdoc-sorttestsets=false` kann die Sortierung ausgeschaltet werden.

-testdoc-splitparagraph (nur Batchmodus)
Legt fest, ob Kommentare an Leerzeilen in Absätze aufgeteilt werden. Standard ist ja, mit `-testdoc-splitparagraph=false` können Sie die Option ausschalten.

-testdoc-stylesheet <Datei> (nur Batchmodus)
Optionales XSLT Stylesheet für die zweite Stufe der Transformation.

-testdoc-teststeps (nur Batchmodus)
Legt fest, ob Testschritte in der Testdoc-Dokumentation aufgelistet werden. Standard ist ja, mit `-testdoc-teststeps=false` können die Testschritte ausgeschaltet werden.

-testdoc-xml <Verzeichnis> (nur Batchmodus)
Mit diesem Argument erstellt QF-Test eine XML-Testdoc-Dokumentation. Ist kein Verzeichnis angegeben, wird der Name aus dem Namen der Testsuite gebildet.

-threads <Anzahl> (batch mode only)
Führt des selbe Testsuite in einer Anzahl von parallelen Threads zur Durchführung von Lasttests aus. Pro Thread wird eine Lizenz benötigt, daher sollte normalerweise das Argument `-runtime`(989) ebenfalls angegeben werden. Näheres zu Lasttests finden Sie in Kapitel 33(437).

-timeout <Millisekunden> (nur Batchmodus oder Calldaemon-Modus)
Maximale Dauer eines Tests, der im Batchmodus oder über den QF-Test Daemon ausgeführt wird (in Millisekunden, Vorgabe ist unendlich).

-usercfg <Datei> (nur interaktiver Modus)
Legt die Konfigurationsdatei für Benutzereinstellungen fest (siehe Abschnitt 1.6(12)).

-userdir <Verzeichnis>
Überschreibt das Verzeichnis mit den benutzerspezifischen Konfigurationsdateien (siehe Abschnitt 1.6(12)). Falls `-usercfg <Datei>`(994) oder `-runlogdir <Verzeichnis>`(989) zusätzlich angegeben sind, haben diese Vorrang.

-variable <Name>=<Wert>
Durch die Angabe von `-variable <Name>=<Wert>` geben Sie der Variable `<Name>` den Wert `<Wert>` (vgl. Kapitel 6(116)). Sie können beliebig viele Variablen definieren.

-verbose [<level>]
Ausgabe von Fortschritts- und Statusinformationen während eines Testlaufs auf der Konsole. Diese Option ist insbesondere dann sinnvoll, wenn der Test via `-calldaemon` auf einem anderen Rechner ausgeführt wird und daher die Ausführung u.U. nicht so einfach verfolgt werden kann. Unter Windows muss man allerdings `qftestc.exe` (anstelle von `qftest.exe`) verwenden, um die Ausgaben zu sehen. Die Angabe eines Levels ist optional, mögliche Werte sind `all` (alle Knoten ausgeben) und `tests` (Vorgabe; nur Testfallsatz und Testfall

Knoten werden ausgegeben). Jeder dieser Werte kann zusätzlich mit `errors` (Ausgabe von Fehler- und Exceptionmeldungen) kombiniert werden, etwa `tests,errors`.

`-version`
Hiermit gibt QF-Test Versionsinformationen aus und beendet sich dann.

44.2.4 Platzhalter im Dateinamen für Protokoll und Report

Im Dateinamen, der bei den Kommandozeilenargumenten `-runid [<ID>]`(989), `-runlog [<Datei>]`(989), `-runlogdir <Verzeichnis>`(989), `-report <Verzeichnis>`(986), `-report-html <Verzeichnis>`(987), `-report-name <Name>`(987), `-report-xml <Verzeichnis>`(988) oder `-report-junit <Verzeichnis>`(987) angegeben werden kann, können Platzhalter der Form %X oder +X verwendet werden (letzteres muss unter Windows verwendet werden, da % Zeichen dort besondere Bedeutung haben), wobei X für ein Zeichen aus der folgenden Tabelle steht. Wenn das Protokoll oder der Report erstellt werden, setzt QF-Test den entsprechenden Wert ein. Alle Zeitwerte beziehen sich auf die Startzeit des Testlaufs.

Hinweis Werden mehrere Testsuiten ausgeführt, sollten Sie auf jeden Fall den Namen der Suite als Teil des Namens für das Protokoll oder den Report verwenden, indem Sie %b angeben. Andernfalls wird eventuell nur ein Protokoll oder Report entsprechend der Ausführung der letzten Testsuite erstellt.

Zeichen	Bedeutung
%	'%'-Zeichen.
+	'+'-Zeichen.
i	Die aktuelle Runid wie mit `-runid [<ID>]`(989) angegeben.
p	Das Verzeichnis der Testsuite relativ zu `-sourcedir <Verzeichnis>`(990) oder absolut falls `-sourcedir <Verzeichnis>` nicht angegeben ist. Ist `-sourcedir <Verzeichnis>` angegeben, die Testsuite aber nicht darunter enthalten, ist dieser Wert leer.
P	Das absolute Verzeichnis der Testsuite. Darf nur am Anfang stehen.
b	Der Name der Testsuite ohne Verzeichnis oder die Endung `.qft`.
r	Der Rückgabewert des Testlaufs (nur `-runlog`).
w	Die Anzahl der Warnungen im Testlauf (nur `-runlog`).
e	Die Anzahl der Fehler im Testlauf (nur `-runlog`).
x	Die Anzahl der Exceptions im Testlauf (nur `-runlog`).
y	Das aktuelle Jahr (2 Ziffern).
Y	Das aktuelle Jahr (4 Ziffern).
M	Der aktuelle Monat (2 Ziffern).
d	Der aktuelle Tag (2 Ziffern).
h	Die aktuelle Stunde (2 Ziffern).
m	Die aktuelle Minute (2 Ziffern).
s	Die aktuelle Sekunde (2 Ziffern).

Tabelle 44.2: Platzhalter im Dateinamen Parameter

Wenn Sie also zum Beispiel das Protokoll in ein Verzeichnis namens `logs` unterhalb des Verzeichnisses der Testsuite schreiben und dabei einen Zeitstempel und den Rückgabewert erhalten wollen, verwenden Sie

```
-runlog %p/logs/%b-%y%M%d-%h%m%s-%r.qrl
```

Hinweis Die Platzhalter %b, %p und %P können auch für kollektive Parameter wir runid oder report verwendet werden. Wirklich Sinn macht das nur, falls nur eine einzelne Testsuite verarbeitet wird. Im Fall mehrerer Testsuiten wird für diesen Fall der Name der ersten Testsuite herangezogen.

44.3 Rückgabewerte von QF-Test

Wird QF-Test im interaktiven Modus ausgeführt, besitzt der Rückgabewert keine besondere Aussagekraft. Er ist entweder negativ, falls QF-Test nicht richtig startet, oder 0.

Im Batchmodus drückt der Rückgabewert dagegen das Ergebnis des Testlaufs aus. Ne-

gative Werte stehen für Fehler, die verhindern, dass der Test überhaupt gestartet wird, 0 für einen fehlerfreien Durchgang und positive Werte für Fehler während des Tests. Manche Systeme unterstützen nur Rückgabewerte zwischen 0 und 255. In diesem Fall sind alle Werte modulo 256 zu sehen, also -1=255, -2=254 etc.

Die folgenden Rückgabewerte sind zur Zeit definiert:

Wert	Bedeutung
0	Alles OK
1	Beim Testlauf sind Warnungen aufgetreten
2	Beim Testlauf sind Fehler aufgetreten
3	Beim Testlauf sind Exceptions aufgetreten
-1	Unerwartete Exception
-2	Falsche Kommandozeilenargumente
-3	Keine oder ungültige Lizenz
-4	Fehler beim Aufbau der RMI Verbindung
-5	Fehler beim Laden der Suite
-6	Die Suite enthält keine Tests
-12	Der Prozess wurde von außen über das `-shutdown <ID>`(990) Batch-Kommando beendet

Tabelle 44.3: Rückgabewerte von QF-Test

Daneben gibt es spezielle Rückgabewerte, wenn QF-Test mit dem Argument `-calldaemon` ausgeführt wird:

Wert	Bedeutung
-7	Der Daemon konnte nicht gefunden werden
-8	Fehler beim Erstellen eines TestRunDaemon-Objekts
-9	Fehler beim Erstellen eines Context-Objekts
-10	Der Test konnte nicht gestartet werden
-11	Der Test wurde innerhalb des angegeben Timeouts nicht beendet

Tabelle 44.4: `calldaemon`-Rückgabewerte von QF-Test

Kapitel 45

GUI-Engines

Swing, JavaFX und SWT können zusammen in einer Anwendung kombiniert werden, nicht nur in Form von getrennten Fenstern, sondern auch durch Einbettung von Komponenten der einen in Fenstern der anderen Art. QF-Test unterstützt das Testen solcher gemischten Anwendungen.

4+ Auch Webseiten können in Java-Anwendungen mit Hilfe von eingebetteten Browsern, wie der WebView Komponente bei JavaFX oder dem JxBrowser, integriert werden. QF-Test unterstützt diverse Kombinationen solcher hybriden Anwendungen.

Zu diesem Zweck wurde das Konzept einer *GUI-Engine* eingeführt. Eine GUI-Engine ist für Aufnahme und Wiedergabe in einem GUI Toolkit Thread zuständig. Normale Anwendungen haben nur einen solchen Thread. Wie oben beschrieben, sind aber auch Kombinationen von Swing, JavaFX und SWT möglich, die je einen eigenen Thread benötigen und somit auch zwei GUI Engines. Theoretisch ist es sogar möglich, mehrere GUI-Engines der selben Art zu haben, z.B. wenn mehrere Instanzen der SWT Display Klasse erzeugt werden.

Hinweis Die erste GUI-Engine, die in einem SUT erzeugt wird, heißt auch Default-Engine. Sie wird immer dann verwendet, wenn keine GUI-Engine explizit angegeben wird, insbesondere bei SUT-Skript[720] Knoten mit leerem GUI-Engine[722] Attribut.

Jede GUI-Engine wird in QF-Test über ein Kürzel für das GUI Toolkit und eine Zahl identifiziert. `awt0`, `fx0` und `swt0` sind die primären Engines für AWT/Swing, JavaFX und SWT. Wenn Sie nicht eine *sehr* spezielle Anwendung haben wird es niemals eine Engine namens `awt1`, `fx1` oder `swt1` geben, so dass die Bezeichnung `awt`, `fx` bzw. `swt` normalerweise ausreicht. Beim Aufnehmen verwendet QF-Test immer die kurze Variante.

Hinweis Wenn Ihre Anwendung nur die Default-Engine benutzt, können Sie auf Engine Namen ganz verzichten, oder diese explizit über das Kürzel `default` adressieren.

Im Regelfall werden Engine Namen während der Aufnahme automatisch richtig gesetzt. Nur wenn Knoten händisch eingefügt werden, ist auf das richtige Eintragen zu achten.

In einer Testsuite kommen Engine Namen in folgenden Knoten vor:

- Warten auf Client(758) Knoten. Wird nur benötigt wenn Ihre Anwendung AWT/Swing, JavaFX und SWT kombiniert und Sie auf die Initialisierung einer spezifischen GUI-Engine warten wollen.
- Fenster(919) Knoten. Die Engine eines Fenster Knotens ordnet das Fenster und alle entsprechenden Knoten der jeweiligen Engine zu. Eingebettete Komponenten der jeweils anderen Art werden in einen Knoten für ein Pseudo-Fenster ausgelagert.
- SUT-Skript(720) Knoten. Ein SUT-Skript wird auf dem Event Thread des SUT ausgeführt. Für eine kombinierte AWT/Swing, JavaFX und/oder SWT-Anwendung muss daher festgelegt werden, ob das Skript auf dem AWT/Swing, JavaFX oder dem SWT Thread laufen soll. Daher kann ein SUT-Skript Knoten nur Komponenten einer Art ermitteln und mit ihnen interagieren.
- Dateiauswahl(802) Knoten. Für Swing-Anwendungen wird der Dateiauswahl Knoten nur selten benötigt, da der Swing `JFileChooser` in Java implementiert ist und vollständig von QF-Test angesteuert werden kann. Der SWT `FileDialog` ist dagegen analog zum AWT `FileChooser` auf Betriebssystem Ebene implementiert, so dass QF-Test keinen Zugriff auf die Komponenten innerhalb des Dialogs hat. Auch der JavaFX `FileChooser` bedarf besonderer Behandlung. Daher muss das Auswählen einer Datei mit Hilfe eines Dateiauswahl Knotens simuliert werden. Da dieser Knoten nicht explizit mit einem Komponente oder Fenster Knoten assoziiert ist, muss die Engine im Knoten angegeben werden.

Kapitel 46

Starten einer Applikation aus QF-Test

Hinweis

Der Erzeugung der Startsequenz - Schnellstart-Assistent(33) ist das empfohlene Werkzeug, um Ihre Anwendung als SUT einzurichten. Als Resultat wird eine erweiterte Startsequenz erzeugt, die auch bereits für spätere Anforderungen vorbereitet ist.

Diese Kapitel enthält notwendige Details, falls die Startsequenz ”von Hand” erzeugt werden soll.

46.1 Verschiedene Methoden zum Starten des SUT

Mit dem Schnellstart Wizard bietet QF-Test ein Hilfsmittel, um Sie Schritt für Schritt durch den Prozess zur Erstellung einer Startsequenz für Ihr SUT zu leiten. Informationen zum Schnellstart Wizard finden Sie im Kapitel Kapitel 3(32).

Trotzdem soll hier auch der ”händische” Weg erklärt werden, wie Sie einen Startknoten für Ihr SUT erstellen können. Es gibt im Wesentlichen zwei Methoden zum Start einer Java-Anwendung aus QF-Test als SUT. Die erste entspricht einem normalen `java ...` Aufruf über die Kommandozeile, wobei es dabei Varianten zum Start einer Klasse oder einer jar Datei gibt. Die Alternative ist ein Skript oder ein ausführbaren Programm, das dann die Java-VM startet. Indirekte Methoden wie der Start des SUT über `ant` fallen ebenso in diese Kategorie wie Java WebStart.

Im Folgenden werden einige typische Konstellationen beispielhaft erläutert. Für nähere Informationen zu den auszufüllenden Werten folgen Sie bitte den entsprechenden Verweisen in den Referenzteil. Das Tutorial enthält weitere Beispiele zu diesem Thema.

Unabhängig davon wie Sie das SUT starten, der jeweilige Knoten sollte normalerweise direkt von einem Warten auf Client(758) Knoten mit identischem Client Attribut gefolgt werden. Auch hierzu finden Sie weiterführende Informationen im Referenzteil.

46.1.1 Starten des SUT aus einem Skript oder ausführbaren Programm

Wird Ihre Anwendung im Normalfall durch ein Skript oder ein ausführbares Programm gestartet, erstellen Sie einen SUT-Client starten[728] Knoten wie folgt:

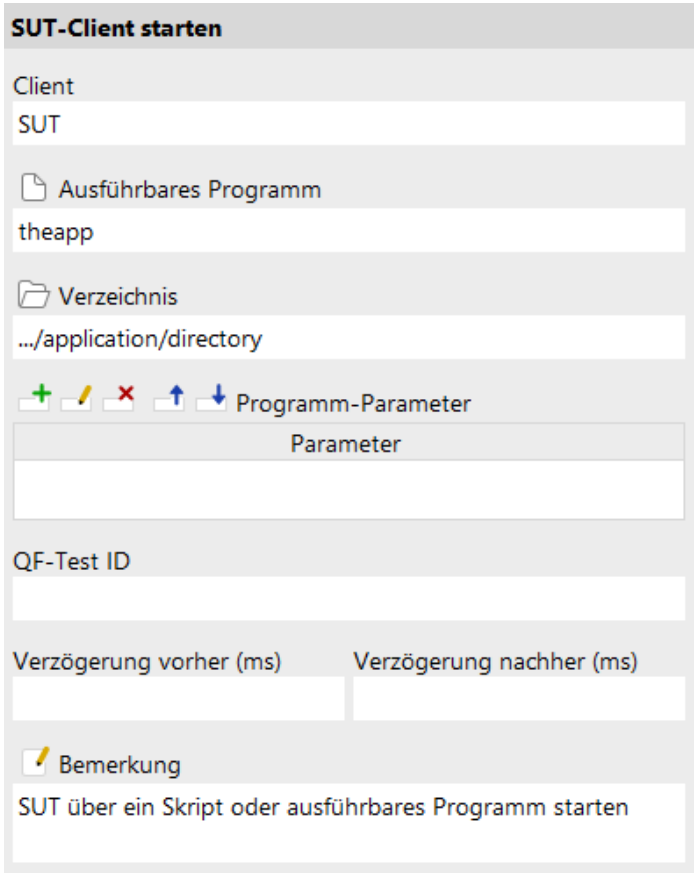


Abbildung 46.1: Starten des SUT aus einem Skript oder ausführbaren Programm

- Fügen Sie einen SUT-Client starten[728] Knoten ein.
- Geben Sie dem SUT im Client[729] Attribut einen Namen.
- Setzten Sie das Attribut Ausführbares Programm[729] auf das Skript oder das Programm, mit dem das SUT gestartet wird. Wenn sich das Skript oder Programm nicht auf dem `PATH` befindet, muss es mit vollem Pfad angegeben werden.
- Legen Sie im Verzeichnis[726] Attribut das Arbeitsverzeichnis der Anwendung fest.
- Im Falle eines Skripts sollten Sie eine eventuell vorhandene Ausgabeumleitung (z.B. `>ausgabe.log`) beim `java` Aufruf entfernen, damit die Ausgabe bei QF-

Test ankommt und im Protokoll aufgezeichnet werden kann. Ebenso behindert unter Windows ein vorangestellter `start` Befehl das Aufzeichnen der Ausgaben des SUT.

46.1.2 Starten des SUT mittels Java WebStart

Mit dem neuen Verbindungsmechanismus kann das SUT mittels Java WebStart direkt über QF-Test gestartet werden, ohne dass Änderungen am JNLP Deskriptor nötig sind (**verwenden Sie also nicht** Extras→WebStart SUT-Client Starter erstellen...). Erstellen Sie stattdessen direkt einen SUT-Client starten[728] Knoten wie folgt:

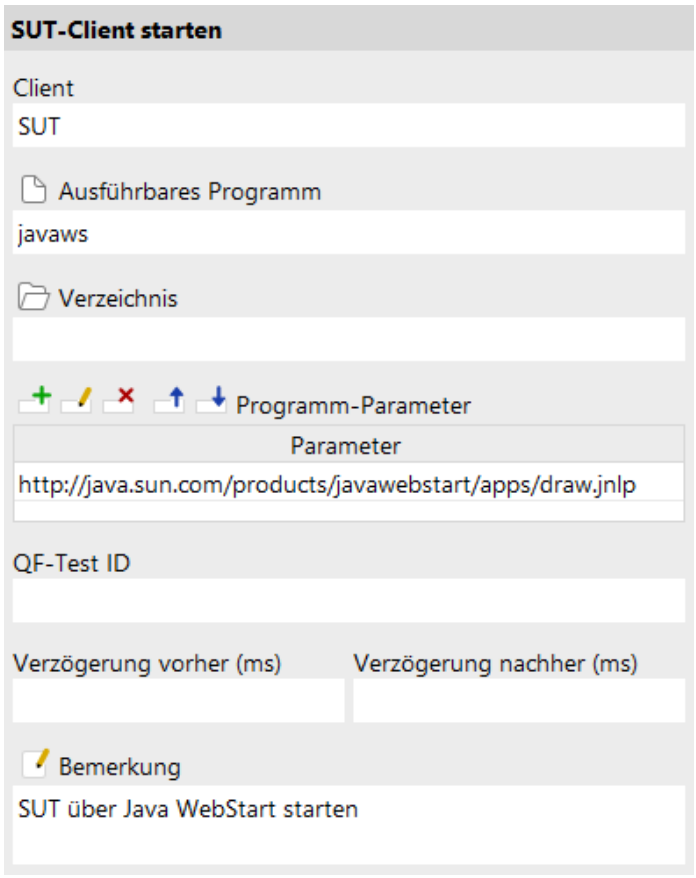


Abbildung 46.2: Starten des SUT mittels Java WebStart

- Fügen Sie einen SUT-Client starten[728] Knoten ein.
- Geben Sie dem SUT im Client[729] Attribut einen Namen.

- Setzten Sie das Attribut Ausführbares Programm[729] auf das Java WebStart Programm. Dieses heißt normalerweise `javaws` und befindet sich innerhalb des JDK oder JRE. Sie werden vermutlich den vollständigen Pfad angeben müssen.
- Für Java WebStart ist das Verzeichnis[726] Attribut normalerweise ohne Bedeutung. Allerdings sucht Java WebStart beim Start in diesem Verzeichnis nach der Datei `.javaws`, in der zum Beispiel Einstellungen zu Debugging Ausgaben vorgenommen werden können.
- Erstellen Sie in der Programm-Parameter[726] Tabelle für das Programm eine Zeile mit der URL für den JNLP Deskriptor der Anwendung.

46.1.3 Starten des SUT mittels `java -jar <Archiv>`

Wird Ihre Anwendung im Normalfall durch ein Kommando der Form `java -jar <Archiv>` gestartet, erstellen Sie einen Java-SUT-Client starten[724] Knoten wie folgt:

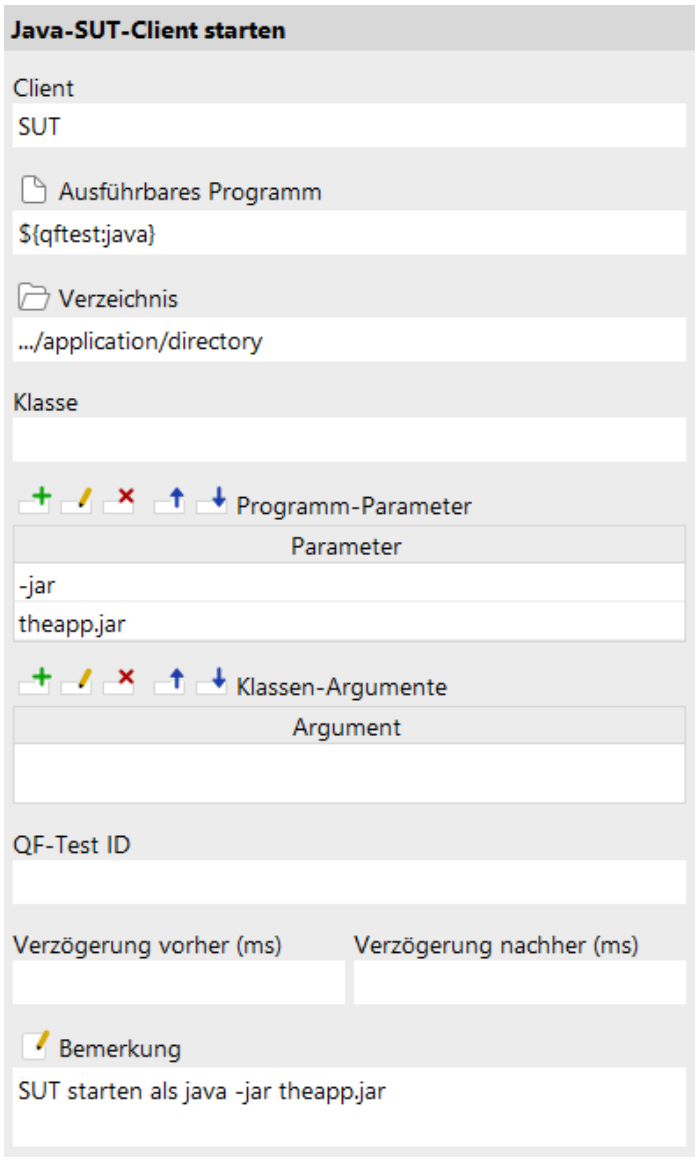


Abbildung 46.3: Starten des SUT aus einem jar Archiv

- Fügen Sie einen Java-SUT-Client starten(724) Knoten ein.
- Geben Sie dem SUT im Client(725) Attribut einen Namen.
- Wenn nötig, ändern Sie das Attribut Ausführbares Programm(726). Sein Standardwert `${qftest:java}` ist das `java` Programm mit dem QF-Test gestartet wurde.
- Legen Sie im Verzeichnis(726) Attribut das Arbeitsverzeichnis der Anwendung fest.
- Erstellen Sie zwei Zeilen in der Programm-Parameter(726) Tabelle für das Programm. Setzen Sie die erste Zeile auf `-jar` und die zweite auf den Namen des jar Ar-

chivs. Die Angabe des vollen Pfades ist nötig, wenn sich das Archiv nicht im oben angegebenen Verzeichnis befinden sollte.

46.1.4 Starten des SUT mittels `java -classpath <Pfad> <Startklasse>`

Wird Ihre Anwendung im Normalfall durch ein Kommando der Form `java -classpath <Pfad> <Startklasse>` gestartet, erstellen Sie einen Java-SUT-Client starten(724) Knoten wie folgt:

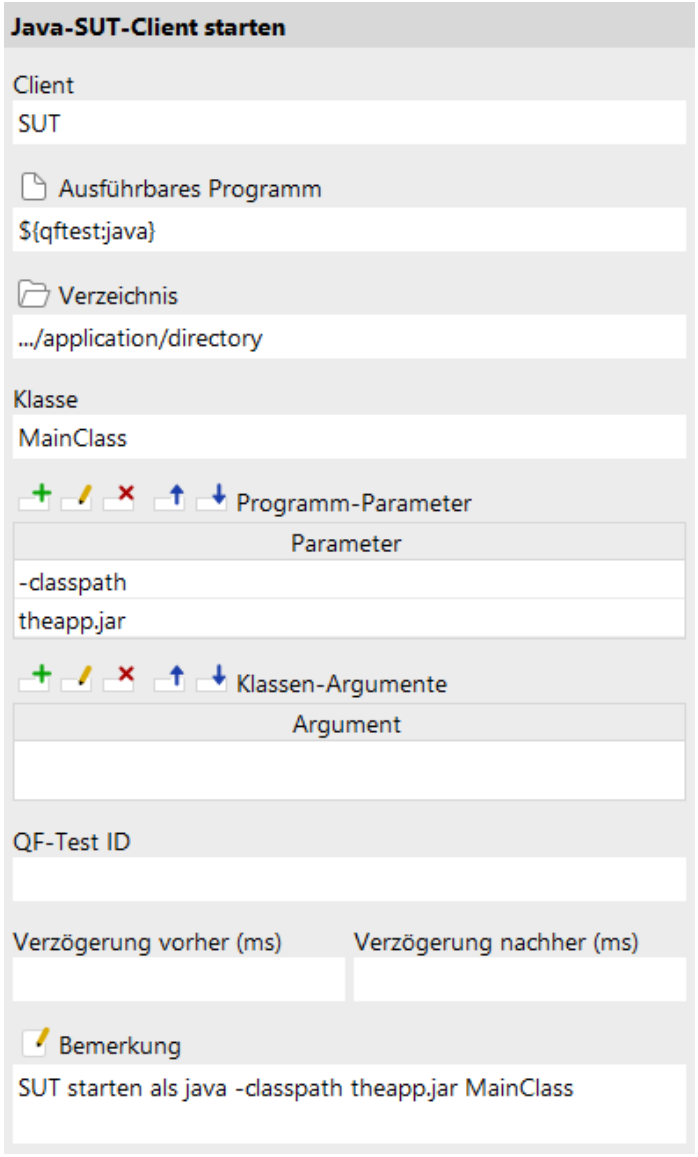


Abbildung 46.4: Starten des SUT über die Startklasse

- Fügen Sie einen Java-SUT-Client starten(724) Knoten ein.
- Geben Sie dem SUT im Client(725) Attribut einen Namen.
- Wenn nötig, ändern Sie das Attribut Ausführbares Programm(726). Sein Standardwert `${qftest:java}` ist das `java` Programm, mit dem QF-Test gestartet wurde.
- Legen Sie im Verzeichnis(726) Attribut das Arbeitsverzeichnis der Anwendung fest.
- Setzen Sie das Attribut Klasse(726) auf den vollen Namen der Startklasse (die Klasse

mit der `main()` Methode), so wie er auch für das `java`-Kommando angegeben wird.

- Erstellen Sie zwei Zeilen in der Programm-Parameter[726] Tabelle für das Programm. Setzen Sie die erste Zeile auf `-classpath` und die zweite auf die Liste der jar Archive und Verzeichnisse, aus denen sich der Classpath zusammensetzt. Für jar Archive, die sich nicht im oben angegebenen Verzeichnis befinden, muss dabei der volle Pfad angegeben werden. Dieses Argument kann sehr lang werden und ist dadurch nur mühsam direkt in der Tabelle zu editieren. In Abschnitt 2.2.5[20] ist beschrieben, wie Sie komfortabel in einem Dialog arbeiten können.

46.1.5 Starten einer Web-Anwendung im Browser

Web

Wie bei Swing, JavaFX und SWT wird der Browser für ein Web-SUT als separater Prozess aus QF-Test heraus gestartet. Um Zugriff auf die Interna des Browsers und die darin dargestellte Webseite mit ihrem *Document Object Model* (DOM) zu erhalten, bettet QF-Test Standard-Browser wie Chrome in eine eigene Hülle ein. Die Technologie zur Einbettung und für den Zugriff auf die Browser zeichnet sich aus durch besonders effizienten Zugriff auf das DOM, weit über die üblichen Browser-Schnittstellen hinaus und bietet dafür eine einheitliche Schnittstelle, welche die Unterschiede zwischen den Browsern versteckt. Dies ermöglicht QF-Test - und damit Ihnen - eine Anwendung in allen unterstützten Browsern und auf mehreren Plattformen mit nur einem Satz von Tests zu automatisieren.

Ein Browser wird mit Hilfe eines Web-Engine starten[737] Knotens gestartet.

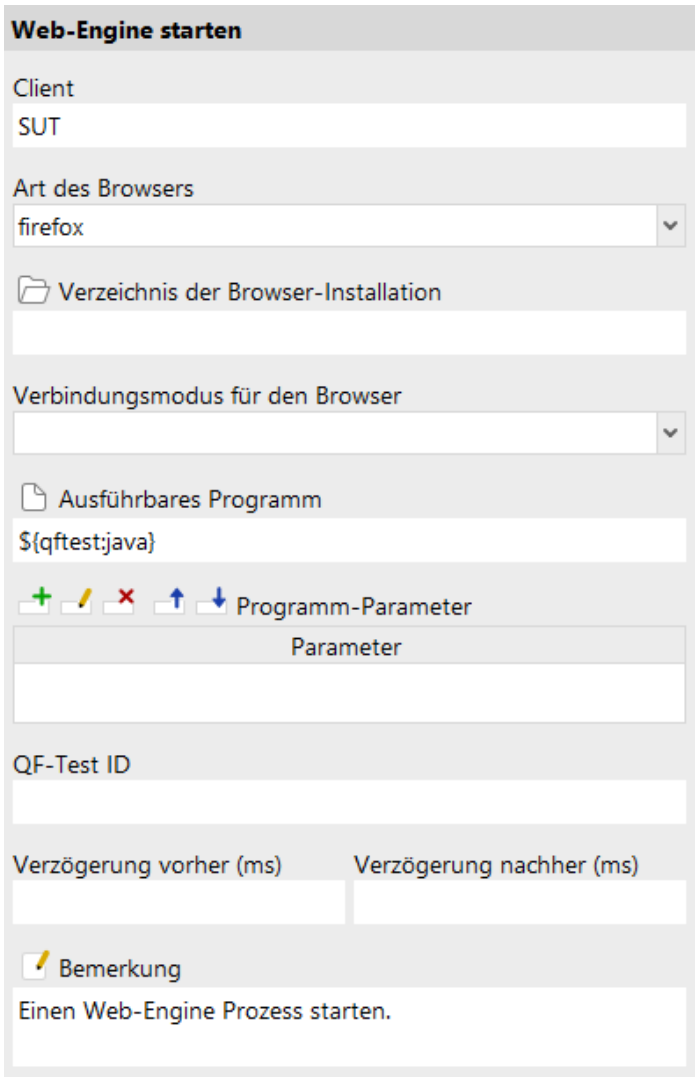


Abbildung 46.5: Starten des Browser-Prozesses

Die zu testende Webseite kann danach über einen Browser-Fenster öffnen Knoten geöffnet werden.

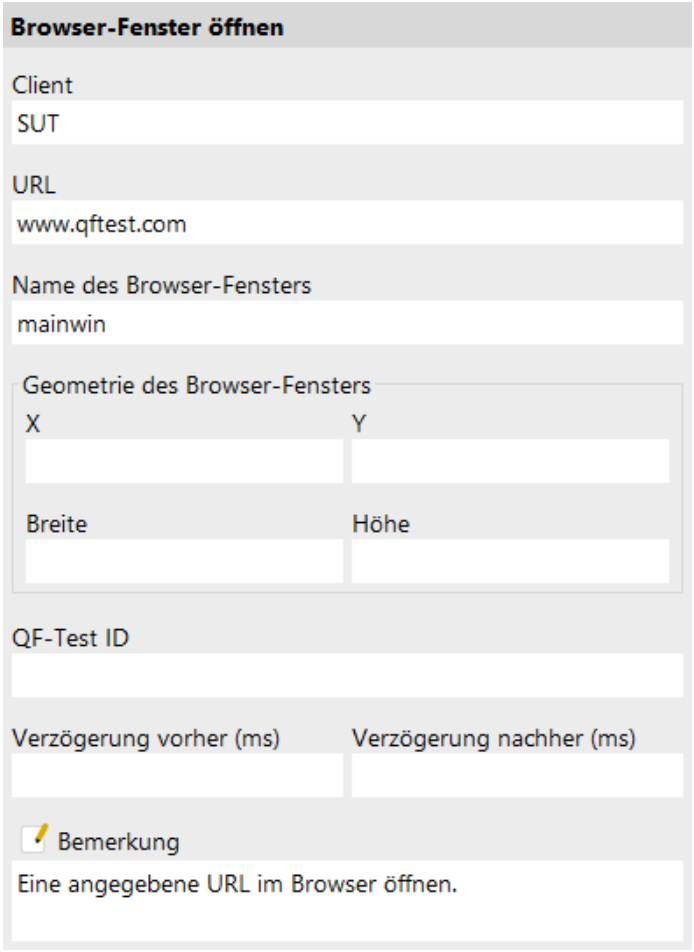


Abbildung 46.6: Öffnen der Webseite im Browser

Hinweis Beim Erstellen Ihrer Startsequenz mit dem Schnellstart-Assistenten - oder beim manuellen Setzen des Attributs Verzeichnis der Browser-Installation(738) - versuchen Sie QF-Test auf eine aktuelle Firefox oder Chrome Installation zu verweisen. Unter Linux kann der Standard-Browser Ihrer Installation an verschiedenen Orten installiert sein.

46.1.6 Öffnen eines PDF-Dokuments

4.2+ QF-Test bietet die Möglichkeit, PDF-Dokumente zu überprüfen. Dazu wird ein Client als separater Prozess aus QF-Test heraus gestartet. Um Zugriff auf die Interna des PDF-Dokuments und die darin dargestellten Objekte zu erhalten, analysiert QF-Test das PDF-Dokument in seinem eigenen Viewer.

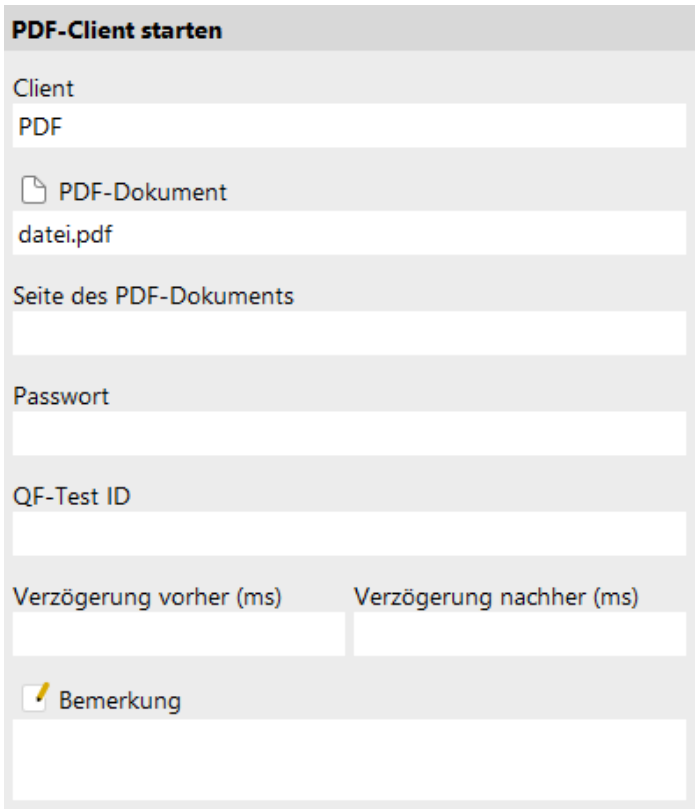


Abbildung 46.7: Öffnen eines PDF-Dokuments

Mit Hilfe des PDF-Client starten(741) Knotens wird der Viewer gestartet und das PDF-Dokument geöffnet.

Ausführliche Informationen finden Sie in Kapitel 18(286).

Kapitel 47

JRE und SWT-Instrumentierung

47.1 Deinstrumentieren eines JRE

Swing

Hinweis

Das Instrumentieren von JREs durch QF-Test ist schon seit langer Zeit nicht mehr notwendig, in den meisten Fällen sogar störend. Der folgende Abschnitt dient nur noch zur Erklärung der Hintergründe und wie ggf. eine bestehende Instrumentierung eines JRE ohne Neuinstallation entfernt werden kann.

Hinweis

Um die Instrumentierung eines JRE entfernen zu können benötigen Sie Schreibrechte auf einige seiner Unterverzeichnisse, so dass Sie gegebenenfalls mit Administratorrechten arbeiten müssen.

Zur Instrumentierung des JRE nutzte QF-Test das offizielle *accessibility* Interface, das von Java zu eben diesem Zweck zur Verfügung gestellt wird. Mit seiner Hilfe können accessibility und capture/replay Werkzeuge mit Java-Anwendungen interagieren, ohne dass diese Anwendungen davon wissen oder gar modifiziert werden müssen.

Um dieses Interface zu aktivieren erstellte oder modifizierte QF-Test die Datei `.../lib/accessibility.properties` im JRE und fügte die Klasse `de.qfs.apps.qftest.start.Connector` zur Property ”assistive_technologies” hinzu. Dies hat den Effekt, dass diese Klasse immer instantiiert wird, wenn das AWT Toolkit in einer Java-Anwendung mit diesem JRE initialisiert wird.

Um sicherzustellen dass diese Klasse immer gefunden werden kann, wurde die Datei `qfconnect.jar`, welche die Connector Klasse enthält, in das Verzeichnis `.../lib/ext` für Java-Erweiterungen kopiert.

Zum Deinstumentieren des JRE entfernen Sie zunächst den Eintrag für `de.qfs.apps.qftest.start.Connector` aus der ”assistive_technologies” Property in `.../lib/accessibility.properties`, so dass die Connector Klasse nicht mehr verwendet wird. Anschließend löschen Sie die Datei `qfconnect.jar` im `.../lib/ext` Verzeichnis. Dies ist nur möglich, wenn gerade kein Java-Programm

mit diesem JRE ausgeführt wird.

47.2 SWT-Instrumentierung

SWT

Zum Testen von SWT basierten Anwendungen mit QF-Test/swt sind spezielle Maßnahmen erforderlich. Da bei der Entwicklung von SWT die Testbarkeit von Anwendungen nicht berücksichtigt wurde, müssen diese mit leicht modifizierten SWT Klassen gestartet werden, in welchen wir SWT um die nötigen Einstiegspunkte zum Filtern von Events und Auffinden von GUI-Komponenten erweitert haben. Die Änderungen sind transparent, so dass das Verhalten einer Anwendung innerhalb und außerhalb von QF-Test nicht verändert wird.

4.5+ Sind beim Start des SUT der QF-Test Agent und die Option Ohne SWT-Instrumentierung verbinden[574] aktiviert, werden die benötigten Klassen beim Laden ohne weiteres Zutun durch den Agent ausgetauscht. Dies funktioniert für alle SWT-Versionen unter Windows und für SWT 4.8 und höher unter Linux. Ältere Versionen unter Linux benötigen nach wir vor die Instrumentierung wir unten beschrieben. Es ist generell eine gute Idee, den Prozeduraufruf zur Instrumentierung in der Startsequenz zu haben und dabei den Parameter `forceInstrumentation` auf `false` zu setzen. So kann QF-Test abhängig von den Optionen und der SWT-Version entscheiden, ob die Instrumentierung notwendig ist.

Wenn Sie den Schnellstart Wizard von QF-Test zur Erstellung der Startsequenz für Ihr SUT verwenden (siehe Kapitel 3[32]), wird er sich auch um die SWT Instrumentierung kümmern. Für diejenigen unter Ihnen, die nicht so gerne mit Wizards arbeiten, sei nun der händische Weg erklärt.

Die Standardbibliothek `qfs.qft`, die Teil der Distribution von QF-Test ist und ausführlich im Tutorial beschrieben wird, enthält im Package[680] `qfs.swt.instrument` eine Prozedur[672] namens `setup`, um die SWT-Instrumentierung durchzuführen. Fügen Sie vor dem Startknoten für Ihr SUT einen Prozeduraufruf[675] Knoten ein. Setzen Sie Name der Prozedur[676] auf `qfs.qft#qfs.swt.instrument.setup` und in den Variablendefinitionen[677] den Parameter `sutdir` auf das Installationsverzeichnis Ihrer Anwendung. Der Parameter `plugin` kann leer gelassen werden, es sei denn, Ihre Anwendung folgt nicht dem üblichen Layout des Plugin-Verzeichnisses. In diesem Fall können Sie das zu instrumentierende Plugin direkt über den `plugin` Parameter angeben. Das ist alles. Für jene, die genau wissen möchten, was hinter den Kulissen abläuft, werden nachfolgend in diesem Kapitel die manuellen Schritte zur SWT-Instrumentierung beschrieben.

47.2.1 Vorbereitung einer manuellen SWT-Instrumentierung

Die für SWT Tests unterstützten Architekturen sind 64 Bit Windows und 64 Bit Linux mit Gtk. Die benötigten Dateien werden mit QF-Test in den Verzeichnissen namens `.../qftest-9.0.0/swt/$ARCH/$VERSION` bereit gestellt, wobei `$ARCH` entweder `win32-64` oder `linux-gtk-64` ist und `$VERSION` eine der unterstützten SWT-Versionen.

Zunächst müssen Sie herausfinden, ob Ihre Anwendung eine eigenständige SWT-Anwendung ist oder auf Eclipse basiert. Werfen Sie dazu einfach einen Blick auf die Verzeichnisstruktur Ihrer Anwendung. Wenn Sie ein Verzeichnis namens `plugins` finden, das eine Datei namens `org.eclipse.swt.win32.win32.x86_X.Y.Z.jar` (unter Windows) oder `org.eclipse.swt.gtk.linux.x86_X.Y.Z.jar` (unter Linux) enthält, wobei `X.Y.Z` einer Versionsnummer wie `3.2.0` entspricht, basiert Ihre Anwendung auf Eclipse. Bei einer eigenständigen SWT-Anwendung sollten Sie dagegen eine Datei namens `swt.jar` finden, üblicherweise in einem Verzeichnis namens `lib`.

47.2.2 Manuelle SWT-Instrumentierung für Eclipse basierte Anwendungen

Ersetzen Sie einfach die Datei mit dem SWT Plugin durch ein von QF-Test instrumentiertes Plugin. Um dieses zu erstellen, führen Sie einmal die oben beschriebene Prozedur `qfs.qft#qfs.swt.instrument.setup` aus. Geben Sie dabei Ihr original Plugin (oder eine Kopie davon) im Parameter `plugin` an. QF-Test erstellt eine Kopie des Originals namens `_org.eclipse.swt....jar.orig`. Kopieren Sie dann die instrumentierte Datei in das `plugin` Verzeichnis Ihrer Anwendung. Die SWT Plugin Dateien enden mit Versionsinformation der Form `...X.Y.Z.jar`, z.B. `org.eclipse.swt.win32.win32.x86_3.2.0.jar`. Um die entsprechende Datei aus QF-Test verwenden zu können, muss der `X.Y` Teil exakt übereinstimmen. Die Unterversion Z muss in der QF-Test Variante größer oder gleich dem Original sein.

Zum Abschluss starten Sie Ihre Anwendung einmal von der Kommandozeile mit dem Argument `-clean` um den Plugin Cache der Anwendung zu aktualisieren:

```
eclipse -clean
```

Die Programmdatei Ihrer Anwendung heißt eventuell nicht `eclipse`, aber alle Eclipse basierten Anwendung sollten das Argument `-clean` unterstützten.

47.2.3 Manuelle Instrumentierung für eigenständige SWT-Anwendungen

Bei einer eigenständigen SWT-Anwendung ersetzen Sie die Datei `swt.jar` mit der gleichnamigen Datei aus dem oben erwähnten Verzeichnis von QF-Test. Machen Sie dabei zunächst eine Sicherheitskopie vom Original.

Hinweis Wenn die Anwendung über den Java-SUT-Client starten[724] Knoten gestartet wird, kann man den Classpath auch auf das entsprechende `.../qftest-9.0.0/swt/$ARCH/$VERSION/swt.jar` Archiv setzen und braucht die Originaldatei nicht zu ersetzen.

Kapitel 48

Technisches zu Komponenten

48.1 Gewichtung der Wiedererkennungsmerkmale bei aufgenommenen Komponenten

Um die gesuchte Komponente zu finden, ermittelt QF-Test für jede Komponente im SUT die Wahrscheinlichkeit, mit der sie der gesuchten Komponente entspricht. Die Komponente mit der höchsten Wahrscheinlichkeit wird dann verwendet, sofern diese Wahrscheinlichkeit über einer frei wählbaren Schwelle liegt. Zunächst werden die Wahrscheinlichkeiten der Fenster im SUT untersucht. Anschließend wird die Suche in dem Fenster mit ausreichend hoher Wahrscheinlichkeit fortgesetzt.

Analog wird auf jeder weiteren Ebene verfahren, d.h. für jede direkten und indirekten Parentknoten des gesuchten Komponente(930) Knotens, allerdings von oben nach unten. Auf jeder Ebene werden die zum Attribut Klasse(932) passenden Komponenten ermittelt und ihre Wahrscheinlichkeit bestimmt. Unsichtbare Komponenten kommen nicht in Betracht.

Auf jeder Ebene wird die Wahrscheinlichkeit einer Komponente in mehreren Stufen ermittelt:

- Ausgegangen wird von einer Wahrscheinlichkeit von 99 Prozent, welche durch Abweichungen von den Geometrievorgaben reduziert wird. Daraus wird die Basiswahrscheinlichkeit für das weitere Vorgehen ermittelt.
- Die folgenden drei Stufen können entweder einen ”Treffer” oder keinen ”Treffer” liefern oder übersprungen werden. Ist für eine Stufe kein Wert angegeben, wird sie übersprungen. Die Wahrscheinlichkeit bleibt unverändert. Für jeden dieser drei Schritte gibt es einen frei wählbaren Bonus für den Fall eines Treffers oder eine Herabsetzung für den Fall einer Abweichung. Der Bonus bewirkt, dass die Wahrscheinlichkeit auf mehr als diesen Wert angehoben, die Herabsetzung, dass sie auf unter diesen Wert reduziert wird.

- Als erstes wird die Struktur von Komponenten[930]überprüft, (nicht von Fenstern[919], für die diese Information nicht existiert). Alle Komponenten der aktuell betrachteten Container-Komponente, deren Klasse dem gegebenen Klasse[932] oder einer abgeleiteten Klasse entsprechen, werden in einer Liste gesammelt (einschließlich unsichtbarer Komponenten). Für einen Treffer muss sowohl die Zahl der zuvor ermittelten Komponenten mit der passenden Klasse, als auch der Index der Komponente in dieser Liste mit den Vorgaben übereinstimmen.

- Im zweiten Schritt wird das Merkmal[932] und eventuelle Weitere Merkmale[933] überprüft. Wenn der Test eines Weiteren Merkmals mit dem Status 'Muss übereinstimmen' fehlschlägt, wird die Komponente verworfen.

- Im dritten und letzten Schritt wird der Name[932] der Komponente getestet. Falls ein Name[932] vorhanden ist, ist dies der entscheidende Test, da der Bonus und der Malus hier die höchsten Werte haben.

Für Dialoge gibt es noch einen weiteren Schritt, der die Modalität des Dialogs überprüft. Normalerweise ist ein Dialog entweder modal oder nicht modal, so dass eine Abweichung standardmäßig die Erkennung verhindert. Es kann allerdings vorkommen, dass der selbe Dialog je nach Kontext modal oder nicht modal ist. Wenn Ihr SUT einen solchen Dialog enthält, müssen Sie die "Herabsetzung für Modal" auf einen Wert oberhalb der Mindestwahrscheinlichkeit setzen.

Wenn die errechnete Wahrscheinlichkeit einen Mindestwert nicht erreicht, wird die Komponente verworfen. Die Komponente mit der höchsten Wahrscheinlichkeit wird verwendet. Wenn in der Komponente eine Abweichung bei der Struktur, dem Merkmal oder dem Namen besteht, wird eine Meldung in das Protokoll geschrieben, da dies darauf hinweisen könnte, dass es sich doch nicht um die richtige Komponente handelt. Meistens deutet dies jedoch nur darauf hin, dass sich das SUT leicht verändert hat. Die Komponente sollte dann aktualisiert werden bevor die Änderungen kumulieren und die Komponente nicht mehr erkannt wird.

Obwohl in diesem Prozess bereits die Suche nach dem Namen dominiert, können Sie dessen Bedeutung noch vergrößern, indem Sie die Optionen Gewichtung von Namen (Wiedergabe)[548] und Gewichtung von Namen (Aufnahme)[520] auf "Name übertrifft alles" setzen. In diesem Fall vereinfacht QF-Test die Suche nach einer Komponente, sofern sie einen Namen besitzt. Statt, wie oben erklärt, alle Parentcontainer von außen nach innen abzuarbeiten, werden diese übersprungen und im Fenster direkt nach einer Komponente mit passenden Namen und Klasse gesucht. Dadurch erhöht sich die Unabhängigkeit von der Struktur des GUI, die Komponente wird auch dann noch erkannt, wenn Sie eine neue Ebene zwischen Fenster und Komponente einführen oder eine solche entfernen. Als Voraussetzung für diese Methode müssen Sie sicherstellen, dass wenn ein Name vergeben wird, dieser zumindest für die gleichzeitig sichtbaren Komponenten der selben Klasse innerhalb eines Fensters eindeutig ist.

Ist eine derartige Eindeutigkeit nicht gegeben, ist ”Hierarchie von Namen” die nächstbeste Einstellung für die beiden Optionen. Sie erfordert, dass zwei gleichnamige Komponenten zumindest unterschiedlich benannte Parentcontainer haben. Diese Einstellung bewahrt den Großteil der Vorteile und der Flexibilität von Namen. Die Wiedererkennung wird damit allerdings scheitern, wenn eine benannte Komponente aus ihrem Parentcontainer verschoben wird.

48.2 Generierung der QF-Test ID der Komponente

QF-Test folgt bei der Erstellung der QF-Test ID[931] der Komponente folgendem Algorithmus, der eine möglichst gute und eindeutige Zuordenbarkeit zum GUI-Objekt erreichen soll:

1. Das Name[932] Attribut des Komponente Knotens hat einen Wert: in diesem Fall wird dieser Wert für die QF-Test ID verwendet.
2. Es ist kein Name, aber ein Merkmal[932] vorhanden: in diesem Fall wird dieser Wert für die QF-Test ID verwendet und davor der Wert des Klasse[932] Attributs in Kleinbuchstaben gesetzt.
3. Es ist kein Name und kein Merkmal[932], aber das Weitere Merkmale[933] mit dem Namen `qfs:label` vorhanden: in diesem Fall wird dieser Wert für die QF-Test ID verwendet und davor der Wert des Klasse[932] Attributs in Kleinbuchstaben gesetzt.
4. Falls weder Name noch Beschriftungen gefunden werden, wird die Klasse in Kleinbuchstaben verwendet.

Falls die auf diesem Weg erhaltene QF-Test ID nicht eindeutig ist, wird eine laufende Zahl angehängt.

Anschließend kann die QF-Test ID noch um einen Präfix ergänzt werden. Das hängt von den folgenden Einstellungen ab:

- QF-Test ID des Fensterknotens vor QF-Test ID der Komponente setzen[522]
- QF-Test ID des Parentknotens vor QF-Test ID der Komponente setzen[522]

Die Einstellungen finden Sie in den Optionen in der Rubrik Aufnahme→Komponenten.
Die Optionen öffnen Sie über den Menüpunkt Bearbeiten→Optionen

Beispiele finden Sie in Woran erkennt man eine robuste Komponentenerkennung?[55].

Hinweis Da das QF-Test ID Attribut nur der Verlinkung der Testknoten zu den aufgezeichne-

ten Komponenten dient, kann es unter Umständen schön sein, wenn es zur besseren Lesbarkeit nachträglich geändert wird. Wenn Sie dabei einen Wert wählen, der schon vergeben ist, gibt QF-Test eine Warnung aus. Haben Sie bereits Events aufgezeichnet, die sich auf diese Komponente beziehen, bietet QF-Test an, deren QF-Test ID der Komponente Attribute automatisch anzupassen. Diese Automatik funktioniert nicht bei Referenzen mit Variablen im QF-Test ID der Komponente Attribut.

48.3 SmartIDs - allgemeine Syntax

Die verschiedenen SmartID-Features können miteinander kombiniert werden. Im Folgenden finden Sie die allgemeine Syntax für die Kombination von SmartID-Merkmalen beschrieben. Eckige Klammern markieren optionale Elemente, während Großbuchstaben für einen Platzhalter stehen:

```
#[%][noscope:][ENGINE:][KLASSE:][WERT][<INDEX>]
```

Eine SmartID besteht aus den folgenden Teilen in der folgenden Reihenfolge:

1. `#` markiert immer den Anfang einer SmartID.
2. `%` (optional) markiert WERT als Regulären Ausdruck (siehe Reguläre Ausdrücke - *Regexps*(1023)).
3. `noscope:` (optional) zeigt an, dass die SmartID auch außerhalb des aktuellen Geltungsbereichs gilt (siehe Geltungsbereich (Scope)(90)).
4. `ENGINE:` (optional) gibt die UI-Technologie an, auf die die SmartID zutrifft. Dies wird nur benötigt falls QF-Test mit mehreren Applikationen mit verschiedenen UI-Technologien gleichzeitig verwendet wird. Gültige Werte sind `awt:`, `fx:`, `swt:`, `web:` und `win:`, wobei die Groß-/Kleinschreibung keine Rolle spielt.
5. `KLASSE:` (optional) gibt die generische Klasse oder Klasse der Komponente an (siehe Kapitel 61(1329)), zum Beispiel `Label:`, `CheckBox:ComboBoxListItemCheckBox:` oder `CheckBox\:MyCheckBox:`.
6. `WERT` (optional falls KLASSE gegeben) gibt den Wert an, dem die Komponente entsprechen muss, z.B. eine Beschriftung, ein Name, oder ein anderes durch MATCH angegebenes Kriterium
7. `<INDEX>` (optional) gibt einen numerischen Index an, beginnend mit 0.

Beispiele:

- `#Button:OK`

- `#TextField:PLZ<1>`
- `#TextField:name=adresse<2>`
- `#PLZ<1>`
- `#noscope:SWT:Label:PLZ<1>`

48.4 SmartIDs: Sonderzeichen

Die Sonderzeichen `:`, `@`, `&` und `%` haben spezielle Bedeutungen in SmartIDs. Eine Komponentenklasse oder UI-Technologie wird mit ":" abgeschlossen. Ein `%` am Beginn einer SmartID kennzeichnet die Verwendung eines regulären Ausdrucks (vgl. Abschnitt 49.3(1023)). Die anderen Sonderzeichen kennzeichnen den Beginn eines Unterelements, z.B. einer Tabellenzelle. Wenn die Sonderzeichen innerhalb der SmartID als normale Zeichen genutzt werden sollen, müssen sie durch Voranstellung von \ geschützt werden.

Beispiel: Ein Dialogtitel, der eine Emailadresse enthält, soll als SmartID verwendet werden. Darin muss `@` geschützt werden, zum Beispiel `#abc\@qftest.com`.

Ein `%` am Anfang der SmartID, also direkt nach dem `#`, kann nicht geschützt werden. Hier ist es besser, einen Präfix oder die Klasse zwischen `#` und das `%` zu setzen. Nun kann `%` mit \ geschützt werden.

Beispiel: Ein Button hat die Beschriftung `%`. Als SmartID kann `#Button:\%` oder `#Label=\%` verwendet werden.

Der Doppelpunkt in vordefinierten generischen Klassen mit Klassentypen (siehe Kapitel 61(1329)), zum Beispiel `Panel:TitledPanel`, muss nicht geschützt werden.

48.5 Android - Liste der trivialen Komponentenbezeichner

Folgende Komponentenbezeichner werden standardmäßig nicht in das Attribut 'Name' übernommen. Abweichende Einstellungen können über einen NameResolver geregelt werden (siehe Abschnitt 54.1.7(1163)).

- `button`
- `canvas`
- `checkbox`

- choice
- container
- content
- dialog
- dock
- drawer
- filedlg
- frame
- label
- list
- menu
- menubar
- menuitem
- pager
- panel
- popup
- row
- scrollbar
- scrollpane
- tabs
- textfield
- toolbar
- title
- text
- win

Kapitel 49

Technische Details zu verschiedenen Themen

49.1 Drag&Drop

Für Java war Drag&Drop von Anfang an ein schwieriges Thema. In JDK 1.1 gab es dafür gar keine Unterstützung, in JDK 1.2 erste, kaum brauchbare Gehversuche und seit JDK 1.3 eine relativ vollständige und portable Implementierung, die allerdings bei manchen VMs für Stabilitätsprobleme sorgt. Der Grund für diese Schwierigkeiten ist zum einen in dem extrem unterschiedlichen Ansatz der verschiedenen Betriebssysteme für die Implementierung von Drag&Drop zu suchen, zum anderen in dem Anspruch, nicht nur Drag&Drop innerhalb von Java Programmen zu unterstützen, sondern auch zwischen Java und dem zugrundeliegenden System.

Die Konsequenz aus dieser Situation ist, dass Drag&Drop in Java vollständig ”native” implementiert ist, d.h. auf der Ebene des Betriebssystems. Die zugehörigen Events sind für QF-Test weder auswertbar, noch können sie von QF-Test mit Java-Mitteln generiert werden. Daher kann Drag&Drop nicht direkt abgespielt werden.

Dennoch bietet QF-Test für JDK 1.3 und höher Unterstützung für Drag&Drop. Die direkte Aufnahme von Drag&Drop wird ab JDK 1.4 unterstützt. Zur Wiedergabe dienen die Mausevents(775) `DRAG_FROM` und `DROP_TO`, sowie das optionalen `DRAG_OVER`. Beim Abspielen setzt QF-Test den sogenannten AWT Robot ein, der es ermöglicht, ”harte” Events auf Betriebssystemebene zu generieren und der erst ab JDK 1.3 zu Java gehört. Diese ”harten” Events bewegen den Mauszeiger tatsächlich über den Bildschirm und lösen so die nötigen Events für eine Drag&Drop Operation aus.

Auch die Simulation von Drag&Drop bei gedrückter Shift oder Strg Taste ist möglich. Ändern Sie hierzu die Modifiers(778) Attribute der `DRAG_FROM`, `DRAG_OVER` und `DRAG_TO` Mausevents und fügen Sie den Shift oder Strg modifier hinzu. Für Strg müssen Sie z.B. 2 zum aktuellen Wert addieren. Es ist sogar möglich, das Drücken oder Loslassen der

Shift oder Strg Tasten während der Drag Operation zu simulieren, indem Sie nur die Modifier eines Teils der Events ändern.

Wie erwähnt war Drag&Drop anfangs auf manchen Systemen nicht gerade stabil. Mit älteren JDKs konnte es vorkommen, dass der Einsatz des AWT Robot zum Simulieren von Drag&Drop das SUT zum Absturz bringt, oder im Extremfall sogar das gesamte System lahm legt. Inzwischen ist die Situation deutlich besser. Durch die Einführung der Interpolation von Mausbewegungen konnte die Zuverlässigkeit der Drag&Drop Wiedergabe deutlich verbessert werden. Näheres hierzu finden Sie in Abschnitt 41.3.5(551).

Hinweis Unter Windows kann es zu allem Überfluss auch zu Konflikten mit dem Maustreiber kommen. Sollte es Probleme mit Drag&Drop geben, reduzieren Sie in den Mauseinstellungen der Systemsteuerung die Geschwindigkeit des Mauszeigers auf 50%.

49.2 Timing

Ein weiteres Problem von automatischen Testläufen sticht zunächst nicht ins Auge: Das Timing. Bei jedem Durchgang ist das Laufzeitverhalten des SUT anders. Das hat verschiedene Ursachen, darunter vor allem die Belastung des Rechners durch andere Programme. Das kann dazu führen, dass eine Zielkomponente für einen von QF-Test zu simulierenden Event noch gar nicht geöffnet wurde.

Um unnötige Fehler dieser Art zu verhindern kombiniert QF-Test mehrere Methoden:

- Events werden mit der AWT `EventQueue` synchronisiert, das heißt es wird so lange kein weiterer Event an das SUT geschickt, bis alles, was sich als Folge des letzten Events in der EventQueue angesammelt hat, abgearbeitet ist.
- In manchen Fällen, vor allem bei asynchronen Updates der Oberfläche, reicht dies nicht aus. Kann eine Komponente nicht gefunden werden, wartet QF-Test eine gewisse Zeit auf deren Erscheinen, bevor ein Fehler ausgelöst wird. Diese Zeitspanne ist mittels der Option Warten auf nicht vorhandene Komponente (ms)(555) einstellbar.
- Eine analoge Verzögerung gibt es mit der Option Warten auf nicht vorhandenes Element (ms)(555) für Unterelemente von komplexen Komponenten wie Baumknoten oder Tabellenzellen.
- Für besonders lange Wartezeiten kann explizit mit Hilfe eines Warten auf Komponente(876) auf eine Komponente gewartet werden.

49.3 Reguläre Ausdrücke - *Regexps*

Die regulären Ausdrücke, die Sie in den Suchen und Ersetzen Dialogen und an diversen anderen Stellen wie den Attributen Merkmal(932) und Primärindex(937) oder in Checks(805) verwenden können, nutzen alle die standard Java-Regexp-Syntax.

8.0+ Vor QF-Test Version 3.1 wurde das GNU Regexp Package verwendet. Danach war es noch per Option als Alternative verfügbar. Mit QF-Test 8.0 wurde GNU Regexp ganz aus QF-Test entfernt.

Eine detaillierte Dokumentation für Regexps mit Hinweisen zu weiterführender Information und sogar einem ganzen Buch darüber, finden Sie bei der Java-Dokumentation für die Klasse `java.util.regex.Pattern` unter http://download.oracle.com/javase/1.5.0/docs/api/java/util/regex/Pattern.html. Ebenfalls einen Blick wert ist der Wikipedia-Eintrag.
Die wichtigsten Grundlagen haben wir kurz zusammengefasst:

- Ein '.' steht für ein beliebiges Zeichen mit Ausnahme eines Zeilenumbruchs. Mit den neuen Java-Regexps können Sie Ihren Ausdruck mit dem eingebetteten Schalter (?s) starten, um zu erreichen, dass mehrzeiliger Text wie eine einzige Zeile behandelt wird und '.' damit für wirklich jedes Zeichen steht.
- Zwischen '[' und ']' können Alternativen angegeben werden.
- Ein '?' kennzeichnet ein optionales Element das 0 oder 1 mal vorkommen darf.
- '+' bedeutet mindestens ein Element.
- Ein '*' bedeutet: beliebig viele des vorhergehenden Elements (auch 0).
- Eine Gruppe wird mit '(' und ')' gebildet. Die Zählung der Untergruppen beginnt mit 0 für den Gesamtausdruck, 1 für die erste '(' Klammer etc. Bei Suchen und Ersetzen Operationen expandiert `$n` in der ersetzen Zeichenkette zu der n^{ten} Gruppe. Beispiel: Um die Dateiendung aller Dateien welche mit `/tmp/` anfangen von `.foo` mit `.bar` zu ersetzen, sucht man nach `(/tmp/.*)\.foo` und ersetzt dann mit `$1.bar`.
- Der Balken '|' trennt Alternativen in einer Gruppe.
- '\' hebt die Sonderbedeutung eines Zeichens auf oder definiert Sonderzeichen, z.B. '\n' für LineFeed (= Zeilenumbruch), '\r' für CarriageReturn (wird für QF-Test nicht benötigt, vgl. Abschnitt 49.4(1024)) oder '\t' für Tab.

Beispiele:

- `.*` steht für einen beliebigen Text, der optional ist.

- `.+` steht für einen beliebigen Text, wobei allerdings mindestens ein Zeichen vorhanden sein muss, also einen Pflichttext.
- `[0-9]` steht für eine beliebige Ziffer.
- `[0-9]+` steht für eine beliebige Folge von Ziffern, aber mindestens eine Ziffer muss vorhanden sein.
- `[0-9]{1,3}` steht für eine beliebige Folge von Ziffern, allerdings sollte mindestens eine und maximal drei Ziffern erscheinen.
- Beliebiger Text, der das Wort 'Baum' enthält: `'.*Baum.*'`.
- Beliebiger mehrzeiliger Text: `'(?s).*'`
- Das Wort 'Baum' in beliebigem Text durch das Wort 'Tree' ersetzen: `'(.*)Baum(.*)'` und zum Ersetzen `$1Tree$2`. In der Ersetzen Box geht das allerdings einfacher indem Sie einfach 'Baum' durch 'Tree' Ersetzen und dabei die Option 'Gesamtes Attribut vergleichen' ausschalten.
- Suche nach 'Name' oder 'Namen': `'Namen?'`
- Suche nach 'Baum' oder 'Tree': `'(Baum|Tree)'`
- Ein beliebiges Wort aus Buchstaben (ohne Umlaute) und Zahlen: [0-9a-zA-Z]+
- ...

QF-Test bietet Ihnen bei Attributen, welche reguläre Ausdrücke erlauben, den Kontextmenüeintrag Text in regulären Ausdruck konvertieren an, um Sonderzeichen von regulären Ausdrücken mit '\' zu versehen, um diese zu schützen. Wo es nicht möglich ist, diese Funktion zu verwenden, zum Beispiel für Variablen, kann die spezielle Syntax `${quoteregexp:$(myVariable)}` genutzt werden, die die Sonderzeichen im Wert der Variablen mit '\' versieht, siehe Spezielle Gruppen(127).

49.4 Zeilenumbrüche in Linux und Windows

Bekanntlich verwenden Linux und Windows unterschiedliche Zeichen, um in Textdateien einen Zeilenumbruch zu definieren. Während Linux lediglich ein LineFeed ('\n', hex 0x0a) verwendet, benutzt Windows die Kombination CarriageReturn/LineFeed ('\r\n', hex 0x0D0A). Java konvertiert je nach System zwischen diesen Konventionen, was im allgemeinen ganz gut funktioniert.

Der XML-Standard definiert jedoch, dass ein XML-Parser grundsätzlich nur LineFeed für einen Zeilenumbruch zurückliefern darf, egal auf welchem System. Da QF-Test seine

Testsuiten als XML-Dateien speichert, würde dies unter Windows zu Problemen führen, wenn z.B. der mehrzeilige Inhalt eines Textfeldes überprüft werden soll. Daher konvertiert QF-Test alle Texte, die es vom SUT ausliest oder übernimmt in die Variante mit einem LineFeed als Zeilentrenner. Damit ist auch sicher gestellt, dass Tests die auf dem einen System erstellt wurden, auf dem anderen System ohne Probleme lauffähig sind.

49.5 Schützen von Sonderzeichen (*quoting*)

Ein immer wiederkehrendes Problem bei komplexen Systemen ist die Behandlung von Sonderzeichen, d.h. von Zeichen, die für das jeweilige System eine besondere Bedeutung haben. Einfachstes Beispiel sind Leerzeichen in Dateinamen. Um solche Dateien auf der Kommandozeile anzugeben, müssen Sie diese entweder in Anführungsstriche stellen, oder unter Linux mit einem *Rückstrich* (\) schützen.

Da auch QF-Test einerseits Sonderzeichen für verschiedene Aufgaben einsetzt, andererseits aber Zeichenketten vom SUT lesen muss, die beliebige Zeichen enthalten können, führt auch hier kein Weg an einem Quoting Mechanismus vorbei. Dass QF-Test auf verschiedenen Betriebssystemen läuft und seinerseits z.B. reguläre Ausdrücke(1023) mit eigenen Sonderzeichen verwendet, macht die Sache nicht einfacher. Daher versucht QF-Test, quoting so einfach wie möglich zu halten und auf die Stellen zu reduzieren, bei denen es sich nicht vermeiden lässt. Außerdem werden bei der Aufnahme vom SUT gelesene Zeichenfolgen korrekt geschützt.

Das wichtigste Sonderzeichen für QF-Test ist das Dollarzeichen '$' für die Variablenexpansion. Variablensyntax kann in fast allen Attributen verwendet werden. Wenn Sie irgendwo ein echtes '$'-Zeichen benötigen, müssen Sie es verdoppeln.

Beispiel: Um den Inhalt eines Textfeldes mittels eines Checks auf den Text "4 US$" zu überprüfen, müssen Sie im Attribut Text(808) den Wert "4 US$$" angeben.

Andere Sonderzeichen werden nur an besonderen Stellen eingesetzt und müssen und dürfen auch nur dort geschützt werden. Dies sind das '#'-Zeichen zum suiteübergreifenden Zugriff auf Prozeduren(672) und Komponenten(930), sowie die Zeichen '@', '&' und '%' für die spezielle Syntax für Unterelemente(95). Da es sich hierbei um Trennzeichen handelt, können Sie nicht durch Verdoppelung geschützt werden. Daher folgt QF-Test der Konvention, den Rückstrich (\) zu verwenden, was auch diesen selbst zum Sonderzeichen macht. Damit das unter Windows nicht zur Quotinghölle mit Dateinamen führt, interpretiert QF-Test den Rückstrich nur dort, wo diese Zeichen ihre besondere Bedeutung haben und selbst dort ist ein Rückstrich, der nicht von einem Sonderzeichen gefolgt wird, ein ganz normales Zeichen.

Präzise heißt das, dass Sie die Zeichen '#' und '\' im Attribut Name der Prozedur(676) eines Prozeduraufrufs(675) schützen müssen, sowie die Zeichen '#', '\', '@', '&' und '%' in den At-

tributen QF-Test ID der Komponente(776) von Events und Checks, sowie Primärindex(937) und Sekundärindex(938) von Elementen(936). Bedenken Sie insbesondere, dass der Rückstrich auch zum Quoten von Sonderzeichen in regulären Ausdrücken(1023) verwendet wird. Um einen echten Rückstrich in einen regulären Ausdruck für ein Unterelement zu bekommen, müssen Sie diesen zunächst selbst für den regulären Ausdruck quoten (also '\\') und diese beiden Rückstriche noch einmal für QF-Test selbst (also '\\\\').

Leider ist damit noch nicht alles gesagt. QF-Test verwendet in Unterelementen für die Knoten von `JTree` Komponenten das Zeichen '/' als Trennzeichen für die Pfade der Knoten. Für Baumknoten müssen Sie daher dieses Zeichen zusätzlich schützen, wenn Sie ein echtes '/'-Zeichen benötigen. Für Unterelemente anderer Komponenten ist dies nicht nötig.

49.6 Auflösen von inkludierten Dateien

Dies ist ein Abschnitt den Sie hoffentlich nie lesen müssen. Er erklärt detailliert, wie implizite Referenzen von Prozeduren und Komponenten bei der Wiedergabe von Tests aufgelöst werden. Wenn Sie diesen Abschnitt doch lesen müssen, ist das ein Zeichen dafür, dass die Include Hierarchie Ihrer Testsuiten zu kompliziert geworden ist und Sie diese eventuell vereinfachen sollten.

Es gibt im wesentlichen zwei Fälle, in denen QF-Test implizit Referenzen in anderen Testsuiten auflöst, nämlich immer wenn die angeforderte Prozedur oder Komponente in der aktuellen Suite nicht gefunden wird und

- Die Suite andere Testsuiten über das Attribut Inkludierte Dateien(596) des Testsuite(595) Knotens einbindet. QF-Test durchsucht dann diese Testsuiten in der angegebenen Reihenfolge.
- Diese Testsuite (genauer gesagt eine ihrer Prozeduren) von einer anderen Testsuite aufgerufen wurde. QF-Test sucht dann auch in der aufrufenden Suite nach einem passenden Knoten.

Die ganze Sache wird kompliziert, wenn (möglicherweise indirekte) Prozeduraufrufe und (möglicherweise indirekte, evtl. sogar rekursive) Includes kombiniert werden. Die folgende detaillierte Aufstellung des Suchalgorithmus wird Ihnen hoffentlich helfen, Probleme mit inkludierten Dateien zu lösen.

- Wann immer ein Testlauf die aktuelle Suite verlässt, um mit einer Prozedur oder einer Komponente in einer anderen Testsuite fortzufahren, wird die andere Suite zur aktuellen Suite. Dabei geschehen zwei Dinge: Die alte Suite wird auf den *Aufrufstapel* gelegt und die Variablen der neuen Suite werden auf den Sekundärstapel für Variablendefinitionen gelegt (vgl. Kapitel 6(116)), so dass sie stärker binden, als

die Variablendefinitionen der alten Suite. Im Protokoll wird dieser Prozess durch einen *Testsuite Wechsel* Knoten festgehalten, der alle Knoten des Testlaufs außerhalb der alten Suite enthält.

- Jede Suche startet in der aktuellen Suite und geht dann von oben nach unten durch den Aufrufstapel weiter. Beispiel: Suite A ruft Suite B auf, die Suite C aufruft. Dann wird zunächst C, dann B, dann A durchsucht.

- Includes haben Vorrang vor dem Aufrufstapel. Das bedeutet, dass bei der Suche in der aktuellen Suite oder den Testsuiten auf dem Aufrufstapel bei jedem Schritt die inkludierten Dateien durchsucht werden, bevor mit der nächsten Suite auf dem Aufrufstapel fortgefahren wird. Beispiel: Suite A ruft Suite B auf, welche Suite C einbindet. A ist auf dem Aufrufstapel und B ist die aktuelle Suite. Dann wird zunächst B, dann C, dann A durchsucht.

- Im Fall mehrfacher, ggf. indirekter Includes wird die Suche ”depth-first”, d.h. zuerst in die Tiefe gehend durchgeführt. Dabei wird die Reihenfolge der Includes beachtet. Beispiel. Suite A bindet Testsuiten B und C ein, Suite B bindet Suite D ein. Dann wird zunächst A, dann B, dann D und dann C durchsucht.

- Wird eine Prozedur in einer (evtl. indirekt) eingebundenen Suite gefunden (im Gegensatz zu einer Suite auf dem Aufrufstapel), findet der Wechsel von der aktuellen zur neuen aktuellen Suite nicht in einem Schritt statt. Für diese Erklärung brauchen wir von Anfang an ein Beispiel: Sagen wir Suite A ruft Suite B auf und A bindet Suite C ein. B ruft eine Prozedur auf, die (auf dem Umweg über A) in C gefunden wird. Anstatt direkt von B zu C zu wechseln, wird zunächst A zur aktuellen Suite und dann B. Dadurch landet A noch einmal über B auf dem Aufrufstapel und die Variablendefinitionen von A werden über die von B auf den Sekundärstapel gelegt. Die Argumentation hierfür ist folgende: Suite C, welche jetzt die aktuelle Suite ist, ”ist näher an” Suite A, welche C einbindet, als an Suite B, die lediglich von A aufgerufen wurde. Man könnte auch sagen, dass Testsuite und Include Datei eine Art Einheit bilden, so dass aus der Sicht von B die Testsuiten A und C immer wie eine einzelne Testsuite aussehen, solange B nicht explizit C aufruft.

- Das war es, bis auf ein kleines Detail: Während der Suche überprüft QF-Test jede Testsuite höchstens einmal. Eine zweiter Durchgang wäre ohnehin sinnlos, aber es handelt sich hierbei nicht nur um eine Optimierung, da dadurch Probleme mit rekursiven Includes vermieden werden, wenn z.B. Suite A Suite B einbindet und umgekehrt.

Sollten Sie wirklich ein Problem haben, herauszufinden wie, warum oder warum nicht eine bestimmte Prozedur oder Komponente ermittelt wurde, sollten Sie als erstes einen Blick auf das Protokoll werfen. Es zeigt Ihnen genau, welche Testsuiten aktuell sind und welche Variablenexpansionen stattgefunden haben.

Kapitel 50

Skripting (Jython, Groovy und JavaScript)

Dieser Abschnitt erläutert die technischen Details der Integration von Jython, Groovy und JavaScript in QF-Test. Er enthält eine vollständige Referenz der API, die Jython-, Groovy- und JavaScript-Skripten zur Interaktion mit QF-Test zur Verfügung steht. Eine weniger technische Einführung finden Sie in Kapitel 11 (186).

50.1 Pfad für das Laden der Module

Der Loadpath für Module werden in den Skriptsprachen aus verschiedenen Teilen in folgender Reihenfolge zusammengesetzt:

- Das benutzerdefinierte Skript-Verzeichnis.
- Das Verzeichnis `qftest/qftest-9.0.0/<Name der Skriptsprache>`

Zusätzlich wird dem Pfad während der Ausführung eines Server-Skript oder SUT-Skript Knotens das Verzeichnis der Testsuite vorangestellt.

Das Verzeichnis `qftest/qftest-9.0.0/<Skriptsprache>` enthält interne Module der jeweiligen Skriptsprache. Sie sollten diese Dateien nicht modifizieren, da sie sich jederzeit ändern können und auf die jeweilige Version von QF-Test zugeschnitten sind.

Ihre eigenen Module sollten Sie in das benutzerdefinierte Skript-Verzeichnis legen. Die Pfade zu diesen Verzeichnissen finden Sie via Hilfe→Info unter ”Systeminfo” als dir.<Skriptsprache>. Dabei müssen die Module die jeweilige Endung haben (`.py` für Jython, `.groovy` für Groovy und `.js` für JavaScript) haben.

In Jython können Sie die System Property `python.path` nutzen, um weitere Verzeichnisse zum Loadpath hinzuzufügen.

50.2 Das Plugin Verzeichnis

Mit den Skriptsprachen kann auch auf Java-Klassen und Methoden zugegriffen werden, die nichts speziell mit QF-Test zu tun haben, indem diese Klassen einfach importiert werden, z.B.:

```
from java.util import Date
from java.text import SimpleDateFormat
print SimpleDateFormat("yyyy-MM-dd").format(Date())
```

Beispiel 50.1: Zugriff auf Java-Klassen mittels Jython

Zugriff gibt es auf alle Klassen im Java-Klassenpfad, das heißt die der Standard Java-API sowie QF-Test's eigene Klassen. Beachten Sie, dass die Umgebungsvariable `CLASSPATH` von QF-Test ignoriert wird; falls nötig, kann aber `QFTEST_CLASSPATH` mit demselben Wert definiert werden. Für das SUT hängt vieles vom verwendeten ClassLoader Konzept ab. Insbesondere bei WebStart und Eclipse/RCP ist es schwierig, Klassen des SUT direkt zu importieren.

Zusätzlich gibt es Plugin Verzeichnisse, in die Sie einfach eine jar Datei stellen können, um sie in Skripten verfügbar zu machen. QF-Test sucht dazu nach einem `plugin` Verzeichnis. Das aktuell verwendete Plugin-Verzeichnis finden Sie via Hilfe→Info unter "Systeminfo" als dir.plugin. Das Plugin-Verzeichnis kann auch über das Kommandozeilenargument `-plugindir <Verzeichnis>`(986) geändert werden.

Jar-Dateien im primären Pluginverzeichnis stehen sowohl Server-Skript, als auch SUT-Skript Knoten zur Verfügung. Um eine jar Datei ausschließlich für Server-Skripte oder ausschließlich für SUT-Skripte bereitzustellen, stellen Sie diese stattdessen in das zugehörige Unterverzeichnis namens `qftest` bzw. `sut`.

50.3 Initialisierung (Jython)

Beim Start von QF-Test und des SUT wird ein Jython-Interpreter erzeugt und in die Applikation eingebettet. Für QF-Test wird dabei das Modul `qftest` geladen, für das SUT das Modul `qfclient`. Beide basieren auf dem Modul `qfcommon`, das den gemeinsamen Code enthält. Diese Module werden benötigt, um die Schnittstelle zum Runcontext und den globalen Namespace bereitzustellen.

Anschließend wird der Loadpath `sys.path` nach Ihren persönlichen Modulen für die Initialisierung durchsucht. Für QF-Test wird die Datei `qfserver.py` geladen, für das SUT die Datei `qfsut.py`. In beiden fällen werden die Dateien mittels `execfile` ausgeführt, womit der Inhalt der Dateien direkt im globalen Namespace landet. Dies ist

für Initialisierungsdateien von Vorteil, da alle Module, die von diesen importiert werden, direkt für Server-Skript und SUT-Skript Knoten zur Verfügung stehen. Bedenken Sie bei Ihren Initialisierungsdateien, dass zu diesem Zeitpunkt noch kein Runcontext und kein Testsuite-spezifisches Verzeichnis in `sys.path` zur Verfügung stehen.

50.4 Die Namespace Umgebung für Skript-Knoten (Jython)

Die Umgebung in der Server-Skripte oder SUT-Skripte ausgeführt werden, wird durch den globalen und lokalen Namespace bestimmt, die während der Ausführung definiert sind. Namespaces sind Jython Dictionaries, welche die Bindungen für die globalen und lokalen Variablen beinhalten.

Der globale Namespace wird von allen Skripten, die in demselben Jython-Interpreter laufen, gemeinsam genutzt. Er enthält zunächst nur die Klassen `TestException` und `UserException`, das Modul `qftest` bzw. `qfclient` für QF-Test bzw. das SUT, und alles, was in `qfserver.py` bzw. `qfsut.py` definiert und importiert wurde. Wenn Sie in einem Skript eine Variable mit Hilfe des `global` Statements als global deklarieren und ihr einen Wert zuweisen, wird diese in den globalen Namespace aufgenommen und steht damit für weitere Skripte zur Verfügung. Zusätzlich nimmt QF-Test automatisch alle Module in den globalen Namespace auf, die während der Ausführung eines Skripts importiert werden.

Der lokale Namespace wird für jedes Skript neu erstellt. Beim Aufruf des Skripts enthält er das Objekt `rc`, die Schnittstelle zu QF-Tests Runcontext, sowie `true` und `false` mit den Werten `1` und `0` zur besseren Integration mit QF-Test.

Für den Zugriff auf und das Setzen von Variablen in einem fremden Jython-Interpreter, stehen die Runcontext Methoden `fromServer`, `fromSUT`, `toServer` und `toSUT` zur Verfügung.

50.5 Die API des Runcontexts

Das Runcontext-Objekt `rc` ist die Schnittstelle zum Ausführungszustand des laufenden Tests in QF-Test. Die Verwendung einer zusätzlichen Schicht erlaubt Änderungen an der Implementierung von QF-Test, ohne die Schnittstelle für Skripte zu gefährden.

Es folgt eine alphabetische Aufstellung aller Methoden des Runcontext-Objekts `rc`. Die verwendete Syntax ist ein Gemisch aus Java und Python. Python unterstützt zwar selbst keine statische Typisierung, die Parameter werden jedoch an Java weitergereicht, so dass falsche Typen Exceptions auslösen können. Folgt einem Parameter ein '='-Zeichen

und ein Wert, ist dies der Defaultwert des Parameters und eine Angabe beim Aufruf ist optional.

Hinweis

Die Groovy-Syntax für Keyword-Parameter unterscheidet sich von Jython. Groovy verwendet ':' statt '='. Dies ist besonders tückisch, weil z.B. `rc.logMessage("bla", report=true)` durchaus legaler Groovy-Code ist, der allerdings nicht den gewünschten Effekt hat. Das '=' ist hierbei eine Zuweisung mit dem Wert true, der dann ganz einfach als zweiter Parameter übergeben wird, so dass der Aufruf äquivalent ist zu `rc.logMessage("bla", true)`. Hierbei wird true aber für `dontcompactify` statt `report` verwendet. Die korrekte Version für Groovy lautet `rc.logMessage("bla", report:true)`.

`void addDaemonLog(byte[] data, String name=None, String comment=None, String externalizename=None)`

Fügt ein von einem DaemonRunContext abgeholtes Protokoll in das aktuelle Protokoll ein.

Parameter

`data`	Das Bytearray, das mittels DaemonRunContext.getRunLog() abgeholt wurde.
`name`	Ein optionaler Name für den Knoten des Daemon-Protokolls. Falls nicht angegeben wird die ID des Daemon verwendet.
`comment`	Ein optionaler Kommentar für den Knoten des Daemon-Protokolls.
`externalizename`	Ein optionaler Name zum extrahieren des Daemon-Protokolls, um es als Teil eines geteilten Protokolls zu speichern.

`void addResetListener(ResetListener listener)`

Nur Server. Registriert einen `ResetListener` beim aktuellen Runcontext.

Parameter

`listener`	Den Listener der registriert werden soll. Der Listener sollte das Interface de.qfs.apps.qftest.extensions.qftest.ResetListener implementieren.

`void addTestRunListener(TestRunListener listener)`

Registriert einen `TestRunListener` beim aktuellen Runcontext. Im interaktiven Modus und im Batchmodus gibt es nur einen gemeinsamen Runcontext, so dass der Listener so lange aktiv bleibt, bis er via `removeTestRunListener` oder `clearTestRunListeners` entfernt wird. Im Daemonmodus hat jeder `DaemonRunContext` seinen eigenen Satz von TestRunListenern. Details zum `TestRunListener` API finden Sie in Abschnitt 54.6(1223).

Parameter

`listener`	Der zu registrierende Listener.

`String callProcedure(String name, Map parameters=None)`

Ruft eine Prozedur(672) in einer Testsuite auf.

Diese Methode kann auch aus einem SUT-Skript aufgerufen werden. Dabei können allerdings seltsame Seiteneffekte auftreten, da das Skript im AWT Event Dispatch Thread ausgeführt wird. Obwohl QF-Test diese Effekte sauber abfängt, sollten Sie, wann immer möglich, Prozeduren aus einem Server-Skript heraus aufrufen.

Parameter

`name`	Der vollständige Name der Prozedur.
`parameters`	Die Parameter für die Prozedur, ein Dictionary. Die Schlüssel und Werte können beliebige Objekte sein. Diese werden beim Aufruf in Zeichenketten umgewandelt.
Rückgabewert	Der von der Prozedur mittels eines optionalen Return(678) Knotens zurückgegebene Wert.

`int callTest(String name, Map parameters=None)`

Nur Server. Ruft einen Testfall(599) oder Testfallsatz(606) aus einer Testsuite oder eine ganze Testsuite auf.

Parameter

`name`	Der vollständige Name des Testfall oder Testfallsatz.
`parameters`	Die Parameter für den aufzurufenden Knoten, ein Dictionary. Die Schlüssel und Werte können beliebige Objekte sein. Diese werden beim Aufruf in Zeichenketten umgewandelt.
Rückgabewert	Der Status der Testausführung. Entweder rc.OK, rc.WARNING, rc.ERROR, rc.EXCEPTION, rc.SKIPPED oder rc.NOT_IMPLEMENTED.

`int callTestAsProcedure(String name, Map parameters=None)`

Nur Server. Ruft einen Testfall[599], Testfallsatz[606] aus einer Testsuite oder eine ganze Testsuite auf, allerdings analog zu einen Prozeduraufruf, so dass eine unbehandelte Exception den gesamten Aufruf beendet und nicht nur den gerade ausgeführten Testfall.

Parameter

`name`	Der vollständige Name des Testfall oder Testfallsatz.
`parameters`	Die Parameter für den aufzurufenden Knoten, ein Dictionary. Die Schlüssel und Werte können beliebige Objekte sein. Diese werden beim Aufruf in Zeichenketten umgewandelt.
Rückgabewert	Der Status der Testausführung. Entweder rc.OK, rc.WARNING, rc.ERROR, rc.EXCEPTION, rc.SKIPPED oder rc.NOT_IMPLEMENTED.

Boolean check(boolean condition, String message, int level=rc.ERROR, boolean report=true, boolean nowrap=false)

Prüft, ob eine Bedingung zutrifft und gibt eine entsprechende Meldung aus.

Parameter

condition	Die zu überprüfende Bedingung.
message	Die Meldung, die ausgegeben werden soll. Abhängig vom Ergebnis wird "Check OK: " oder "Check fehlgeschlagen: " vorangestellt. Für den alten XML- oder HTML-Report wird die Meldung wie ein Check-Knoten behandelt, wenn sie mit einem '!' beginnt.
level	Die Gewichtung des Fehlers, falls der Check fehlschlägt. Die folgenden Konstanten sind hierfür im Runcontext definiert: • rc.OK • rc.WARNING • rc.ERROR • rc.EXCEPTION Für die Stufe rc.EXCEPTION wird im Fehlerfall eine UserException(966) geworfen.
report	Fall true wird der Check im Report aufgeführt. Nur anwendbar wenn level <= rc.WARNING.
nowrap	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen.
Rückgabewert	Das Ergebnis des Checks.

`Boolean checkEqual(Object actual, Object expected, String message, int level=rc.ERROR, boolean report=true, boolean nowrap=false)`

Prüft, ob ein Objekt einen vorgegebenen Wert hat und gibt eine entsprechende Meldung aus.

Parameter

`actual`	Der tatsächliche Wert.
`expected`	Der erwartete Wert.
`message`	Die Meldung, die ausgegeben werden soll. Abhängig vom Ergebnis wird ”Check OK: ” oder ”Check fehlgeschlagen: ” vorangestellt. Im Fehlerfall werden auch der erwartete und der tatsächliche Wert ausgegeben.
`level`	Die Gewichtung des Fehlers, falls der Check fehlschlägt. Die folgenden Konstanten sind hierfür im Runcontext definiert: • `rc.OK` • `rc.WARNING` • `rc.ERROR` • `rc.EXCEPTION` Für die Stufe `rc.EXCEPTION` wird im Fehlerfall eine `UserException`(966) geworfen.
`report`	Fall true wird der Check im Report aufgeführt. Nur anwendbar wenn level <= rc.WARNING.
`nowrap`	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen.
Rückgabewert	Das Ergebnis des Checks.

Boolean checkImage(ImageRep actual, ImageRep expected, String message, int level=rc.ERROR, boolean report=true, boolean nowrap=false)

Prüft, ob zwei `ImageRep` (siehe Abschnitt 54.9.1[1233]) Objekte gleich sind und gibt eine entsprechende Meldung aus. Der Vergleich wird mittels der `equals` Methode des `ImageComparator` (siehe Abschnitt 54.9.2[1236]) Objektes des erwarteten `ImageRep` Objektes durchgeführt.

Parameter

actual	Das aktuelle `ImageRep` Objekt.
expected	Das erwartete `ImageRep` Objekt.
message	Die Meldung, die ausgegeben werden soll. Abhängig vom Ergebnis wird ”Check OK: ” oder ”Check fehlgeschlagen: ” vorangestellt. Im Fehlerfall werden auch der erwartete und der tatsächliche Wert ausgegeben. Für den alten XML- oder HTML-Report wird die Meldung wie ein Check-Knoten behandelt, wenn sie mit einem ’!’ beginnt.
level	Die Gewichtung des Fehlers, falls der Check fehlschlägt. Die folgenden Konstanten sind hierfür im Runcontext definiert: • `rc.OK` • `rc.WARNING` • `rc.ERROR` • `rc.EXCEPTION` Für die Stufe `rc.EXCEPTION` wird im Fehlerfall eine `UserException`[966] geworfen.
report	Fall true wird der Check im Report aufgeführt. Nur anwendbar wenn level <= rc.WARNING.
nowrap	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen.
Rückgabewert	Das Ergebnis des Checks.

Object[] checkImageAdvanced(ImageRep actual, ImageRep expected, String message, String algorithm, int level=rc.ERROR, boolean report=true, boolean nowrap=false)

Prüft, ob zwei `ImageRep` (siehe Abschnitt 54.9.1(1233)) Objekte gleich sind und gibt eine entsprechende Meldung aus. Der Vergleich wird mittels des angegebenen Algorithmus durchgeführt.

Parameter

actual	Das aktuelle `ImageRep` Objekt.
expected	Das erwartete `ImageRep` Objekt.
message	Die Meldung, die ausgegeben werden soll. Abhängig vom Ergebnis wird ”Check OK: ” oder ”Check fehlgeschlagen: ” vorangestellt. Im Fehlerfall werden auch der erwartete und der tatsächliche Wert ausgegeben. Für den alten XML- oder HTML-Report wird die Meldung wie ein Check-Knoten behandelt, wenn sie mit einem ’!’ beginnt.
algorithm	Spezifiziert den für den Vergleich zu nutzenden Algorithmus wie in Kapitel 59(1309) beschrieben.
level	Die Gewichtung des Fehlers, falls der Check fehlschlägt. Die folgenden Konstanten sind hierfür im Runcontext definiert: • `rc.OK` • `rc.WARNING` • `rc.ERROR` • `rc.EXCEPTION` Für die Stufe `rc.EXCEPTION` wird im Fehlerfall eine `UserException`(966) geworfen.
report	Fall true wird der Check im Report aufgeführt. Nur anwendbar wenn level <= rc.WARNING.
nowrap	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen.
Rückgabewert	Ein Array mit dem Inhalt: Das Ergebnis des Checks als Boolean. Das Ergebnis des Checks als Wahrscheinlichkeit der Übereinstimmung. Abhängig vom Algorithmus das transformierte Bild des erwarteten Abbildes als `ImageRep`. Abhängig vom Algorithmus das transformierte Bild des erhaltenen Abbildes als `ImageRep`. Gegebenenfalls weitere Informationen.

`void clearGlobals()`

Nur Server. Löscht alle globalen Variablen.

`void clearProperties(String group)`

Nur Server. Löscht einen Satz von geladenen Properties oder Ressourcen.

Parameter

`group`	Der Name der Gruppe von Properties oder Ressourcen.

`void clearTestRunListeners()`

Entfernt alle `TestRunListener` aus dem aktuellen Runcontext.

`String expand(String text)`

Expandiert eine Zeichenkette unter Verwendung der üblichen QF-Test Variablensyntax für `$(...)` oder `${...:...}`.

Denken Sie daran, die '$'-Zeichen zu verdoppeln, damit die Expansion nicht bereits vor dem Aufruf des Skripts geschieht (vgl. Abschnitt 49.5[1025]).

Parameter

`text`	Der zu expandierende Text.
Rückgabewert	Der expandierte Text.

`Object fromServer(String name)`

Nur SUT. Ermittelt den Wert einer globalen Variablen im Jython- oder Groovy-Interpreter von QF-Test. Von einem Groovy-SUT-Skript können Sie somit zum Beispiel den Wert einer globale Variable des Groovy-Interpreters von QF-Test auslesen. Wenn die Variable nicht definiert ist, wird ein `KeyError` geworfen.

Parameter

`name`	Der Name der Variablen.
Rückgabewert	Der Wert der Variablen.

`Object fromSUT(String client, String name)`

Nur Server. Ermittelt den Wert einer globalen Variablen im Jython- oder GroovyInterpreter des SUT. Von einem Groovy-Server-Skript können Sie somit zum Beispiel den Wert einer globale Variable des SUT Groovy-Interpreters auslesen. Wenn die Variable nicht definiert ist, wird ein `KeyError` geworfen.

Parameter

`client`	Der Name des SUT Clients.
`name`	Der Name der Variablen.
Rückgabewert	Der Wert der Variablen.

`Boolean getBool(String varname)`
Ermittelt den Wert einer QF-Test Variable, vergleichbar mit `$(varname)`, und interpretiert ihn als Boolean.

Parameter

`varname`	Der Name der Variable.
Rückgabewert	Der Wert der Variable.

`Boolean getBool(String group, String name)`
Ermittelt den Wert einer QF-Test Ressource oder Property, vergleichbar mit `${group:name}`, und interpretiert ihn als Boolean.

Parameter

`group`	Der Name der Gruppe.
`name`	Der Name der Ressource oder Property.
Rückgabewert	Der Wert der Ressource oder Property.

`Exception getCaughtException()`
Nur Server. Wird das Skript unterhalb eines Catch(707) Knotens ausgeführt, liefert diese Methode die darin gefangene Exception. In allen anderen Fällen ist der Rückgabewert `None`.

Rückgabewert	Die gefangene Exception.

`Component getComponent(String id, int timeout=0, boolean hidden=false)`

Nur SUT. Ermittelt eine Komponente oder ein Unterelement mit Hilfe von QF-Tests Algorithmus zur Wiedererkennung.

Parameter

`id`	Die QF-Test ID[931] des Komponente[930] Knotens, der die Komponente in der Testsuite repräsentiert.
`timeout`	Dieser Parameter wird ignoriert und ist immer 0 für SUT-Skripte, die auf dem Event Dispatch Thread der jeweiligen GUI-Engine ausgeführt werden, da dieser Thread nicht auf sichere weise freigegeben werden kann, um auf die Komponente zu warten.
`hidden`	Legt fest, ob auch nach unsichtbaren Komponenten gesucht wird, was z.B. für Menüeinträge sinnvoll ist.
Rückgabewert	Die tatsächliche Java-Komponente. Für Unterelemente wird ein Paar der Form `(component, index)` zurückgeliefert, wobei der Typ von `index` von der Art des Unterelements abhängt. Für Baumknoten ist `index` ein `javax.swing.tree.TreePath` Objekt, für Zellen von Tabellen ein Paar der Form `(row, column)` und ansonsten ein Integer Wert. Spaltenindizes werden immer im Bezugssystem der Tabelle zurückgeliefert, nicht im Bezugssystem des Modells.

`List getConnectedClients()`

Liefert die Namen der aktuell verbundenen SUT-Clients.

Rückgabewert	Eine Liste mit den Namen der aktuell verbundenen SUT-Clients, eine leere Liste, falls es keine gibt.

`Map getGlobalObjects()`

Liefert die globalen Variablen des aktuellen Runcontexts.

Bei der Verarbeitung der gelieferten Objekte ist zu beachten, dass die Eigenschaften und Methoden der Objekte auch davon abhängig sind, mit welchem Skriptinterpreter die Objekte erstellt wurden.

Rückgabewert	Die globalen Variablen des aktuellen Runcontexts.

`Properties getGlobals()`

Liefert die globalen Variablen des aktuellen Runcontexts mit ihren Werten als Strings.

Rückgabewert	Die globalen Variablen des aktuellen Runcontexts mit ihren Werten als Strings.

`Map getGroupObjects(String group)`
Liefert einen Satz von geladenen Properties oder Ressourcen.
Bei der Verarbeitung der gelieferten Objekte ist zu beachten, dass die Eigenschaften und Methoden der Objekte auch davon abhängig sind, mit welchem Skriptinterpreter die Objekte erstellt wurden.

Parameter

`group`	Der Name der Gruppe von Properties oder Ressourcen.
Rückgabewert	Die für die angegebene Gruppe gebundenen Variablen oder None falls keine solche Gruppe existiert.

`Integer getInt(String varname)`
Ermittelt den Wert einer QF-Test Variable, vergleichbar mit `$(varname)`, und interpretiert ihn als Integer.

Parameter

`varname`	Der Name der Variable.
Rückgabewert	Der Wert der Variable.

`Integer getInt(String group, String name)`
Ermittelt den Wert einer QF-Test Ressource oder Property, vergleichbar mit `${group:name}`, und interpretiert ihn als Integer.

Parameter

`group`	Der Name der Gruppe.
`name`	Der Name der Ressource oder Property.
Rückgabewert	Der Wert der Ressource oder Property.

`Object getJson(String varname, boolean expand=true)`
Interpretiert den Wert der übergebenen QF-Test Variablen, vergleichbar mit `$(varname)`, als JSON-Serialisierung und gibt diese als Objekt zurück.

Parameter

`varname`	Der Name der Variablen.
`expand`	Legt fest, ob die Variable rekursiv expandiert werden soll, siehe Der Parameter `expand`(1058).
Rückgabewert	Das durch Deserialisierung des Variablenwertes erhaltene Objekt.

`Object getJson(String group, String name, boolean expand=true)`
Interpretiert den Wert der übergebenen QF-Test Ressource oder Property, vergleichbar mit `${group:name}`, als JSON-Serialisierung und gibt diese als Objekt zurück.

Parameter

`group`	Der Name der Gruppe.
`name`	Der Name der Ressource oder Property.
`expand`	Legt fest, ob die Variable rekursiv expandiert werden soll, siehe Der Parameter `expand`(1058).
Rückgabewert	Das durch Deserialisierung der Ressource oder Property erhaltene Objekt.

`Object getLastComponent()`
Nur SUT. Liefert die letzte Komponente, die von QF-Test für einen Event, einen Check oder eine sonstige Operation adressiert wurde. Ein Aufruf von `rc.getComponent()` hat hierauf keinen Einfluss.

Rückgabewert	Die letzte von QF-Test adressierte Komponente.

`Exception getLastException()`
Nur Server. Gibt die letzte Exception zurück, die während des Testlaufs geworfen wurde, unabhängig davon, ob und wie diese gefangen wurde. In den meisten Situationen ist `getCaughtException` die bessere Variante.

Rückgabewert	Die zuletzt geworfene Exception.

`Object getLastItem()`
Nur SUT. Liefert das letzte Unterelement, das von QF-Test für einen Event, einen Check oder eine sonstige Operation adressiert wurde. Ein Aufruf von `rc.getComponent()` hat hierauf keinen Einfluss.

Rückgabewert	Das letzte von QF-Test adressierte Unterelement.

`Map getLocalObjects(nonEmpty=false)`
Liefert die innersten lokalen Variablen des aktuellen Runcontexts oder innerhalb einer Prozedur die Parameter der Aufrufs, wenn zuvor keine lokalen Variablen in der Prozedur gesetzt wurden. Dann kann die Methode mit `nonEmpty=true` in einer Prozedur genutzt werden, um die Parameter der Aufrufs zu erhalten, vergleichbar mit Keyword-Argumenten in Jython oder Groovy.
In der interaktiven Testausführung sind diese Variablen im Debugging-Modus gut zu erkennen: Im unteren (rechten) Bereich des QF-Test Fensters wird dann die Tabelle mit den Variablendefinitionen angezeigt. Die zurückgegeben innersten Variablen befinden sich bei `nonEmpty=false` immer in der obersten Zeile der Tabelle und bei `nonEmpty=true` in der ersten Zeile von oben, bei der die Anzahl der Definitionen ungleich null ist, oder es sich um einen Prozeduraufruf handelt.
Bei der Verarbeitung der gelieferten Objekte ist zu beachten, dass die Eigenschaften und Methoden der Objekte auch davon abhängig sind, mit welchem Skriptinterpreter die Objekte erstellt wurden.

Parameter

`nonEmpty`	Falls true, wird der erste nicht-leere Satz von Variablen geliefert, andernfalls immer der innerste Satz, auch wenn dieser leer ist.
Rückgabewert	Die innersten lokalen Variablen des aktuellen Runcontexts.

`Properties getLocals(nonEmpty=false)`
Liefert die innersten lokalen Variablen des aktuellen Runcontexts oder innerhalb einer Prozedur die Parameter der Aufrufs, wenn zuvor keine lokalen Variablen in der Prozedur gesetzt wurden. Dann kann die Methode mit `nonEmpty=true` in einer Prozedur genutzt werden, um die Parameter der Aufrufs zu erhalten, ähnlich wie `getLocalObjects`. Die Werte werden als Strings geliefert.
In der interaktiven Testausführung sind diese Variablen im Debugging-Modus gut zu erkennen: Im unteren (rechten) Bereich des QF-Test Fensters wird dann die Tabelle mit den Variablendefinitionen angezeigt. Die zurückgegeben innersten Variablen befinden sich bei `nonEmpty=false` immer in der obersten Zeile der Tabelle und bei `nonEmpty=true` in der ersten Zeile von oben, bei der die Anzahl der Definitionen ungleich null ist, oder es sich um einen Prozeduraufruf handelt.

Parameter

`nonEmpty`	Falls true, wird der erste nicht-leere Satz von Variablen geliefert, andernfalls immer der innerste Satz, auch wenn dieser leer ist.
Rückgabewert	Die innersten lokalen Variablen des aktuellen Runcontexts als Strings.

Number getNum(String varname)
Ermittelt den Wert einer QF-Test Variable, vergleichbar mit `$(varname)`, und interpretiert ihn als Zahl, d.h als int oder float für Jython bzw. Integer oder BigDecimal für Groovy.

Parameter

varname	Der Name der Variable.
Rückgabewert	Der Wert der Variable.

Number getNum(String group, String name)
Ermittelt den Wert einer QF-Test Ressource oder Property, vergleichbar mit `${group:name}`, und interpretiert ihn als Zahl, d.h als int oder float für Jython bzw. Integer oder BigDecimal für Groovy.

Parameter

group	Der Name der Gruppe.
name	Der Name der Ressource oder Property.
Rückgabewert	Der Wert der Ressource oder Property.

Object getObj(String varname, boolean expand=true)
Ermittelt den Wert einer QF-Test Ressource oder Property, vergleichbar mit `$(varname)`, und gibt das in der Property gespeicherte Objekt zurück.
Bei der Verarbeitung der gelieferten Objekte ist zu beachten, dass die Eigenschaften und Methoden der Objekte auch davon abhängig sind, mit welchem Skriptinterpreter die Objekte erstellt wurden.

Parameter

varname	Der Name der Variable.
expand	Legt fest, ob die Variable rekursiv expandiert werden soll, siehe Der Parameter `expand`(1058).
Rückgabewert	Der Objekt-Wert der Variable.

Object getObj(String group, String name, boolean expand=true)
Ermittelt den Wert einer QF-Test Ressource oder Property, vergleichbar mit `${group:name}`, und gibt das in der Property gespeicherte Objekt zurück.
Bei der Verarbeitung der gelieferten Objekte ist zu beachten, dass die Eigenschaften und Methoden der Objekte auch davon abhängig sind, mit welchem Skriptinterpreter die Objekte erstellt wurden.

Parameter

group	Der Name der Gruppe.
name	Der Name der Ressource oder Property.
expand	Legt fest, ob die Variable rekursiv expandiert werden soll, siehe Der Parameter `expand`(1058).
Rückgabewert	Der Objekt-Wert der Ressource oder Property.

`Object getOption(String name)`
Liest den Wert einer Option zur Laufzeit. Diese Methode ist eher der vollständigkeit halber vorhanden, Sie werden diese vermutlich nicht brauchen. Für den naheliegenden Anwendungsfall, den Wert einer Option nach einer Änderung mittels `setOption` wieder zurückzusetzen, sollten Sie stattdessen `unsetOption` verwenden, da auf Skript-Ebene gesetzte Optionen den interaktiv im Optionen-Dialog eingestellten Wert verdecken. Für temporäre Änderungen einer Option ist `pushOption` / `popOption` am besten geeignet.

Parameter

`name`	Der Name der Option, eine Konstante aus der Klasse `Options`, welche in Jython- und Groovy-Skripten automatisch importiert ist. Die Namen der Optionen, die auf diese Weise gelesen werden können, sind in Kapitel 41(482) dokumentiert.
Rückgabewert	Der aktuelle Wert der Option.

`Object getOverrideElement(String id)`

Nur SUT. Liefert die überschriebene Zielkomponente für die angegebene ID.

Parameter

`id`	Die QF-Test ID oder SmartID, für welche die Komponente überschrieben wurde.
Rückgabewert	Das vorher für die angegebene ID registrierte GUI Element. None/null wenn kein GUI Element registriert wurde oder es nicht mehr gültig ist.

`Pattern getPattern(String varname, boolean expand=true)`
Ermittelt den Wert einer QF-Test Variable und interpretiert ihn als regulären Ausdruck (vgl. Abschnitt 49.3(1023)). Im Unterschied zu `rc.getStr` oder `rc.getInt` wird kein `String` oder `Integer` geliefert sondern ein Java-Pattern-Objekt. Beispiel für den Vergleich einer Zeichenkette mit einem vorgegebenen Muster: `rc.check(rc.getPattern("myRegExp").matcher(rc.getStr("myString")).matches(), "sample check")`

Parameter

`varname`	Der Name der Variable.
`expand`	Legt fest, ob die Variable rekursiv expandiert werden soll, siehe Der Parameter `expand`(1058).
Rückgabewert	Ein Java-Pattern-Object mit dem Wert der Variable als regulärem Ausdruck.

`Pattern getPattern(String group, String name, boolean expand=true)`

Ermittelt den Wert einer QF-Test Ressource oder Property und interpretiert ihn als regulären Ausdruck (vgl. Abschnitt 49.3(1023)). Im Unterschied zu `rc.getStr` oder `rc.getInt` wird kein `String` oder `Integer` geliefert sondern ein Java-Pattern-Objekt. Beispiel für den Vergleich einer Zeichenkette mit einem vorgegebenen Muster: `rc.check(rc.getPattern("groupname", "myRegExp").matcher(rc.getStr("myString")).matches(), "sample check")`

Parameter

`group`	Der Name der Gruppe.
`name`	Der Name der Ressource oder Property.
`expand`	Legt fest, ob die Variable rekursiv expandiert werden soll, siehe Der Parameter `expand`(1058).
Rückgabewert	Ein Java-Pattern-Object mit dem Wert der Ressource oder Property als regulärem Ausdruck.

`Properties getProperties(String group)`

Liefert einen Satz von geladenen Properties oder Ressourcen mit ihren Werten als Strings.

Parameter

`group`	Der Name der Gruppe von Properties oder Ressourcen.
Rückgabewert	Die für die angegebene Gruppe gebundenen Variablen mit ihren Werten als Strings oder None falls keine solche Gruppe existiert.

`String getPropertyGroupNames()`

Listet alle benutzerdefinierten Propertygruppen auf. Die Rückgabe ist alphabetisch sortiert.

Rückgabewert	Eine Zeichenkette welche alle vom benutzerdefinierten Propertygruppen auflistet. Die Namen der definierten Propertygruppen sind alphabetisch sortiert und durch Zeilenumbrüche getrennt.

String getStr(String varname, boolean expand=true)
Ermittelt den Wert einer QF-Test Variable und interpretiert ihn als Zeichenkette. In Jython-Skripten hat es gegenüber `$(varname)` den Vorteil, dass keine Probleme mit '\u'-Sequenzen auftreten, die Jython als Unicode-Zeichen zu interpretieren versucht und sich beschwert, wenn die Syntax nicht stimmt (siehe auch Jython Strings und Zeichenkodierung(201)).

Parameter

varname	Der Name der Variable.
expand	Legt fest, ob die Variable rekursiv expandiert werden soll, siehe Der Parameter `expand`(1058).
Rückgabewert	Der Wert der Variable als String.

String getStr(String group, String name, boolean expand=true)
Ermittelt den Wert einer QF-Test Ressource oder Property, und interpretiert ihn als Zeichenkette. In Jython-Skripten hat es gegenüber `${group:name}` den Vorteil, dass keine Probleme mit '\u'-Sequenzen auftreten, die Jython als Unicode-Zeichen zu interpretieren versucht und sich beschwert, wenn die Syntax nicht stimmt (siehe auch Jython Strings und Zeichenkodierung(201)).

Parameter

group	Der Name der Gruppe.
name	Der Name der Ressource oder Property.
expand	Legt fest, ob die Variable rekursiv expandiert werden soll, siehe Der Parameter `expand`(1058).
Rückgabewert	Der Wert der Ressource oder Property als String.

String id(String id)
Liefert die QF-Test ID der angegebenen Komponente zurück, also immer den Parameter selbst. Diese Methode sollte in Skripten zur Kennzeichnung von QF-Test IDs verwendet werden, so dass diese Referenzen nach Verschieben bzw. Änderung der verwendeten QF-Test IDs angepasst werden können.

Parameter

id	Die QF-Test ID der Komponente.
Rückgabewert	Die QF-Test ID der Komponente.

boolean isOptionSet(String name)

Prüft, ob eine Option per Skript gesetzt wurde.

Parameter

name	Der Name der Option, eine Konstante aus der Klasse `Options`, welche in Jython- und Groovy-Skripten automatisch importiert ist. Die Namen der Optionen, die auf diese Weise gelesen werden können, sind in Kapitel 41[482] dokumentiert.
Rückgabewert	True falls die Option gesetzt wurde, andernfalls false.

boolean isResetListenerRegistered(ResetListener listener)

Nur Server. Überprüft, ob ein ResetListener registriert wurde.

Parameter

listener	Den zu überprüfenden ResetListener.
Rückgabewert	True wenn der ResetListener registriert wurde, sonst False.

void logDiagnostics(String client)

Nur Server. Schreibt Event-Informationen, die für eine mögliche Fehlerdiagnose im SUT Client zwischengespeichert wurden, in das Protokoll.

Parameter

client	Der Name des SUT Clients, von dem die Informationen abgeholt werden sollen.

void logError(String msg, boolean nowrap=false)

Schreibt eine benutzerdefinierte Fehlermeldung in das Protokoll.

Parameter

msg	Die Meldung.
nowrap	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen.

void logImage(ImageRep image, String title=None, boolean dontcompactify=false, boolean report=false)

Schreibt ein `ImageRep` (Abschnitt 54.9.1[1233]) Objekt in das Protokoll.

Parameter

title	Ein optionaler Titel für das Abbild.
image	Das `ImageRep` Objekt.
dontcompactify	Falls true wird das Abbild nicht aus kompakten Protokollen entfernt.
report	Falls true wird das Abbild im Report angezeigt (impliziert dontcompactify).

`void logMessage(String msg, boolean dontcompactify=false, boolean report=false, boolean nowrap=false)`

Schreibt eine Meldung in das Protokoll.

Parameter

`msg`	Die Meldung.
`dontcompactify`	Falls true wird die Meldung nicht aus kompakten Protokollen entfernt
`report`	Falls true wird die Meldung im Report aufgeführt.
`nowrap`	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen.

`void logWarning(String msg, boolean report=true, boolean nowrap=false)`

Schreibt eine benutzerdefinierte Warnmeldung in das Protokoll.

Parameter

`msg`	Die Meldung.
`report`	Falls true (default) wird die Warnung im Report aufgeführt. Sie können diese spezielle Warnung vom Report ausschließen, indem Sie diesen Parameter auf false setzen.
`nowrap`	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen.

`void overrideElement(String id, Component com)`

Nur SUT. Überschreibt die Zielkomponente für die Wiedererkennung eines GUI Elements mit der angegebenen ID. Bei einem Zugriff auf diese QF-Test ID oder SmartID ignoriert QF-Test alle zugehörigen Informationen und verwendet direkt das registrierte Element.
Ungültig gewordene Komponenten werden automatisch deregistriert.

Parameter

`id`	Die QF-Test ID oder SmartID der zu überschreibenden Komponente.
`com`	Die Komponente, welche angesprochen werden soll. None/null um zum normalen Mechanismus zurückzukehren.

void popOption(String name)

Negiert einen vorangegangen `pushOption` Aufruf.

Parameter

name	Der Name der zurückzusetzenden Option, eine Konstante aus der Klasse `Options`, welche in Jython- und Groovy-Skripten automatisch importiert ist. Die Namen der Optionen, die auf diese Weise gesetzt werden können, sind in Kapitel 41[482] dokumentiert.

void pushOption(String name, object value)

Der Aufruf setzt den Wert einer Option zur Laufzeit, analog zu `setOption`. Im Gegensatz zu letzterem wird für jeden verschachtelten Aufruf der ursprüngliche Wert gespeichert, um mittels `popOption` wiederhergestellt zu werden. Die `pushOption` und `popOption` Aufrufe, die am besten in einer Try[704] / Finally[711] Kombination genutzt werden, sind ideal für Prozeduren geeignet, um eine Option temporär auf einen bestimmten Wert zu setzen, ohne einen eventuell früher erfolgten `setOption` Aufruf zu negieren.

Parameter

name	Der Name der Option, eine Konstante aus der Klasse `Options`, welche in Jython- und Groovy-Skripten automatisch importiert ist. Die Namen der Optionen, die auf diese Weise gesetzt werden können, sind in Kapitel 41[482] dokumentiert.
value	Der zu setzende Wert, üblicherweise ein Boolean, eine Zahl oder eine Konstante aus der `Options` Klasse für solche Optionen, die über eine Auswahlliste gesetzt werden. Für Optionen wie den Hotkey für Wiedergabe unterbrechen (”Keine Panik”-Taste), deren Wert ein Tastenkürzel ist, muss diese Kürzel als Text wie ”F12” oder ”Shift-F6” angegeben werden. Mögliche Modifier sind ”Shift”, ”Control” oder ”Ctrl”, ”Alt” und ”Meta”, sowie deren Kombinationen. Der angegebenen Taste wird ein ”VK_” vorangestellt und ihr Wert dann der Klasse `java.awt.event.KeyEvent` entnommen. Groß-/Kleinschreibung ist für beide irrelevant, so dass auch ”shift-alt-enter” funktioniert.

void removeResetListener(ResetListener listener)

Nur Server. Entfernt einen ResetListener.

Parameter

listener	Den zu entfernenden ResetListener.

void removeTestRunListener(TestRunListener listener)

Entfernt einen `TestRunListener` aus dem aktuellen Runcontext.

Parameter

listener	Der zu entfernende Listener.

void resetDependencies(String namespace=None)

Setzt den Stapel von Abhängigkeiten zurück, ohne Aufräumsequenzen auszuführen.

Parameter

namespace	Ein optionaler Namensraum für die Abhängigkeiten.

void resolveDependency(String dependency, String namespace=None, Map parameters=None)

Löst eine Abhängigkeit(630) auf.

Parameter

dependency	Der vollständige Name der Abhängigkeit.
namespace	Ein optionaler Namensraum für die Abhängigkeit.
parameters	Die Parameter für die Abhängigkeit, ein Dictionary. Die Schlüssel und Werte können beliebige Objekte sein. Diese werden beim Aufruf in Zeichenketten umgewandelt.

void returnValue(object value)

Bricht die Ausführung einer Prozedur ab und gibt den angegebenen Wert zurück.

Parameter

value	Ein beliebiger Wert für die Variable. Wird der Wert aus einem SUT-Skript zurückgegeben, so wird das Objekt serialisiert. Wenn dies mehr als 25 MB Speicherplatz beansprucht wird an Stelle des Objektes sein String übergeben.

void rollbackAllDependencies()

Baut die Stapel von Abhängigkeiten aus allen Namensräumen ab. Dies geschieht in der umgekehrten Reihenfolge wie sie erstmalig angelegt wurden, mit der Ausnahme, dass der Stapel aus dem allgemeinen Namensraum immer zuletzt abgebaut wird.

void rollbackDependencies(String namespace=None)

Baut den Stapel von Abhängigkeiten ab.

Parameter

namespace	Ein optionaler Namensraum für die Abhängigkeiten.

void setGlobal(String name, object value)

Definiert eine globale QF-Test Variable.

Parameter

name	Der Name der Variable.
value	Ein beliebiger Wert für die Variable. Der Wert `None` führt zum Löschen der Variablen. Beim Zugriff auf die Variable aus einem SUT-Skript wird das Objekt serialisiert. Wenn dies mehr als 25 MB Speicherplatz beansprucht wird an Stelle des Objektes sein String übergeben.

void setGlobalJson(String name, Object value)

Definiert eine globale QF-Test Variable, wobei das übergebene Objekt zu einem JSON-String serialisiert wird.

Parameter

name	Der Name der Variable.
value	Ein beliebiger Wert für die Variable. Er wird automatisch in eine JSON-Zeichenkette konvertiert. Der Wert `None` führt zum Löschen der Variablen.

void setGroupObject(String group, String name, Object value)

Setzt den Wert eines Objektes (Ressource oder Property) in einer Gruppe.

Parameter

group	Der Name der Gruppe. Falls noch nicht vorhanden wird die Gruppe neu angelegt.
name	Der Name des Objekts (Ressource oder Property).
value	Ein beliebiger Wert für das Gruppenobjekt (die "Property"). Der Wert `None` führt zum Löschen des Objekts. Diese Methode funktioniert auch für die speziellen Gruppen 'env' und 'system'. Auf diesem Weg können Environment-Variablen oder System-Properties definiert werden. Werte in anderen speziellen Gruppen (zum Beispiel 'qftest') können meist nicht überschrieben werden. In diesem Fall wird dann eine `ReadOnlyPropertyException`[962] geworfen. Alias von `setProperty`.

`void setLocal(String name, Object value)`

Definiert eine lokale QF-Test Variable.

Parameter

`name`	Der Name der Variable.
`value`	Ein beliebiger Wert für die Variable. Der Wert `None` führt zum Löschen der Variablen. Beim Zugriff auf die Variable aus einem SUT-Skript wird das Objekt serialisiert. Wenn dies mehr als 25 MB Speicherplatz beansprucht wird an Stelle des Objektes sein String übergeben.

`void setLocalJson(String name, Object value)`

Definiert eine lokale QF-Test Variable, wobei das übergebene Objekt zu einem JSON-String serialisiert wird.

Parameter

`name`	Der Name der Variable.
`value`	Ein beliebiger Wert für die Variable. Er wird automatisch in eine JSON-Zeichenkette konvertiert. Der Wert `None` führt zum Löschen der Variablen.

`void setOption(String name, object value)`
Setzt den Wert einer Option zur Laufzeit. Dieser Wert hat Vorrang vor dem Wert, der aus der Konfigurationsdatei gelesen oder im Optionen-Dialog eingestellt wurde und wird selbst weder im Dialog angezeigt, noch in eine Konfigurationsdatei gespeichert. Der ursprüngliche Wert aus der Konfiguration kann via `unsetOption` wieder hergestellt werden. Der Wert eines eventuell vorhergegangenen Aufrufs von `setOption` wird dagegen überschrieben. Soll dieser nicht verloren gehen, muss stattdessen `pushOption` / `popOption` verwendet werden.

Parameter

`name`	Der Name der Option, eine Konstante aus der Klasse `Options`, welche in Jython- und Groovy-Skripten automatisch importiert ist. Die Namen der Optionen, die auf diese Weise gesetzt werden können, sind in Kapitel 41 (482) dokumentiert.
`value`	Der zu setzende Wert, üblicherweise ein Boolean, eine Zahl oder eine Konstante aus der `Options` Klasse für solche Optionen, die über eine Auswahlliste gesetzt werden. Für Optionen wie den Hotkey für Wiedergabe unterbrechen (”Keine Panik”-Taste), deren Wert ein Tastenkürzel ist, muss diese Kürzel als Text wie ”F12” oder ”Shift-F6” angegeben werden. Mögliche Modifier sind ”Shift”, ”Control” oder ”Ctrl”, ”Alt” und ”Meta”, sowie deren Kombinationen. Der angegebenen Taste wird ein ”VK_” vorangestellt und ihr Wert dann der Klasse `java.awt.event.KeyEvent` entnommen. Groß-/Kleinschreibung ist für beide irrelevant, so dass auch ”shift-alt-enter” funktioniert.

void setProperty(String group, String name, object value)

Setzt den Wert einer Ressource oder Property in einer Gruppe.

Parameter

group	Der Name der Gruppe. Falls noch nicht vorhanden wird die Gruppe neu angelegt.
name	Der Name der Ressource oder Property.
value	Ein beliebiger Wert für die Property. Der Wert `None` führt zum Löschen der Property. Diese Methode funktioniert auch für die speziellen Gruppen 'env' und 'system'. Auf diesem Weg können Environment-Variablen oder System-Properties definiert werden. Werte in anderen speziellen Gruppen (zum Beispiel 'qftest') können meist nicht überschrieben werden. In diesem Fall wird dann eine `ReadOnlyPropertyException`[962] geworfen.

void skipTestCase()

Beendet die Ausführung des aktuellen Testfalls und markiert diesen als übersprungen.

void skipTestSet()

Beendet die Ausführung des aktuellen Testfallsatzes und markiert diesen als übersprungen.

void stopTest()

Beendet den aktuellen Testlauf.

void stopTestCase(boolean expectedFail=false)

Beendet die Ausführung des aktuellen Testfalls.

Parameter

expectedFail	Falls true, werden eventuelle Fehler in diesem Testfall als erwartete Fehler behandelt.

void stopTestSet()

Beendet die Ausführung des aktuellen Testfallsatzes.

void syncThreads(String name, int timeout, int count=-1, boolean throw=true, int remote=0)
Nur Server. Synchronisiert eine Anzahl von parallelen Threads für Lasttests. Der aktuelle Thread wird blockiert, bis alle Threads den Synchronisationspunkt erreicht haben oder die Wartezeit überschritten wird. In letzterem Fall wird eine `TestException`(958) geworfen oder ein Fehler ausgegeben.
Parameter

name	Ein Identifikator für den Synchronisationspunkt.
timeout	Die maximale Wartezeit in Millisekunden.
count	Die Zahl der Threads auf die gewartet wird. Standardwert -1 bedeutet alle Threads in der aktuellen QF-Test Instanz.
throw	Entscheidet ob bei Überschreitung der Wartezeit eine Exception geworfen (Standard) oder eine Fehlermeldung ausgegeben wird.
remote	Die Zahl der QF-Test Instanzen - eventuell auf unterschiedlichen Rechnern - die synchronisiert werden sollen. Standardwert 0 bedeutet nur interne Synchronisation.

void toServer(...)
Nur SUT. Setzt globale Variablen im entsprechenden Interpreter von QF-Test. Von einem Groovy-SUT-Skript können somit zum Beispiel globale Variablen im Groovy-Interpreter von QF-Test gesetzt werden.
Folgende Arten von Argumenten sind möglich:

Ein String
Dieser wird als Name einer globalen Variablen im lokalen Interpreter aufgefasst. Die Variable des selben Namens im Interpreter von QF-Test wird auf ihren Wert gesetzt.

Ein Dictionary mit String Schlüsseln
Für jeden Schlüssel im Dictionary wird einer globalen Variable dieses Namens der entsprechende Wert aus dem Dictionary zugewiesen.

Ein Keyword Argument der Form `name=value`
Die globale Variable namens `name` wird auf den Wert `value` gesetzt.

`void toSUT(String client, ...)`
Nur Server. Setzt globale Variablen im entsprechenden Interpreter des SUT. Von einem Groovy-Server-Skript können somit zum Beispiel globale Variablen im Groovy-Interpreter des SUT gesetzt werden.
Mit Ausnahme von `client` sind folgende Arten von Argumenten möglich:

Ein String
Dieser wird als Name einer globalen Variablen im lokalen Interpreter aufgefasst. Die Variable des selben Namens im Interpreter des SUT wird auf ihren Wert gesetzt.

Ein Dictionary mit String Schlüsseln
Für jeden Schlüssel im Dictionary wird einer globalen Variable dieses Namens der entsprechende Wert aus dem Dictionary zugewiesen.

Ein Keyword Argument der Form `name=value`
Die globale Variable namens `name` wird auf den Wert `value` gesetzt.

Parameter

`client`	Der Name des SUT Clients.

`void unsetOption(String name)`
Stellt den ursprünglichen Wert einer Option wieder her indem der Wert aus einem vorhergehenden Aufruf von `setOption` gelöscht wird.
Parameter

`name`	Der Name der zu löschenden Option, eine Konstante aus der Klasse `Options`, welche in Jython- und Groovy-Skripten automatisch importiert ist. Die Namen der Optionen, die auf diese Weise gesetzt werden können, sind in Kapitel 41[482] dokumentiert.

`RunContext withDefault(Object defaultResult)`
Erzeugt ein neues Runcontext-Objekt, bei welchem alle lesenden Zugriffe auf Variablen bzw. Resourcen/Properties in Gruppen keine `UnboundVariableException`[962] oder `MissingPropertyException`[961] werfen, sondern statt dessen im Fall einer fehlenden Variable den angegebenen Standard-Wert zurückgeben. Die übrigen Methoden und Eigenschaften des `rc`-Objekten verhalten sich unverändert.
Parameter

`defaultResult`	Das Objekt, welches an Stelle des nicht vorhandenen Variablen-Wertes zurückgegeben wird.
Rückgabewert	Ein neues Runcontext (`rc`)-Objekt, welches den entsprechenden Standard-Wert für Variablen-Zugriffe gesetzt hat.

50.5.1 Der Parameter `expand`

Die Methoden `getStr`, `getObj`, `getInt`, `getNum`, `getBool`, `getPattern` und `getJson` unterstützen einen optionalen boolschen Parameter `expand`. Dieser legt fest, ob die Variable bei der Auswertung rekursiv expandiert werden soll, das heißt, ob eine in der String-Darstellung des Variablenwerts enthaltene Zeichenkette, die (zufällig) die QF-Test Variablensyntax $(irgendwelche_Zeichen) hat, als Variable expandiert oder als einfacher Text behandelt werden soll. Ist dieser nicht angegeben oder `null`, so führt QF-Test die Expansion (auch rekursiv) genau dann durch, wenn der Variablenwert ein String ist. Um Probleme zu vermeiden werden bestimmte Zeichenketten, zum Beispiel das Ergebnis eines Text auslesen(839) Knotens, die Client-Ausgabe aus der speziellen Gruppe `${qftest:client.output.<name>}` oder der Rückgabewert der Standard-Prozedur `qfs.utils.readTextFromFile` nicht automatisch expandiert, sondern nur, wenn der `expand` Parameter explizit auf `true` gesetzt wurde.

Wenn Sie diesen Parameter angeben wollen, ohne eine Gruppe zu spezifizieren, müssen Sie die Python Keyword Syntax verwenden, um Konflikte mit zum Beispiel der Methode `getStr(String group, String name)` zu vermeiden, also `rc.getStr("var", expand=0)` an Stelle von `rc.getStr("var", 0)` - andernfalls würde in der Gruppe `var` nach der Property `0` gesucht.

Beispiel

Wenn die QF-Test Variablen die folgenden Werte besitzen,

Variablenreferenz	Wert
$(simplevar)	foo
$(nestedvar)	Ein Wert: $(simplevar)
${group:var}	Ein Wert: $(simplevar)

Tabelle 50.1: QF-Test Variable für nachfolgendes Bespielskript

hat der Parameter `expand` folgende Wirkung:

```
print rc.getStr("nestedvar", expand=True)  # "Ein Wert: foo"
print rc.getStr("nestedvar", expand=False) # "Ein Wert: $(simplevar)"
print rc.getStr("group", "var", True)      # "Ein Wert: foo"
print rc.getStr("group", "var", False)     # "Ein Wert: $(simplevar)"
```

Beispiel 50.2: Verwendung des Parameters `expand` (Jython script)

50.6 Das qf Modul

In manchen Fällen ist kein Runcontext verfügbar, insbesondere wenn eines der in den folgenden Abschnitten beschriebenen Interfaces zur Erweiterung von QF-Test implementiert wird. Das Modul `qf` ermöglicht Logging auch in diesen Fällen und bietet weitere allgemein hilfreiche Methoden, die keinen Runcontext benötigen. Es folgt eine alphabetische Aufstellung aller Methoden des `qf` Moduls. Sofern nicht anders angegeben sind die Methoden in Jython und Groovy sowohl für Server-Skript als auch SUT-Skript Knoten verfügbar.

Hinweis

Die Groovy-Syntax für Keyword-Parameter unterscheidet sich von Jython. Groovy verwendet ':' statt '='. Dies ist besonders tückisch, weil z.B. `qf.logMessage("bla", report=true)` durchaus legaler Groovy-Code ist, der allerdings nicht den gewünschten Effekt hat. Das '=' ist hierbei eine Zuweisung mit dem Wert true, der dann ganz einfach als zweiter Parameter übergeben wird, so dass der Aufruf äquivalent ist zu `qf.logMessage("bla", true)`. Hierbei wird true aber für `dontcompactify` statt `report` verwendet. Die korrekte Version für Groovy lautet `qf.logMessage("bla", report:true)`.

`Pattern asPattern(String regexp)`

Diese Methode interpretiert die Eingabe als regulären Ausdruck (vgl. Abschnitt 49.3[1023]) und liefert das entsprechende Java-Pattern-Objekt zurück. Die möglichen Eingabewerte sind in der Java-API des Pattern-Objekts definiert.

Parameter

`regexp`	Der reguläre Ausdruck
Rückgabewert	Ein Pattern-Objekt, welches für Stringvergleiche genutzt werden kann.

`String getClassName(Object objectOrClass)`

Liefert den qualifizierten Namen der Klasse eines Java-Objekts bzw. einer Java-Klasse. Hauptsächlich für Jython sinnvoll, wo das Ermitteln des Namens einer Klasse richtig lästig werden kann.

Parameter

`objectOrClass`	Das Java-Objekts oder die Klasse, deren Name ermittelt werden soll.
Rückgabewert	Der Name der Klasse oder None falls kein Java-Objekt übergeben wird.

Object getProperty(Object object, String name)

Liest eine Property für ein Objekt, die vorher via `setProperty` gesetzt wurde.

Parameter

object	Das Objekt, für das die Property gelesen werden soll.
name	Der Name der Property.
Rückgabewert	Der Wert der Property.

boolean isInstance(Object object, String className)

Diese Methode ist eine Alternative zu `instanceof` in Groovy oder `isinstance()` in Jython, die gezielt nur Namen von Klassen oder Interfaces vergleicht und so Probleme mit unterschiedlichen ClassLoadern vermeidet.

Parameter

object	Das zu prüfende Object.
className	Der Name der Klasse oder des Interfaces auf das getestet wird.
Rückgabewert	True, wenn das Objekt einer Instanz der angegeben Klasse ist oder das angegebene Interface implementiert.

void logError(String msg, boolean nowrap=false)

Schreibt eine benutzerdefinierte Fehlermeldung in das Protokoll. Falls ein Runcontext verfügbar ist wird dieser genutzt und die Meldung sofort geschrieben. Andernfalls wird sie gepuffert und bei nächster Gelegenheit in das Protokoll übernommen.

Parameter

msg	Die Meldung.
nowrap	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen. Falls die Meldung gepuffert werden muss, hat dieser Parameter keine Wirkung.

void logMessage(String msg, boolean dontcompactify=false, boolean report=false, boolean nowrap=false)

Schreibt eine Meldung in das Protokoll. Falls ein Runcontext verfügbar ist wird dieser genutzt und die Meldung sofort geschrieben. Andernfalls wird sie gepuffert und bei nächster Gelegenheit in das Protokoll übernommen.

Parameter

msg	Die Meldung.
dontcompactify	Falls true wird die Meldung nicht aus kompakten Protokollen entfernt
report	Falls true wird die Meldung im Report aufgeführt.
nowrap	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen. Falls die Meldung gepuffert werden muss, hat dieser Parameter keine Wirkung.

void logWarning(String msg, boolean report=true, boolean nowrap=false)

Schreibt eine benutzerdefinierte Warnmeldung in das Protokoll. Falls ein Runcontext verfügbar ist wird dieser genutzt und die Meldung sofort geschrieben. Andernfalls wird sie gepuffert und bei nächster Gelegenheit in das Protokoll übernommen.

Parameter

msg	Die Meldung.
report	Falls true (default) wird die Warnung im Report aufgeführt. Sie können diese spezielle Warnung vom Report ausschließen, indem Sie diesen Parameter auf false setzen.
nowrap	Falls true werden die Zeilen der Meldung im Report nicht umgebrochen. Sinnvoll für potentiell lange Meldungen. Falls die Meldung gepuffert werden muss, hat dieser Parameter keine Wirkung.

void print(Object object, ...)

Schreibt eine Zeichenkette oder die String-Repräsentation eines Objektes in das Terminal. Sind mehrere Objekte angeben, werden deren Repräsentationen mit Leerzeichen getrennt aneinander gehängt. Im Gegensatz zu einem einfachen `print`-Ausdruck wird der Text nicht über die Standard-Ausgabe geleitet.

Parameter

object	Das Objekt, welches ausgegeben werden soll.

`void println(Object object)`
Schreibt eine Zeichenkette oder die String-Repräsentation eines Objektes in das Terminal und beginnt eine neue Zeile. Sind mehrere Objekte angeben, werden deren Repräsentationen mit Leerzeichen getrennt aneinander gehängt. Im Gegensatz zu einem einfachen `println`-Ausdruck wird der Text nicht über die Standard-Ausgabe geleitet.

Parameter

`object`	Das Objekt, welches ausgegeben werden soll.

`void setProperty(Object object, String name, Object value)`
Setzt eine beliebige Property für ein Objekt. Im Fall einer Swing, JavaFX, SWT oder Web Komponente wird der Wert in den Anwender-Daten via `putClientProperty`, `setData` oder `setProperty` gesetzt. Für alles andere kommt eine `WeakHashMap` zum Einsatz. In keinem Fall wird die Garbage Collection des Objekts durch die Property verhindert.

Parameter

`object`	Das Objekt, für das die Property gesetzt werden soll.
`name`	Der Name der Property.
`value`	Der zu setzende Wert. Null um die Property zu löschen.

`String toString(Object object, String nullValue)`
Liefert die String-Darstellung eines Objekts. Vor allem für Jython hilfreich, aber auch für Groovy dank der Default-Konvertierung von None in den leeren String hilfreich.

Parameter

`object`	Das zu konvertierende Objekt.
`nullValue`	Der zu liefernde Wert falls das Objekt None ist, Default ist der leere String.
Rückgabewert	Jython 8-bit oder Unicode-Strings werden unverändert zurückgegeben, Java-Objekte werden via `toString` in einen String verwandelt. In Jython wird alles andere in einen 8-bit Jython-String konvertiert.

50.7 Image API

3.0+

Die Image API stellt Klassen und Interfaces bereit, um Bildschirmabbilder zu erstellen, Bilder in Dateien zu speichern oder zu aus Dateien zu laden. Es können auch eigene Bildvergleiche durchgeführt werden. Die Image API ist so konzipiert, dass im Allgemeinen keine Exceptions geworfen werden. Um dennoch auf mögliche Fehler eingehen zu können, werden Warnungen geloggt.

50.7.1 Die `ImageWrapper` Klasse

Um Bildschirmabbilder zu erstellen, können Sie die Jython Klasse ImageWrapper aus dem Modul `imagewrapper.py` verwenden. Dieses Modul wird mit dem QF-Test Installationspaket mitgeliefert.

Hier sehen Sie ein kleines Jython Beispiel, um die Verwendung der Image API darzustellen:

```
from imagewrapper import ImageWrapper
#create ImageWrapper instance
iw = ImageWrapper(rc)
#take screenshot of the whole screen
currentScreenshot = iw.grabScreenshot()
#save screenshot to a file
iw.savePng("/tmp/screenshot.png", currentScreenshot)
```

Beispiel 50.3: Image API in Jython

Das gleiche mit Groovy:

```
import de.qfs.ImageWrapper
def iw = new ImageWrapper(rc)
def currentScreenshot = iw.grabScreenshot()
iw.savePng("/tmp/screenshot.png", currentScreenshot)
```

Beispiel 50.4: Image API in Groovy

Es folgt eine alphabetische Aufstellung aller Methoden der `ImageWrapper` Klasse. Die verwendete Syntax ist ein Gemisch aus Java und Python. Python unterstützt zwar selbst keine statische Typisierung, die Parameter werden jedoch an Java weitergereicht, so dass falsche Typen Exceptions auslösen können. Folgt einem Parameter ein '='-Zeichen und ein Wert, ist dies der Defaultwert des Parameters und eine Angabe beim Aufruf ist optional.

`ImageWrapper ImageWrapper(RunContext rc)`

Konstruktor der `ImageWrapper` Klasse.

Parameter

`rc`	Der aktuell Runcontext von QF-Test.

`int getMonitorCount()`

Liefert die Anzahl aller Monitor zurück.

Rückgabewert	Die Anzahl der Monitore.

`ImageRep grabImage(Object com, int x=None, int y=None, int width=None, int height=None)`

Erstellt ein Bildschirmabbild einer spezifischen Komponente. Wenn die Parameter x, y, with und height gesetzt werden, dann wird ein Bildschirmabbild des beschriebenen Teilbereiches der Komponente erstellt.

Parameter

`com`	Die QF-Test ID der Komponente.
`x`	Die X-Koordinate der linken oberen Ecke des Teilbereiches.
`y`	Die Y-Koordinate der linken oberen Ecke des Teilbereiches.
`width`	Die Breite des Teilbereiches.
`height`	Die Höhe des Teilbereiches.
Rückgabewert	Ein `ImageRep` Objekt, welches das aktuelle Bildschirmabbild beinhaltet.

`ImageRep grabScreenshot(int x=None, int y=None, int width=None, int height=None)`

Erstellt ein Bildschirmabbild des gesamten Bildschirms. Wenn die Parameter x, y, with und height gesetzt werden, dann wird ein Bildschirmabbild des beschriebenen Teilbereiches erstellt.

Parameter

`x`	Die X-Koordinate der linken oberen Ecke des Teilbereiches.
`y`	Die Y-Koordinate der linken oberen Ecke des Teilbereiches.
`width`	Die Breite des Teilbereiches.
`height`	Die Höhe des Teilbereiches.
Rückgabewert	Ein `ImageRep` Objekt, welches das aktuelle Bildschirmabbild beinhaltet.

`ImageRep[] grabScreenshots(int monitor=None)`

Erstellt Bildschirmabbilder aller vorhandenen Monitore. Diese Methode ist von Nutzen, wenn Sie mehr als einen Monitor verwenden.

Wenn Sie ein Bildschirmabbild eines speziellen Monitors machen wollen, dann können Sie das mit Hilfe des `monitor` Parameters.

Parameter

`monitor`	Index des Monitors. Der Index für den ersten ist 0, der für den zweiten 1 etc.
Rückgabewert	Ein Feld von `ImageRep` Objekten aller Bildschirmabbilder oder das spezielle `ImageRep` Objekt, wenn der `monitor` Parameter gesetzt wurde.

`ImageRep loadPng(String filename)`
Lädt ein Bild aus einer angegebenen Datei und liefert ein `ImageRep` Objekt zurück. Die Datei muss im PNG Format sein.

Parameter

`filename`	Die Datei, in der das Bild gespeichert ist.
Rückgabewert	Ein `ImageRep` Objekt, das das geladene Bild enthält.

`void savePng(String filename, ImageRep image)`
Speichert ein angegebenes `ImageRep` Objekt in eine Datei. Das Bild wird PNG formatiert.

Parameter

`filename`	Der Pfad zu Zieldatei, wo das Bild gespeichert werden soll.
`image`	Das abzuspeichernde `ImageRep` Objekt.

50.8 Das JSON Modul

Das JSON-Modul, welches in allen Skripten ohne speziellen Import zur Verfügung steht, bersetzt einen JSON-String in eine Datenstruktur aus Maps, Listen und primitiven Typen wie Integer, Double, Boolean und String. Die Serialisierung geschieht über die Methode `stringify()`. **Anmerkung:** Verwenden Sie `rc.getJson()`, wenn Sie Daten aus einer JSON-Struktur in einer QF-Test Variable auslesen möchten.

Object parse(Object text, Object reviver=None)

Die statische Methode zerlegt eine JSON-Zeichenkette und erstellt den über das `text`-Objekt festgelegten Objektwert oder Objekt.
Wenn die Methode aus Javascript heraus gerufen wird, wird die originale JavaScript-Version von `JSON.parse()` verwendet.
Wenn ein Wert für `reviver` angegeben ist, wird der ermittelte Wert transformiert bevor er zurückgegeben wird. Dabei werden der ermittelte Wert und alle seine Eigenschaften (beginnend mit den am tiefsten verschachtelten Eigenschaften über die weniger verschachtelten bis zum Originalwert selbst) einzeln durch den `reviver` bearbeitet.

- `reviver` hat zwei Argumente: Schlüssel und Wert, also den Namen der Eigenschaft als Zeichenkette (auf bei Listen), und der Wert der Eigenschaft.
- Wenn die `reviver`-Funktion eine `NoSuchElementException` wirft, wird die entsprechende Eigenschaft aus dem Objekt gelöscht (oder bei einer Liste durch `null` ersetzt). Bei einer `UnsupportedOperationException` bleibt der Wert unverändert. Ansonsten wird der Wert der Eigenschaft auf den Rückgabewert gesetzt.
- Wenn der `reviver` nur einige Werte transformieren soll, so stellen Sie sicher, dass die nicht transformierten Werte so wie sie sind zurückgegeben werden oder eine `UnsupportedOperationException` geworfen wird. Andernfalls werden sie im Ergebnisobjekt gelöscht.

Analog zum Parameter `replacer` bei `JSON.stringify()` für `List` und `Map`, wird der `reviver` als letztes für den obersten Wert (`root value`) mit einer leeren Zeichenkette als Schlüssel und dem obersten Objekt als Wert gerufen.
Für andere gültige JSON-Werte arbeitet `reviver` analog und wird einmalig mit einer leeren Zeichenkette als Schlüssel und dem Wert selbst als Wert gerufen.
Wenn Sie einen anderen Wert vom `reviver` zurückgeben, wird der Wert den ursprünglich ermittelten Wert vollständig ersetzen. Dies gilt auch für den obersten Wert.

Parameter

text	Der `String` oder `InputStream`, der in einen JSON-String umgewandelt werden soll.
reviver	*(Optional)* Diese Funktion bzw. Closure beschreibt, wie jeder ursprünglich ermittelte Wert transformiert werden soll, bevor er zurückgegeben wird. Werte, die keine Funktion darstellen, also nicht aufrufbar sind, werden ignoriert.
Rückgabewert	Der `Map, List, String, Number, Boolean oder null`-Wert, der dem übergebenen JSON-Text entspricht.

Object stringify(Object value, Object replacer=None, Object spacer=None)

Die statische Methode konvertiert ein Objekt in eine JSON-Zeichenkette.
Wenn die Methode aus Javascript heraus gerufen wird, wird die originale JavaScript-Version von `JSON.parse()` verwendet.
Der Parameter `replacer` kann entweder eine Funktion oder ein Array sein.

- Als Array geben die einzelnen Elemente die Namen der Eigenschaften im Objekt an, die in die erstellte JSON-Zeichenkette einbezogen werden sollen. Es werden nur Text- oder numerische Werte berücksichtigt.
- Wenn eine Funktion angegeben wird, müssen zwei Parameter verarbeitet werden: Der Schlüssel und der Wert, der in Text umgewandelt werden soll.li>

Die replacer-Funktion wird auch für das initiale Objekt, das in Text umgewandelt wird, aufgerufen. In diesem Fall ist der Schlüssel eine leere Zeichenkette (""). Sie wird dann für jede Eigenschaft des umzuwandelnden Objekts oder Arrays gerufen. Der aktuelle Eigenschaftswert wird durch den Rückgabewert der replacer-Funktion ersetzt. Das heißt:

- Wenn Sie eine Zahl, Text, Booleschen Wert oder `null` zurückliefern, wird dieser Wert direkt serialisiert und als Eigenschaftswert verwendet.
- Wenn Sie eine `NoSuchElementException` werfen, wird die Eigenschaft nicht in die Ausgabe eingefügt.
- Wenn Sie ein anderes Objekt zurückgeben, wird dieses Objekt rekursiv in Text umgewandelt, wobei die replacer-Funktion für jede Eigenschaft gerufen wird.

Anmerkung: Wenn eine JSON-Zeichenkette, die mit der replacer-Funktion generiert wurde, zerlegt werden soll, ist es hilfreich, den Parameter `reviver` für die Umkehrfunktion zu verwenden.
Typischerweise verändern sich die Indizes der Array-Elemente nicht. (Auch wenn das Element ein ungültiger Wert wie zum Beispiel eine Funktion ist. In diesem Fall wird es zu `null`, wird aber nicht gelöscht.) Die Verwendung der replacer-Funktion erlaubt es, die Reihenfolge der Array-Elemente zu bestimmen, indem ein anderes Array zurückgeben wird.

Parameter

`value`	Der Wert, der in eine JSON-Zeichenkette umgewandelt werden soll.
`replacer`	*(Optional)* Eine Funktion, die den Vorgang der Textumwandlung beeinflusst oder ein Array aus Texten und Zahlen, die die Eigenschaften angeben, die in das Ergebnis übernommen werden sollen. Wenn es sich weder um eine Funktion noch um ein Array handelt, werden alle Objekteigenschaften, deren Schlüssel vom Typ `String` sind, in das Ergebnis übernommen.

space	*(Optional)* Eine Zeichenkette oder eine Zahl, die verwendet wird, um Leerzeichen sowie Einrückung, Zeichenumbrüche etc. einzufügen, um die Lesbarkeit der JSON-Zeichenkette zu verbessern. • Als Zahl gibt der Parameter an, wie viele Leerzeichen für die Einrückung verwendet werden sollen (begrenzt auf 10 Leerzeichen). Werte unter 1 geben an, dass keine Leerzeichen eingefügt werden sollen. • Als Zeichenkette wird diese (beziehungsweise ihre ersten 10 Zeichen) vor jedem geschachtelten Objekt oder Array eingefügt. • Wenn `space` kein Text oder keine Zahl ist, zum Beispiel `null`, oder nicht angegeben wurde, werden die Elemente nicht eingerückt.
Rückgabewert	Eine JSON-Zeichenkette, die den übergebenen Wert darstellt, oder `null`.

50.9 Sprechende Prüfausdrücke (Assertions)

Inspiriert von Chai.js haben wir eine eigene API für sprechende Asserts erstellt. Damit können Prüfausdrücke (Assertions) in einer formalisierten Sprache, die normale Wörter aus dem Englischen verwendet, formuliert werden. Sie kann in Groovy, JavaScript und Jython Skripten genutzt werden.

50.9.1 Motivation

Das Ziel ist, Prüfausdrücke, die in Skripten verwendet werden, besser lesbar zu machen und auch komplexe Datenstrukturen prüfen zu können. Aktuell geschieht die Überprüfung von Daten in Server und SUT Skripten im Normalfall über `rc.check()` oder über `assert` von Java. Das funktioniert gut für Basisdatentypen wie Zeichenketten. Es ist jedoch mühsam, komplexe Datenstrukturen wie strukturierte Objekte, die zum Beispiel aus einem JSON-String erzeugt wurden, damit zu prüfen. Die QF-Test Assertions API macht hier das Leben deutlich leichter.

Nachfolgend finden Sie zwei Groovy Skripte, die exemplarisch den Unterschied zwischen sprechenden Prüfausdrücken und rc.check() beziehungsweise assert() zeigen.

```
def foo = 'bar'
def beverages = [ tea: [ 'chai', 'matcha', 'oolong' ] ]
expect(foo).to.be.a('String')
foo.should.be.equal('bar')
expect(foo).to.have.lengthOf(3)
expect(beverages).to.have.property('tea').with.lengthOf(3)
```

Beispiel 50.5: Groovy-Skript mit sprechenden Asserts

```
def foo = 'bar'
def beverages = [ tea: [ 'chai', 'matcha', 'oolong' ] ]
rc.check(foo instanceof String,"")
rc.checkEqual(foo,'bar',"")
rc.checkEqual(foo.length(),3,"")
assert(beverages.tea!=null)
assert(beverages.tea.size()==3)
```

Beispiel 50.6: Datenprüfung mit QF-Test `rc.check` und Java `assert`

50.9.2 API-Dokumentation

Die QF-Test Assertions API bietet die Schnittstellen `Assert` und `expect`. In Groovy-Skripten steht auch zusätzlich `should` zur Verfügung. `expect` und `should` unterstützen Wort- bzw. Aufrufketten. Die Syntax von `Assert` ist eher die klassische Assert-Syntax wie in Java.

Das Prüfergebnis wird als erfolgreicher bzw. fehlgeschlagener Check oeder optional als Exception im Protokoll gespeichert. Zusätzlich kann das Ergebnis auch als boolescher Wert abgefragt und in einer Variable geschrieben werden. Details hierzu finden Sie in Ergebnisbehandlung(1073).

Die API-Dokumentation steht in `doc/javadoc/qfaa.zip` zur Verfügung. Darin sind alle verfügbaren Methoden für `Assert` enthalten. Diese können ebenfalls bei `expect` oder `should` (in Groovy) als Teil der Wortketten verwendet werden. Da die QF-Test Assertions API sehr ähnlich zu Chai.js ist, funktionieren die meisten Beispiele von https://www.chaijs.com/api/ auch in QF-Test. Die Methoden, die in Chai.js verfügbar sind, aber noch nicht für QF-Test implementiert wurden, sind in Abschnitt 50.9.2(1072) aufgelistet.

Für `Assert` kann eine Autovervollständigung mit Dokumentation der verfügbaren Methoden aktivieren werden, indem Sie `Assert.` eintippen und anschließend Strg-Leertaste gleichzeitig drücken.

Reguläre Ausdrücke - *Regexps*(1023) können auch genutzt werden. Diese sind über das Modul `java.util.regex.Pattern` implementiert.

Wortketten

Der größte Vorteil ist die gute Lesbarkeit, die über Wortketten erreicht wird. Diese können bei `expect()` und `should()` verwendet werden. Die folgenden Ausdrücke können aneinander gekettet werden: `.to .be .been .is .that .which .and .has .have .with .at .of .same .but .does .still .also`

```
def testObj = [
"name": "test",
"sub": [
"name": 'test sub'
],
"numbers": [1, 2, 3, 4],
"hasNumbers" : true
];
expect(testObj).to.be.an('Object').and.is.ok
expect(testObj).to.have.property('sub').that.is.an('Object').and.is.ok
expect(testObj.sub).to.have.property('name')
  .that.is.a('String').and.to.equal('test sub')
expect(testObj).to.have.property('numbers')
  .that.deep.equals([1, 2, 3, 4])
expect(testObj).to.have.property('hasNumbers', true)
```

Beispiel 50.7: Wortketten bei `expect`

```
rc.setLocal("jsonData", """
{
  "Actors": [
    {
      "name": "Tom Cruise",
      "age": 56,
      "Born At": "Syracuse, NY",
      "Birthdate": "July 3, 1962",
      "photo": "https://jsonformatter.org/img/tom-cruise.jpg",
      "wife": null,
      "weight": 67.5,
      "hasChildren": true,
      "hasGreyHair": false,
      "children": [
        "Suri",
        "Isabella Jane",
        "Connor"
      ]
    },
    {
      "name": "Robert Downey Jr.",
      "age": 53,
      "Born At": "New York City, NY",
      "Birthdate": "April 4, 1965",
      "photo": "https://jsonformatter.org/img/Robert-Downey-Jr.jpg",
      "wife": "Susan Downey",
      "weight": 77.1,
      "hasChildren": true,
      "hasGreyHair": false,
      "children": [
        "Indio Falconer",
        "Avri Roel",
        "Exton Elias"
      ]
    }
  ]
}""")
def data = rc.getJson("jsonData")
data.Actors.should.be.a("ArrayList")
expect(data.Actors[0]).to.be.a("LinkedHashMap")
Assert.instanceOf(data.Actors[0], "LinkedHashMap", "Bla")
data.Actors[0].name.should.be.a("String")
data.Actors[0].age.should.be.a("Long")
data.Actors[0].weight.should.be.a("Double")
data.Actors[0].hasChildren.should.be.a("Boolean")
rc.setGlobalJson("gData",data)
```

Beispiel 50.8: Wortketten bei `should`

Die Dokumentation hierzu finden sie in https://www.chaijs.com/api/bdd.

Unterschiede zwischen dem QF-Test Assertions API und Chai.js

Durch die Implementierung in Java ergeben sich ein paar Unterschiede zu Chai.js.

- Weil `assert` in Java und Groovy ein reserviertes Schlüsselwort ist, muss das `Assert` von QF-Test anders (nämlich groß) geschrieben werden. Dies gilt ebenfalls für die Prüfausdrücke `TRUE`, `FALSE` und `NULL`, welche vollständig in Großbuchstaben geschrieben werden müssen.
- Alle Methoden, die mit dem Präfix `strict` beginnen, verwenden bei Vergleichen den Gleichheitsoperator `==`, ansonsten wird die Java-Methode `equal()` genutzt. Eine Liste der verfügbaren `strict`-Methoden erhalten Sie, wenn Sie im Skript-Editor `Assert.strict` eintippen und anschließend Strg-Leertaste drücken.
- `Assert.test` wird anstelle von `assert` verwendet.

```
Assert.test('foo' !== 'bar', 'foo is not bar')
                    Assert.test({true}, 'Closures can return true')
```

Beispiel 50.9: Assert.test(...)

Nicht verfügbare Prüfausdrücke

Einige der Prüfungen, die unter Chai.JS zu verfügung stehen, können nicht direkt von Javascript nach Java übertragen werden, und einige Assertions sind noch nicht implementiert. Dazu gehören unter anderem:

Assert

- `isAbove(), isAtLeast(), isBelow(), isAtMost()`
- `isNaN(), isNotNan()`
- `isUndefined()`
- `isFinite()`
- `throws(), doesNotThrow()`
- `operator()`
- `closeTo()`

Expect/Should

- `.to.be.above(), .to.be.least(), .to.be.below(), .to.be.most()`
- `.to.be.NaN, .not.to.be.NaN`
- `.to.be.undefined`
- `.to.be.finite`
- `.to.throw(), .to.not.throw()`
- `.to.be.closeTo()`

50.9.3 Ergebnisbehandlung

Ergebnisbehandlung mit Assert, expect() und should() (System)

Server (automatisch weiter an SUT) Skript Name: OPT_PLAY_HANDLE_ASSERTION
Mögliche Werte: VAL_PLAY_HANDLE_ASSERTION_AS_CHECK, VAL_PLAY_HANDLE_ASSERTION_WITH_EXCEPTION, VAL_PLAY_HANDLE_ASSERTION_SILENTLY
Mit dieser Option beeinflusst man den Rückgabewert und die Protokollierung.

- "Als Check behandeln" - VAL_PLAY_HANDLE_ASSERTION_AS_CHECK
  Im Fehlerfall wird ein Fehler protokolliert, im Erfolgsfall ein erfolgreicher Check.
- "Als Exception" - VAL_PLAY_HANDLE_ASSERTION_WITH_EXCEPTION
  Es wird im Fehlerfall eine Exception geworfen. Sie kann in Skripten als Throwable und in Try-Catch-Knoten als `ScriptException` gefangen werden
- "Als Rückgabewert" - VAL_PLAY_HANDLE_ASSERTION_SILENTLY
  Die Prüfung wird ausgeführt, aber nicht als Fehler gekennzeichnet. Es wird auch keine Exception geworfen. Das Ergebnis der Prüfung wird als Wert zurückgegeben, entweder `true` oder `false`. Wenn `expect` oder `should` verwendet wird, muss `.getResult()` angehängt werden, um an den Rückgabewert zu gelangen.
- VAL_PLAY_HANDLE_ASSERTION_AUTOMATICALLY (Standardwert)
  Entspricht `VAL_PLAY_HANDLE_ASSERTION_AS_CHECK`, außer bei der Verwendung der QF-Test Assertions API in einem Unit-Test(896)-Knoten. Dann wird automatisch `VAL_PLAY_HANDLE_ASSERTION_WITH_EXCEPTION` verwendet um kompatibel mit JUnit zu sein.

Die Option wird am Anfang des Skripts gesetzt:

```
rc.setOption(Options.OPT_PLAY_HANDLE_ASSERTION,
                Options.VAL_PLAY_HANDLE_ASSERTION_SILENTLY)
                def a = 54
                def b = 55
                isEqual = Assert.test(a==b, "")
                if (isEqual) {...}
```

Beispiel 50.10: Prüfung mit Rückgabewert in einem Groovy-Skript

Wenn die Assertion in natürlicher Sprache formuliert wird, muss `.getResult()` angehängt werden, um an den Rückgabewert zu gelangen:

```
rc.setOption(Options.OPT_PLAY_HANDLE_ASSERTION,
                Options.VAL_PLAY_HANDLE_ASSERTION_SILENTLY)
                a = 54
                b = 55
                isEqual = expect(a).to.equal(b).getResult()
                if isEqual.getResult():
                    ...
```

Beispiel 50.11: Prüfung mit Rückgabewert in einem Jython-Skript

50.10 Exceptions

Alle QF-Test Exceptions(958) werden in den Skriptsprachen automatisch importiert und stehen zum Beispiel für `try/except` zur Verfügung:

```
try:
    com = rc.getComponent("someId")
except ComponentNotFoundException:
    ...
```

Beispiel 50.12: Beispiel für das Abfangen einer `ComponentNotFoundException` in Jython

In Groovy erfolgt dies über `try/catch`:

```
try {
    com = rc.getComponent("someId")
} catch (ComponentNotFoundException) {
    ...
}
```

Beispiel 50.13: Beispiel für das Abfangen einer `ComponentNotFoundException` in Groovy

Explizit sollten aus Skriptcode nur die folgenden Exceptions geworfen werden (mit `raise` bzw. `throw new`):

- `UserException("Irgendeine Meldung...")` dient als Signal für eine außergewöhnliche Fehlersituation.
- `BreakException()` oder `raise BreakException("loopId")` kann dazu verwendet werden, um aus einer Schleife[684] oder einem While[688] Knoten auszubrechen. Die Variante ohne Parameter unterbricht die innerste Schleife, mit der QF-Test ID als Parameter kann gezielt eine bestimmte Schleife abgebrochen werden.
- `ReturnException()` oder `ReturnException("value")` kann - mit oder ohne Rückgabewert - zur Rückkehr aus einer Prozedur[672] verwendet werden, analog zu einem Return[678] Knoten. Besser lesbar ist hier jedoch der Aufruf von `rc.returnValue(...)`.

50.11 Debuggen von Skripten (Jython)

Wenn Sie mit Jython-Modulen arbeiten, müssen Sie nach der Änderung eines Moduls QF-Test oder das SUT nicht neu starten sondern können mit Hilfe von `reload(<Modulname>)` das Modul erneut laden.

Das Debuggen von Skripten in einem eingebetteten Interpreter kann etwas mühsam sein. Um dies zu Vereinfachen bietet QF-Test Konsolenfenster für die Kommunikation mit allen Interpretern an. Informationen hierzu finden Sie am Ende des Abschnitt 11.1[187].

Alternativ können Sie eine Netzwerkverbindung zu einem Jython-Interpreter aufbauen, um eine interaktive Kommandozeile zu erhalten. Dies funktioniert sowohl mit QF-Test als auch mit dem SUT. Um dieses Feature zu aktivieren, müssen Sie QF-Test mit dem Kommandozeilenargument `-jythonport <Nummer>`[982] starten, mit dem Sie die Portnummer für den Zugang zum Jython-Interpreter angeben. Für das SUT definieren Sie diese mit Hilfe eines Programm-Parameter[726] in der "Extra" Tabelle des

Java-SUT-Client starten[724]- oder SUT-Client starten[728]-Knotens. Setzen Sie dieses auf `-jythonport=<Portnummer>`. Anschließend können Sie sich mittels

```
telnet localhost <Portnummer>
```

mit dem entsprechenden Jython-Interpreter verbinden. Dank Jythons Möglichkeiten zum Zugriff auf die gesamte Java-API erhalten Sie auf diesem Weg sogar eine ganz allgemeine interaktive Debugging-Schnittstelle zu Ihrer Applikation.

Kapitel 51

Web

Web

Dieses Kapitel behandelt einige Punkte, die nur für den Test von Web-Anwendungen relevant sind.

51.1 Verbesserte Komponentenerkennung mittels `CustomWebResolver`

Video

Video:

'CustomWebResolver in QF-Test'
https://www.qftest.com/de/yt/lift-off-webanwendung-spezialwebinar.html

HTML ist eine sehr flexible Sprache um den Inhalt und Aufbau einer Webseite zu beschreiben. Allerdings gibt es keine wirklichen Standards im Hinblick auf die Art und Weise wie Komponenten, die einen Button, ein Textfeld oder eine Tabelle, die Daten repräsentieren, aufgebaut werden sollen. Aus diesem Grund erstellt nahezu jedes Webframework seine Komponenten auf seine eigene Art. Dies bedeutet, dass die HTML-Struktur der Seite, der so genannte DOM Baum, je nach Framework unterschiedlich aussieht. Damit nun QF-Test die jeweiligen Objekte als Buttons oder Tabellen identifizieren kann, braucht es eine Art Übersetzungstabelle zwischen den Eigenschaften der Webseite und dem QF-Test Vokabular.

QF-Test bietet hierfür einen generischen, konfigurierbaren Ansatz, den `CustomWebResolver`, an. Dieser erlaubt es QF-Test, die Komponentenerkennung ohne zu großen Aufwand individuell an die Spezifika Ihrer Webseite anzupassen.

Die Komponentenerkennung sollte vor der Erstellung von Testfällen geprüft und gegebenenfalls optimiert werden. Empfohlene Vorgehensweise:

1. Aufnahme von GUI-Objekten, mit denen im Test interagiert werden soll, auf unter-

schiedlichen Webseiten der Applikation.

2. Prüfung der aufgenommenen QF-Test Komponenten,

   - ob diesen Generische Klassen(1329) zugewiesen wurden,
   - ob sie ausreichend Wiedererkennungsmerkmale aufweisen (Name(64) und Merkmal(70) Attribute, `qfs:label*`-Varianten(74) in Weitere Merkmale)
   - wie tief die Komponentenhierarchie ist
   - ob komplexe Komponenten, also Tabellen, Listen, Bäume etc, als solche erkannt wurden und Unterelemente über Index referenziert werden.

   Detaillierte Informationen zur Standardaufnahme von Web-Objekten und Entscheidungskriterien, ob die Standardaufnahme ausreichend ist, finden Sie in Erkennung von Web-Komponenten und Toolkits(229).

3. Falls Schwachstellen identifiziert werden, Untersuchen der jeweiligen GUI-Objekte auf unterschiedlichen Webseiten und Identifizierung der charakteristischen Attribute einer bestimmten Objektklasse und eindeutiger HTML-Attribute für Name und Merkmal-Attribute in QF-Test. Die Webseite kann mit UI-Inspektor(108) untersucht werden.

4. Konfiguration der Komponentenzuordnung, siehe Der CustomWebResolver installieren Knoten(1082).

5. Wenn bereits Tests bestehen: Aktualisierung bereits vorhandener Komponente(930) Knoten, siehe Komponenten aktualisieren(105).

51.1.1 Generelle Konfigurationsmöglichkeiten

Der CustomWebResolver installieren Knoten(1082) bietet folgende Konfigurationsmöglichkeiten:

Zuordnung von HTML-Objekten zu funktionalen Komponenten

Funktionalen GUI-Objekten wie Buttons, Textfeldern, Datentabellen etc. können QF-Test Komponenten einer bestimmten generischen Klasse zugeordnet werden. Vorteile:
- die Aufnahme zusätzlicher Wiedererkennungsmerkmale,
- klassenspezifische Checks,
- die Indexierung von Unterelementen bei der Aufnahme,
- die Aufnahme eines generischen Klassentyps,
- die Schärfung der Wiedererkennung allein durch die generische Klasse gegenüber HTML-Klassen.

Die aufgenommenen Informationen hängen von der jeweiligen generischen Klasse ab. Was im Detail aufgenommen wird, ist in Generische Klassen(1329) beschrieben.

Verringerung der aufgenommenen Komponentenhierarchie

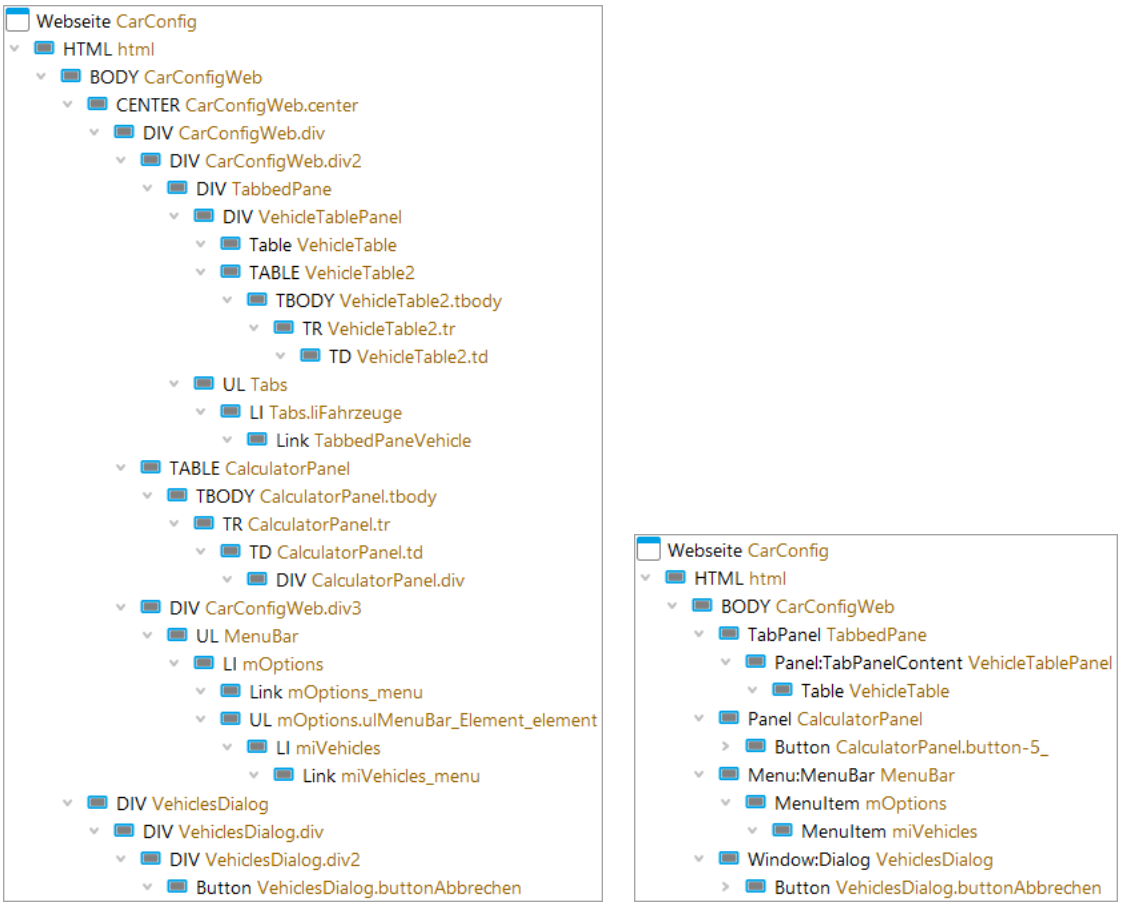
Es kann angegeben werden, welche Elemente der Webseite für den GUI-Test relevant sind und welche nicht, z.B. können HTML-Elemente, die zur Formatierung der Anzeige dienen, gezielt ignoriert werden. So kann die komplexe interne HTML-Struktur der Webseiten auf die für den Test relevanten Komponenten und Datenobjekte in QF-Test abgebildet werden. Das Video

Video

'Die Explosion der Komplexität in der Web Testautomatisierung eindämmen'

https://www.qftest.com/de/yt/web-testautomatisierung-40.html

zeigt eindrucksvoll den Umgang von QF-Test mit tief geschachtelten DOM-Strukturen.



Klassische Aufzeichnung Verbesserte Aufzeichnung

Abbildung 51.1: Verbesserte Komponentenauflösung am Beispiel des ”CarConfigurator Web”

Verwendung alternativer Id-Attribute

Standardmäßig werden die HTML-Attribute `id` oder `name` für die Identifizierung verwendet und im Attribut Name der Komponente abgespeichert. Wenn andere Attribute verwendet werden sollen, kann dies konfiguriert werden.

Identifizierung weiterer Erkennungsmerkmale

Es kann ein HTML-Attribut angegeben werden, dessen Wert für die Wiedererkennung im Attribut Merkmal der Komponente abgespeichert werden sollen.

Ein HTML-Objekt kann über die folgenden HTML Merkmale identifiziert werden:

- auf Basis des `class` Attributs,
- auf Basis eines beliebigen Attributs,

- auf Basis des HTML-Tags.

Die einzelnen Zuordnungen können ihrerseits Bedingungen unterliegen. QF-Test bietet die folgenden Möglichkeiten, die einzelnen Zuordnungen weiter zu parametrisieren. Eine Kombination der Optionen ist möglich.

- Verwendung regulärer Ausdrücke.
- Zuordnung nur, wenn das Objekt in einem anderen Objekt einer bestimmten Klasse liegt, entweder in einer beliebigen oder einer fest definierten Ebene in der Objekthierarchie.
- Zuordnung nur, wenn das Objekt zusätzlich zu den angegebenen Kriterien ein bestimmtes HTML-Tag besitzt.

Eine funktionale Komponente ist in HTML häufig durch mehrere, ineinander verschachtelte Elemente implementiert. Für die Aufnahme und Wiedergabe der Komponente in QF-Test ist es dabei völlig egal, welches der Elemente der funktionalen QF-Test Komponentenklasse zugeordnet wird. Entscheidend ist, dass die ausgewählte Komponente gute Wiedererkennungsmerkmale besitzt. Dabei werden von QF-Test auch die darin liegenden Objekte nach Wiedererkennungsmerkmalen und Texten durchsucht und für das zugeordnete Objekt abgespeichert. Beispiel: CustomWebResolver – TabPanel und Accordion(1107)

Außerdem gibt es funktionale Komponenten, die in HTML durch mehrere Objekte implementiert werden müssen. Diese werden in QF-Test ”komplexe Komponenten” genannt. Es handelt sich hierbei um Comboboxen, Listen, Bäume, Tabellen etc. So ist es zum Beispiel bei einer Liste notwendig, QF-Test mitzuteilen, welche Art von HTML-Objekten den Container der Liste darstellen und welche die einzelnen Listeinträge.

Eine Aufstellung, der HTML-Objekte, die den Teilen einer komplexen Komponente zugeordnet werden müssen, damit die komplexe Komponente selbst richtig erkannt werden kann, inklusive passender Beispiele, finden Sie in den folgenden Abschnitten:

- CustomWebResolver – Combobox(1105)
- CustomWebResolver – Liste(1103)
- CustomWebResolver – Tabelle(1096)
- CustomWebResolver – TabPanel und Accordion(1107)
- CustomWebResolver – Baum (Tree)(1099)

In den meisten Fällen wird das Attribut `class` der HTML-Komponenten signifikant für die Komponentenerkennung sein. In so einem Attribut finden Sie Informationen zur

fachlichen Funktion der Komponente, z.B. ob es sich um einen Button oder eine Datentabelle handelt. Andere Webframeworks arbeiten mit dem `type` Attribut oder auch anderen Attributen der HTML-Objekte. Außerdem gibt es auch eine Reihe von Frameworks, bei denen Sie nur mit speziellen JavaScript Methoden an diese Informationen herankommen. Für solche Frameworks sind zusätzlich zur Konfiguration mittels `CustomWebResolver` weitere Resolver erforderlich. Wir konzentrieren uns hier für's Erste auf den Normalfall. Für das Vorgehen bei komplizierteren Fällen kontaktieren Sie bitte unser Supportteam.

51.1.2 Der CustomWebResolver installieren Knoten

Die Zuordnung der funktionalen HTML-Elemente zu Generische Klassen(1329) erfolgt im Normalfall über den CustomWebResolver installieren(902) Knoten.

7.0+ Vor QF-Test Version 7 erfolgte diese Zuordnung über die Prozedur `installCustomWebResolver`(948) und `updateCustomWebResolverProperties` aus der Standardbibliothek `qfs.qft`. Diese Prozeduraufrufe sollten nun in CustomWebResolver installieren Knoten konvertiert werden. Vor der Konvertierung des Prozeduraufrufs wird eine automatische Prüfung der enthaltenen Parameter durchgeführt, um mögliche fehlerhafte Angaben aufzudecken und den Wechsel auf den CustomWebResolver installieren Knoten zu vereinfachen. Falls Ihr Prozeduraufruf Variablen enthält, müssen Sie bei der Konvertierung ein Testlaufprotokoll angeben, in dem die gewünschten Variablenwerte ersichtlich sind. Falls Ihr Prozeduraufruf ungültige Einträge als Kommentar verwendet, müssen Sie diese möglicherweise vor der Konvertierung entfernen oder nach der Konvertierung an die gewünschte Stelle verschieben.

Wenn Sie zur Erstellung der Verbindungssequenz den Schnellstart-Assistenten, siehe Kapitel 3(32), verwendet haben, was wir sehr empfehlen, finden Sie den CustomWebResolver installieren Knoten im letzten Knoten der Startsequenz. Diesen Knoten sollten Sie bei Bedarf für Ihre Anwendung konfigurieren.

Abbildung 51.2: Aufruf des CustomWebResolvers im Vorbereitung Knoten des Schnellstart-Assistenten

Allgemeine Informationen zur Web-Komponentenerkennung finden Sie in Erkennung von Web-Komponenten und Toolkits[229] sowie in Generelle Konfigurationsmöglichkeiten[1078].

Nachfolgend wird zunächst die Syntax des CustomWebResolver installieren Knotens und anschließend die Konfigurationskategorien erläutert.

Hinweis

Bitte beachten Sie, dass Änderungen im CustomWebResolver installieren Knoten die Erkennungsmerkmale für ein GUI-Element verändern können. Infolgedessen können sie von den Wiedererkennungsmerkmalen bereits aufgenommener Komponente[930] Knoten abweichen. Änderungen sollten daher in bereits vorhandenen Komponente Knoten, wie in Komponenten aktualisieren[105] beschrieben, nachgezogen werden. Im Idealfall sollte die Konfiguration des CustomWebResolver installieren Knotens vor der Erstellung der Tests erfolgen und Knoten, die während der Konfigurationsphase erstellt wurden, wieder gelöscht werden.

CustomWebResolver installieren Knoten – Syntax

Die Konfiguration des CustomWebResolver installieren Knotens findet in dessen Editorfeld unter Verwendung der YAML-Syntax statt. Dabei sind nur Grundkenntnisse über die Funktionsweise von YAML notwendig, welche nachfolgend erläutert werden.

Auf oberster Ebene existieren die Kategorien (siehe folgende Abschnitte). Diese stehen in einer eigenen Zeile und werden mit einem Doppelpunkt abgeschlossen (Dictionary-Schlüssel). Auf der zweiten Ebene gibt es Einträge, die jeweils in einer neuen Zeile mit einem Bindestrich beginnen (Listen-Elemente). Weitere Ebenen sind durch Einrückung gekennzeichnet.

Um die Arbeit mit der YAML-Konfiguration zu erleichtern, können über die Werkzeugleiste über dem Editor verschiedene Vorlagen eingefügt werden.

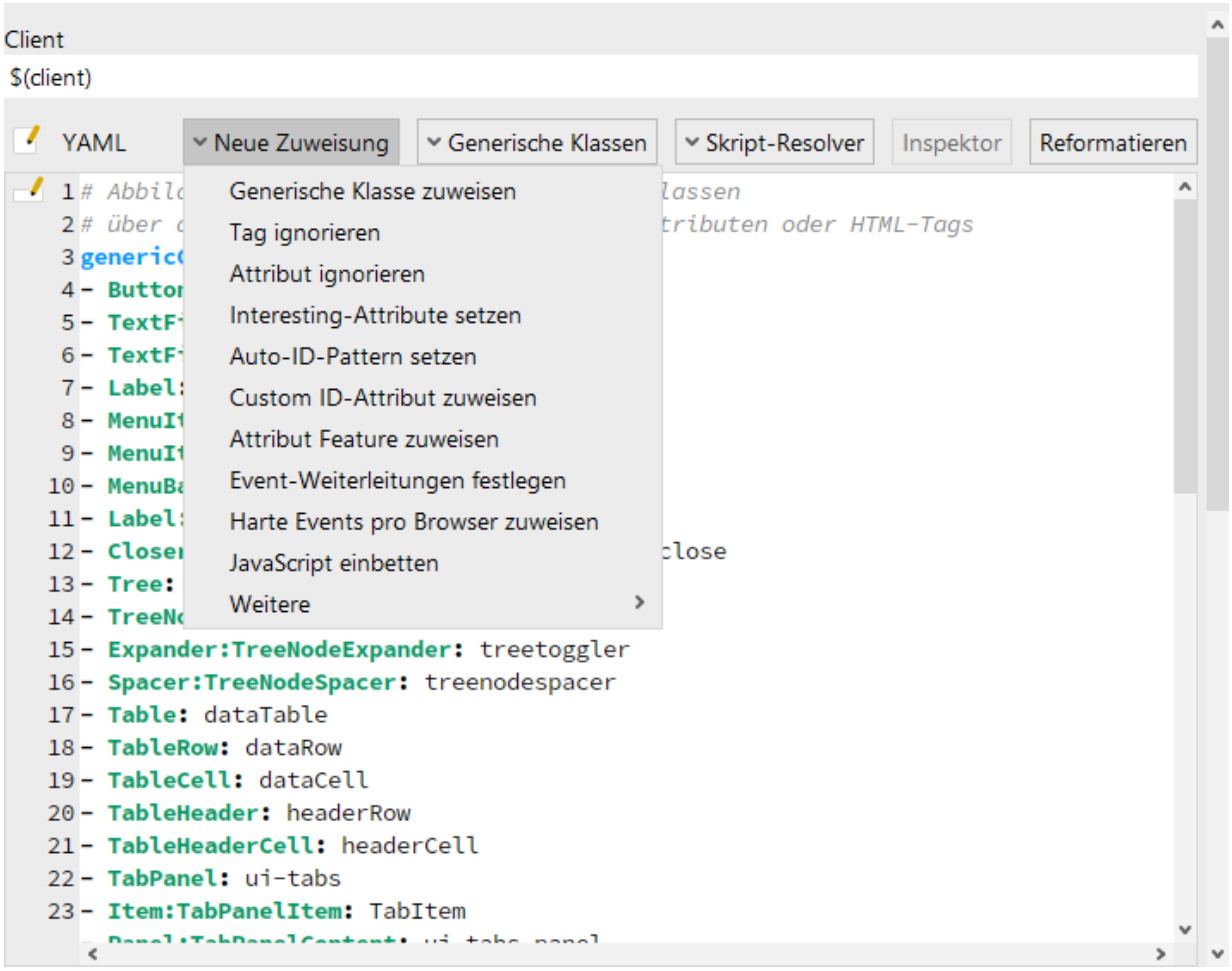


Abbildung 51.3: CustomWebResolver Konfigurationsvorlagen

Das Auswahlmenü, das über den Editier-Button neben der Zeilennummerierung geöffnet wird, ist kontextsensitiv. Es bietet für die jeweilige Zeile alle passenden Aktionen an. Wenn Sie über das Auswahlmenü arbeiten, haben Sie immer die volle Übersicht über die möglichen Aktionen und bekommen automatisch die korrekte Syntax vorgelegt.

Wenn Sie Schnellstart Ihrer Anwendung[32] zur Erstellung der Verbindungssequenz verwendet und dort die Standardeinstellung bei der Framework-Auswahl belassen haben, erhalten Sie eine Konfiguration mit zwei Kategorien und zwei Einträgen auf der zweiten Ebene plus einige erläuternde Kommentare:

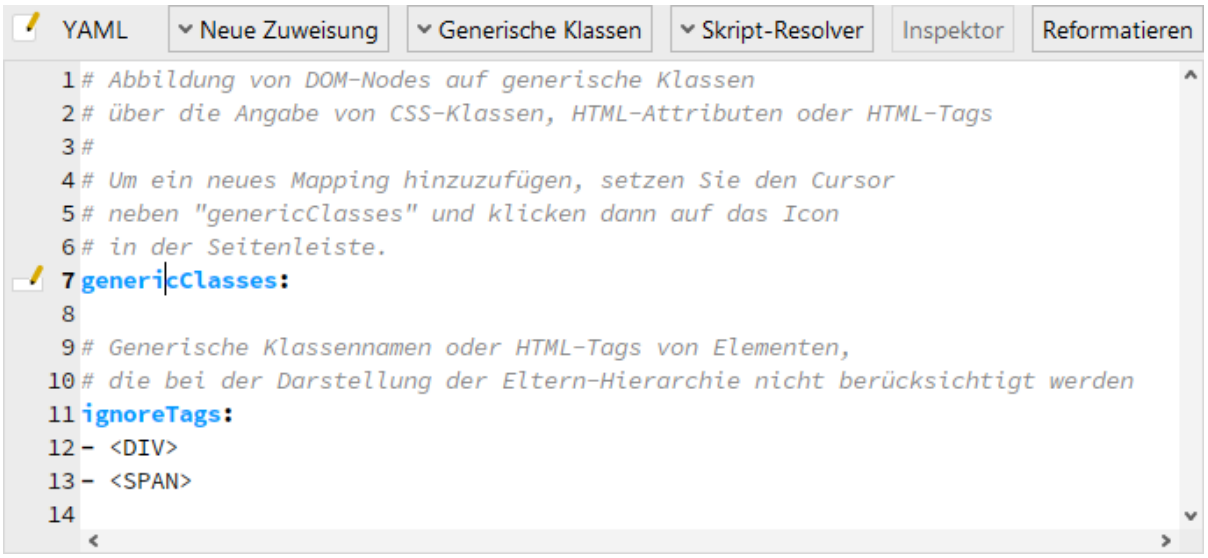


Abbildung 51.4: CustomWebResolver mit Vorlage für `genericClasses`

In dieser Konfiguration wurde die Zeile nach der Kategorie `genericClasses` angeklickt und anschließend über den Editier-Button links neben der Zeilennummerierung eine Vorlage für eine generische Klasse eingefügt. (Die Kommentare wurden entfernt.)

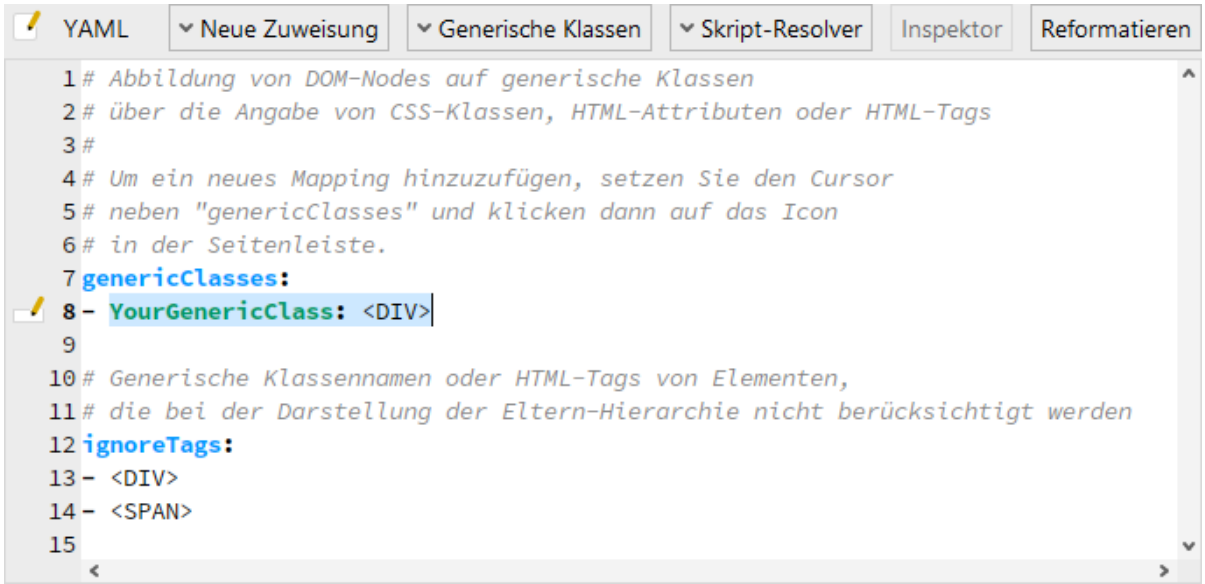


Abbildung 51.5: CustomWebResolver mit zwei generischen Klassen

Anschließend wurde die generische Klasse `List` eingetragen sowie die CSS-Klasse

datalist. HTML-Elementen mit dieser CSS-Klasse wird bei der Komponentenerkennung nun diese generische Klasse zugewiesen. Der Vorgang wurde für die generische Klasse Item:ListItem wiederholt. Diese wird jedem GUI-Element mit dem Tag LI zugewiesen. Im Normalfall gibt es jedoch auch HTML-Elemente mit diesem Tag, die eine andere Funktion haben. Hier sollen nur solche LI-Elemente berücksichtigt werden, die sich in einer List-Komponente befinden. Daher wird im nächsten Schritt über den Editier-Button der Eintrag "Ancestor hinzufügen" ausgewählt. Sie sehen, dass sich die Syntax für den Eintrag ändert: sobald mehr als ein Merkmal für die Zuweisung benötigt wird, wird die erste Zuweisung mit einem passenden Präfix in die nächste Ebene verschoben und das weitere Merkmal auf dieser Ebene hinzugefügt.

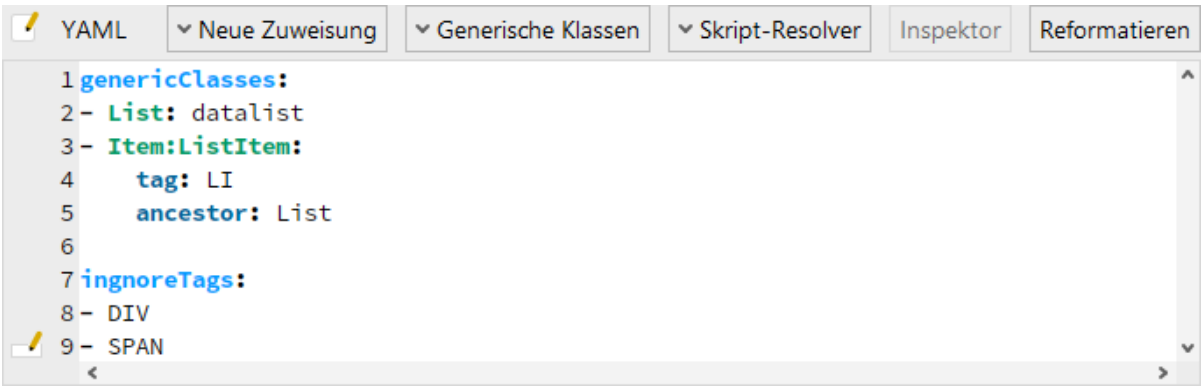


Abbildung 51.6: CustomWebResolver mit komplexerer Zuweisung

CustomWebResolver Konfigurationskategorien

Jeder CustomWebResolver basiert unter anderem auf einer global definierten Standardkonfiguration mit allgemeingültigen Regeln, die Sie in der mitgelieferten Bibliothek qftest-9.0.0/include/qfs-resolvers.qft in der Prozedur qfs.web.cwr.helpers.default einsehen können.

Eine vollständige Liste aller verfügbaren Konfigurationskategorien finden Sie in QF-Test im Knoten CustomWebResolver installieren über die Schaltfläche "Neue Zuweisung".

In den folgenden Abschnitten werden die wichtigsten Konfigurationskategorien kurz erklärt. Bitte beachten Sie, dass Sie sämtliche Funktionen der verschiedenen Kategorien über den kontextsensitiven Editier-Button erreichen können. Nicht jede mögliche Variation der Syntax wird hier beschrieben.

Konfigurationskategorie `base`

Enthält den Kurznamen des Basis-Resolvers, der als Grundlage der Konfiguration verwendet werden soll:

- `autodetect`: automatische Erkennung des verwendeten Frameworks. Fällt auf `custom` zurück, falls kein unterstütztes Framework erkannt wurde.
- `custom`: die Anwendung wurde mit keinem der von QF-Test unterstützten Frameworks erstellt. In diesem Fall kann `base` auch komplett weggelassen werden.
- Der Kurzname des Frameworks, zum Beispiel `vaadin`:
  Es werden die mit QF-Test in `qftest-9.0.0/include/qfs-resolvers.qft` ausgelieferten Zuweisungen für das entsprechende Framework verwendet. In Ihrer Konfiguration können Sie diese mit eigenen Zuweisungen ergänzen.

  Den Kurznamen des jeweiligen Frameworks können Sie der Tabelle Tabelle 51.7(1123) entnehmen. Wenn Sie die Startsequenz über den Schnellstartassistenen erstellen lassen und dort ein Framework angeben, wird der Kurzname direkt hier eingetragen.

  Als Kurzname kann hier auch der volle Name einer eigenen Prozedur angegeben werden, welcher einen eigenen Basis-Resolver für die getestete Anwendung enthält. Man kann auch mehrere eigene Resolver-Prozeduren erstellen und diese in einem CustomWebResolver installieren Knoten frei orchestrieren. Dabei ist auch die Kombination mit einem in QF-Test vordefinierten Basis-Resolver möglich. Dessen Kurzname muss dann an erster Stelle der Liste stehen. Die Resolver-Konfigurationen werden dann in der angegebenen Reihenfolge angewandt.

```
base:
- vaadin
- myResolvers.Panels
- myResolvers.otherClasses
```

Beispiel 51.1: Liste von Basis-Resolvern

Konfigurationskategorie `genericClasses`

In dieser Kategorie werden die Erkennungsmerkmale festgelegt, aufgrund derer einem GUI-Element eine bestimmte generische Klasse zugewiesen werden soll. Die jeweiligen Eigenschaften der generischen Klassen sind in Kapitel 61(1329) erläutert.

Generische Klassen können eine Typerweiterung erhalten. Diese ist beim Mappen bestimmter HTML-Elemente wie z.B. `Item` relevant: `Item:ListItem` bezeichnet Listenelemente, `Button:ComboBoxButton` einen Button innerhalb einer Combobox. Typer-

weiterungen sind auch deshalb interessant, weil sie beim Mapping frei vergeben werden können. Das Beispiel in CustomWebResolver – Tabelle(1096) verwendet diese Technik.

Hinweis

Bei der Verwendung der Typerweiterung in der Klassenangabe von SmartIDs ist zu beachten, dass der Doppelpunkt vor einer eigenen Typerweiterung durch einen Rückstrich geschützt werden muss. Siehe auch SmartID-Syntax für Klasse(84).

Die angegebenen Einträge werden von oben nach unten evaluiert, für jedes HTML-Element wird die oberste zutreffende generische Klasse verwendet. Einzige Ausnahme hiervon bilden Einträge mit `ancestor`, diese werden immer als erstes evaluiert.

Für die Komponentenerkennung stehen grundsätzlich der Tag-Name und die Attribute des GUI-Elements zur Verfügung. Das Attribut `class` hat eine Sonderrolle. Es enthält die CSS-Klassen, die in der HTML-Programmierung die Darstellung des GUI-Elements beeinflussen und somit häufig charakteristisch für eine bestimmte HTML-Elementenklasse sind.

CustomWebResolver installieren bietet die Möglichkeit, jeden dieser drei Fälle als Zuweisung zu verwenden:

CSS-Klasse

Die CSS-Klasse bezieht sich auf einen Eintrag im Attribut `class` des GUI-Elements. Bitte beachten Sie, dass zwar mehrere CSS-Klassen durch Leerzeichen getrennt im Attribut stehen können, jedoch immer nur eine einzelne Klasse angesprochen wird.

Einfache Zuweisung: die CSS-Klasse wird in der gleichen Zeile hinter die generische Klasse geschrieben.

```
genericClasses:
- Button: btn
```

Beispiel 51.2: Einfache Zuweisung CSS-Klasse zu generischer Klasse

Im Beispiel erhalten nur HTML-Elemente mit der CSS-Klasse `btn` die generische Klasse `Button`.

Zuweisung bei mehreren Kriterien: die CSS-Klasse wird eingerückt in einer Zeile unter der generischen Klasse mit dem Präfix `css:` eingetragen.

```
genericClasses:
- Button:
    css: btn
    tag: DIV
```

Beispiel 51.3: Zuweisung CSS-Klasse und Tag-Name zu generischer Klasse

Im Beispiel erhalten nur HTML-Elemente mit der CSS-Klasse `btn` und dem Tag `DIV` die generische Klasse `Button`.

Es ist ebenfalls möglich, mehrere CSS-Klassen gleichzeitig anzugeben. Es muss dann nur eine der angegebenen CSS-Klassen auf ein Element zutreffen.

```
genericClasses:
- Button:
    css:
    - btn
    - button
```

Beispiel 51.4: Zuweisung mehrerer alternativer CSS-Klassen

HTML-Tag-Name

Einfache Zuweisung: der Tag-Name wird in spitzen Klammern in der gleichen Zeile hinter die generische Klasse geschrieben.

```
genericClasses:
- TableCell: <TD>
```

Beispiel 51.5: Einfache Zuweisung Tag-Name zu generischer Klasse

Im Beispiel erhalten nur HTML-Elemente mit dem Tag `TD` die generische Klasse `TableCell`.

Zuweisung mit mehreren Kriterien: der Tag-Name wird eingerückt in einer Zeile unter der generischen Klasse mit dem Präfix `tag:` eingetragen.

```
genericClasses:
- TableCell:
    tag: TD
    ancestor: TableRow
```

Beispiel 51.6: Zuweisung Tag-Name mit `ancestor` zu generischer Klasse

Im Beispiel erhalten nur HTML-Elemente mit dem Tag `TD` die generische Klasse `Button`, wenn sie in einem GUI-Element der Klasse `TableRow` liegen.

Es ist ebenfalls möglich, mehrere HTML-Tagnamen gleichzeitig anzugeben. Es muss dann nur einer der angegebenen Namen auf ein Element zutreffen.

```
genericClasses:
- Button:
    tag:
    - CUSTOM-BUTTON
    - BUTTON
```

Beispiel 51.7: Zuweisung mehrerer alternativer HTML-Tagnamen

HTML-Attribut

Einfache Zuweisung: der Attributname, ein Gleichheitszeichen und der Attributwert werden in der gleichen Zeile hinter die generische Klasse geschrieben.

```
genericClasses:
- TableRow: role=datarow
```

Beispiel 51.8: Einfache Zuweisung Attributwert zu generischer Klasse

Im Beispiel erhalten nur HTML-Elemente mit dem Attribut `role` und dem Wert `datarow` die generische Klasse `TableRow`.

Zuweisung mit mehreren Kriterien: Eingerückt unter der generischen Klasse werden eine Zeile für den Attributname mit dem Präfix `attribute:` und eine Zeile für den Attributwert mit dem Präfix `attributeValue:` eingetragen.

```
genericClasses:
- TableRow:
    attribute: role
    attributeValue: datarow
```

Beispiel 51.9: Zuweisung Attributwert zu generischer Klasse

Im Beispiel erhalten nur HTML-Elemente mit dem Attribut `role` und dem Wert `datarow` die generische Klasse `TableRow`.

Hinweis

Auch das Attribut `class` kann hier genutzt werden. Dann muss jedoch der gesamte Wert des Attributs passen, damit die Zuweisung erfolgt. Wenn zum Beispiel zwei CSS-Klassen vorhanden sein müssen und die anderen ignoriert werden sollen, bietet sich ein regulärer Ausdruck an. Dies ist auch ein Beispiel für eine weitere Definitionsebene.

```
genericClasses:
- TableRow:
    attribute: class
    attributeValue:
        value: (^|.*\s)btn(\s.*|$)
        regex: true
```

Beispiel 51.10: Zuweisung Attributwert zu generischer Klasse

Ancestor/Parent/Sibling

Um eine Zuweisung zusätzlich abhängig von dem Vorhandensein eines bestimmten übergeordneten oder nebengeordneten Elements zu machen werden `ancestor`, `parent`, `interestingparent` oder `sibling` genutzt.

Einfache Zuweisung: die Klasse des Containers wird in der gleichen Zeile hinter das jeweilige Präfix geschrieben.

```
genericClasses:
- TableRow:
    tag: TR
    ancestor: Table
```

Beispiel 51.11: Einfache Ancestor-Zuweisung

Im Beispiel erhalten nur HTML-Elemente mit dem HTML-Tag `TR` die generische Klasse `TableRow`, die irgendwo innerhalb eines Elements mit der Generischen Klasse ”Table” liegen.

Komplexe Zuweisung: Eingerückt unter einem der Typen-Präfixe folgen `level:` und `className:`.

Die definitionen der verschiedenen Typen sind:

- `ancestor`: beliebige Verschachtelung,
- `parent`: direkt innerhalb des übergeordneten Elements,
- `interestingparent`: direkt innerhalb des QF-Test Elements von `node.getInterestingParent()` und
- `sibling`: innerhalb des gleichen Elternelements.

Mit `ancestor` oder `sibling` kann die genaue Distanz zwischen Ursprungs- und Zielelement über `level:` festgelegt werden.

```
genericClasses:
- TableRow:
    tag: TR
    ancestor:
        level: 2
        className: Table
```

Beispiel 51.12: Komplexe Ancestor-Zuweisung

Im Beispiel erhalten nur HTML-Elemente mit dem HTML-Tag `TR` die generische Klasse `TableRow`, die zwei Ebenen tief innerhalb eines Elements mit der Generischen Klasse ”Table” liegen.

Wenn HTML-Elemente mit unterschiedlichen Erkennungsmerkmalen die gleiche generische Klasse erhalten sollen, werden für diese Klasse zwei Einträge vorgenommen:

```
genericClasses:
- TableRow:
    attribute: role
    attributeValue: datarow
- TableRow:
    tag: TR
    ancestor: Table
```

Beispiel 51.13: Gleiche generische Klasse für unterschiedliche HTML-Elemente

Hinweis `ancestor` etc. ist auch in einigen anderen Konfigurationskategorien verfügbar. Prüfen Sie dafür das Editier-Menü auf den Eintrag ”Ancestor hinzufügen” oder ”Sibling hinzufügen”.

Konfigurationskategorie `ignoreTags`

Eine Liste von Klassennamen oder Tags, für deren Komponenten keine Knoten in der Komponentenhierarchie erstellt werden, solange diese nicht durch andere Anweisungen gemappt werden. Um Tags von Klassennamen zu unterscheiden, müssen Tags in Großbuchstaben oder zwischen spitzen Klammern angegeben werden.

Zum Beispiel werden durch den folgenden Eintrag alle `DIV` und `TBODY` Elemente, die nicht anderweitig gemappt wurden und mit denen nicht direkt interagiert wird, bei der Erstellung des Komponentenbaums ignoriert:

```
ignoreTags:
- <DIV>
- <TBODY>
```

Beispiel 51.14: ignoreTags

Konfigurationskategorie `ignoreByAttributes`

Eine Liste von HTML-Attributnamen und -Werten, für deren Komponenten keine Knoten in der Komponentenhierarchie erstellt werden:

```
ignoreByAttributes:
- id: container
```

Beispiel 51.15: ignoreByAttributes

Konfigurationskategorie `autoIdPatterns`

Eine Liste von Mustern, aus denen abgeleitet werden kann, ob Ids automatisch über das Framework generiert wurden. Falls das `id` Attribut dem Muster entspricht, wird der Wert nicht für das Attribut Name der Komponente verwendet:

```
autoIdPatterns:
- myAutoId
- value: auto.*
  regex: true
```

Beispiel 51.16: autoIdPatterns

Konfigurationskategorie `customIdAttributes`

Eine Liste von Attributnamen, deren Werte als Ids für die Komponente verwendet werden können. Beachten Sie, dass Sie das Attribut ”id” hier mit aufnehmen müssen, wenn Sie das Standardverhalten von QF-Test nur ergänzen möchten.

Das folgende Beispiel bewirkt, dass ausschließlich das Attribut `myid` als Komponenten-ID interpretiert wird.

```
customIdAttributes:
- myid
```

Beispiel 51.17: customIdAttributes

Konfigurationskategorie `interestingByAttributes`

Eine Liste von Attribut-Wert-Paaren, die angeben, ob eine Komponente für die Wiedererkennung interessant ist und somit ein Knoten dafür in der Komponentenhierarchie erstellt werden soll.

```
interestingByAttributes:
- id: container
- id: header
```

Beispiel 51.18: interestingByAttributes

Konfigurationskategorie `attributesToQftFeature`

Eine Liste von Attributnamen, deren Werte für das Merkmal Attribut der Komponente verwendet werden sollen.

Konfigurationskategorie `redirectClasses`

In dieser Kategorie können Sie für einzelne generische Klassen konfigurieren, ob Events an Elemente dieser Klasse weitergeleitet werden sollen oder bevorzugt ein Elternelement aufgezeichnet werden soll. Sie können auch mehrere Regeln definieren, um abhängig von der Klasse des Elternelements unterschiedliches Verhalten zu erreichen.

Einträge werden von oben nach unten evaluiert und nur der erste passende Eintrag angewendet.

Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Supportteam.

Konfigurationskategorie `documentJS`

Javascript-Code, der in die Webseite eingebettet werden soll. Kann verwendet werden, um benutzerspezifische JavaScript-Funktionen einzufügen oder bei jedem Seitenaufruf bestimmten JavaScript-Code auszuführen.

Beachten Sie im folgenden Beispiel die Syntax für mehrzeilige Zeichenketten in YAML. Injizierter JavaScript-Code sollte möglichst keine Leerzeilen enthalten, um keinen Konflikt mit der YAML-Syntax auszulösen.

```
documentJS: |-
  window.hello = function() {
      console.log("Hello World");
  }
  hello();
```

Beispiel 51.19: documentJS

Konfigurationskategorie `attributesToQftName`

Eine Liste von Attributnamen, deren Werte für das Name Attribut von Komponenten verwendet werden sollen.

Konfigurationskategorie `nonTrivialClasses`

Eine Liste von CSS-Klassen, für die die zugehörigen Elemente von QF-Test nicht ignoriert werden sollen. Triviale Elemente sind normalerweise `I`, `FONT`, `BOLD` etc., es sei denn sie haben eine der angegebenen CSS-Klassen.

Mit Vorsicht verwenden! Falls Sie sich unsicher sind, wenden Sie sich bitte an das QF-Test Supportteam.

Konfigurationskategorie `browserHardClickClasses`

Eine Liste von Klassen, auf deren Komponenten immer harte oder semi-harte Events abgespielt werden sollen, z.B. spielt der Eintrag `Button` harte Klicks auf Buttons ab. Kann auf bestimmte Browser eingeschränkt werden.

Konfigurationskategorie `treeResolver`

Diese Kategorie bündelt Konfigurationsmöglichkeiten, die den Umgang von QF-Test mit Baumknoten in `Tree` und `TreeTable` steuern. Verwenden Sie diese Kategorie falls QF-Test in Ihrer Anwendung Schwierigkeiten hat, die Hierarchieebenenen von Bäumen zu unterscheiden, einzelne Baumknoten ein- und auszuklappen, oder den Textinhalt von Baumknoten korrekt auszulesen.

In seltenen Fällen, wenn die Parameter der Kategorie nicht ausreichend sind, können Sie auch Das `TreeIndentationResolver` Interface(1186) nutzen.

Konfigurationskategorie `treetableResolver`

Diese Kategorie bündelt Konfigurationsmöglichkeiten, die den Umgang von QF-Test mit Baumknoten in `TreeTable`-Komponenten steuern. Hier können Sie beispielsweise angeben, in welchem Spaltenindex der Tabellen Ihrer Anwendung sich ein Baum befindet, falls QF-Test diesen nicht automatisch bestimmen kann.

51.1.3 CustomWebResolver – Tabelle

Für das Zuweisen von Tabellen ist es notwendig sowohl die Hauptkomponente selbst, also diejenige, die alle Einträge enthält, wie auch die Tabellenzeilen, welche einzelne Zellen enthalten und die Tabellenzellen zu mappen. Darüber hinaus muss sowohl die Zeile, welche die Überschriften enthält, als auch die einzelnen Überschriften generischen Klassen zugewiesen werden.

Klasse	Notwendige Komponenten / Unterelemente
Table	Stellt die Tabelle dar, beinhaltet die Einträge.
TableRow	Stellt eine Tabellenzeile dar.
TableCell	Stellt eine Tabellenzelle dar.
TableHeader	Stellt die Zeile mit den Überschriften dar.
TableHeaderCell	Stellt eine Überschrift dar.
	Optionale Unterelemente
CheckBox:TableCellCheckBox	(Optional) Stellt eine CheckBox unter einer Tabellenzelle dar.
Icon:TableCellIcon	(Optional) Stellt ein Icon unter einer Tabellenzelle dar.
CheckBox:TableCellHeaderCheckBox	(Optional) Stellt eine CheckBox unter einer Tabellenüberschrift dar.
Icon:TableCellHeaderIcon	(Optional) Stellt ein Icon unter einer Tabellenüberschrift dar.

Tabelle 51.1: Mapping von Tabellen

Zusätzlich zu dem folgenden Beispiel finden Sie ein ausführliche Beschreibung zum Mappen einer Tabelle in Klassenzuweisung für komplexe Komponenten wie Tabellen(1116).

Beispiel:

Die folgenden HTML-Befehle definieren zwei Tabellen, eine Datentabelle und eine weitere für die räumliche Anordnung von Buttons:

```
<div role="datatablecontainer">
    <table>
        <th type="header">
            <td class="datacell">Form</td>
            <td class="datacell">Farbe</td>
        </th>
        <tr>
            <td class="datacell">Quadrat</td>
            <td>Rot</td>
        </tr>
        <tr>
            <td class="datacell">Raute</td>
            <td class="datacell">Blau</td>
        </tr>
    </table>
</div>
<table>
     <tr>
        <td>
            <div class="button">Speichern</div>
            <div class="button">Abbrechen</div>
        </td>
    </tr>
</table>
```

Beispiel 51.20: HTML Table

Nachfolgende CustomWebResolver-Konfiguration mappt nur die Datentabelle auf eine Tabellekomponente von QF-Test, nicht aber die Tabelle, die für die Anordnung der Buttons verwendet wird:

```
genericClasses:
- Button: button
- TableCell:
    css: datacell
    ancestor: TableRow
- Panel:myTablePanel: role=datatablecontainer
- TableHeader:
    attribute: type
    attributeValue: header
    tag: th
- Table:
    tag: table
    ancestor:
      type: parent
      className: Panel:myTablePanel
- TableHeaderCell:
    tag: td
    ancestor: TableHeader
- TableRow:
    tag: tr
    ancestor: Table
ignoreTags:
- <DIV>
- <SPAN>
- <TABLE>
```

Beispiel 51.21: HTML Table

In der Zuweisung von `Panel:myTablePanel` wurde der Klassentyp `myTablePanel` frei ”erfunden”. Damit ist das `DIV`-Element eindeutig definiert und kann in der Spezifizierung der Tabelle mit `ancestor: Panel:myTablePanel` verwendet werden.

Beachten Sie Zuweisung für die Spaltenüberschriftenzeile `TableHeader`. Dies ist wie folgt zu lesen: Das Attribut `type` mit dem Wert `header` wird der generischen Klasse `TableHeader` zugewiesen, aber nur, wenn der HTML-Tag-Name gleich `TH` ist.

Um sicher zu gehen, dass die Zuweisungen der weit verbreiteten Tags `TR` und `TD` an anderen Stellen nicht zu Problemen führen, wird `ancestor:` hinzugefügt.

HTML-Elemente mit dem Tag `TABLE`, die keiner QF-Test Tabelle zugewiesen wurden, sollen nicht in der QF-Test Komponentenhierarchie erscheinen. Daher der Eintrag `TABLE` in der Kategorie `ignoreTags`, zusätzlich zu den Standardeinträgen `<DIV>` und `<SPAN>`, die bewirken, dass nicht zugewiesene `DIV`- und `SPAN`- Elemente die Komponentenhierarchie nicht unnötig aufblähen.

51.1.4 CustomWebResolver – Baum (Tree)

Für das Mappen von Bäumen ist es notwendig sowohl die Hauptkomponente selbst, also diejenige, die alle Einträge enthält, wie auch die einzelnen Baumknoten zu mappen. Zusätzlich müssen Sie auch den Baumknoten Öffnen bzw. Schließen-Button mappen.

Klasse	Notwendige Komponenten / Unterelemente
Tree	Stellt den Baum dar, beinhaltet die Einträge.
TreeNode	Stellt einen Baumknoten dar.
Expander:TreeNodeExpander	Stellt den Baumknoten Öffnen bzw. Schließen Button dar.
	Optionale Unterelemente
Spacer:TreeNodeSpacer	(Optional) Wird verwendet um die Einrückung auf die richtige Ebene zu bewerkstelligen.
CheckBox:TreeNodeCheckBox	(Optional) Stellt eine CheckBox unter einem Baumknoten dar.
Icon:TreeNodeIcon	(Optional) Stellt ein Icon unter einem Baumknoten dar.

Tabelle 51.2: Mapping von Bäumen

Falls QF-Test die Einrückungsebenen nicht richtig erkennt oder Probleme beim Öffnen eines Knotens hat, können Sie die Konfigurationskategorie `treeResolver`[1095] des CustomWebResolver installieren entsprechend parametrisieren. Alternativ können Sie Das `TreeIndentationResolver` Interface[1186] zur Konfiguration der Einrückungserkennung verwenden.

Beispiel:

Im Demo ”CarConfigurator Web” befindet sich ein Baum. Um diesen zu öffnen gehen Sie über das Menü Einstellungen→Sondermodelle..., wählen ein Modell und klicken dann auf den Button ”Details”. Sie finden die dazugehörige Testsuite unter `qftest-9.0.0/demo/carconfigWeb/carconfigWeb_de.qft`

Wenn Sie sich die aufgenommenen CSS-Klassen in QF-Test ansehen oder die Seite mit dem Abschnitt 5.12.2[108] analysieren werden Sie folgende HTML-Struktur finden (etwas vereinfacht und gekürzt):

```
<table id="DetailsTree" class="tree">
  <tbody>
    <tr class="treenode even">
      <td>Information</td>
      <td></td>
    </tr>
    <tr class="treenode odd>
      <td>
        <span>
          <span class=treenodespacer">  </span>
          <span class="treetoggler">
            <img src="include/collapse.png" onclick="javascript: ..."></img>
          </span>
          <span>Zubehör</span>
        </span>
      </td>
    </tr>
    <tr class="treenode odd>
      <td>
        <span>
          <span class=treenodespacer">
                  
            </span>
          <span>Zentralverriegelung</span>
        </span>
      </td>
    </tr>
  </tbody>
</table>
```

Beispiel 51.22: HTML Tree

Wenn Sie dann in der Prozedur `startStop.start` die letzte Sequenz, `CustomWebResolver installieren`, ansehen,

Abbildung 51.7: CarConfigurator Web

sind dort unter anderem folgende Zuweisungen für das Mappen des Baums enthalten:

```
genericClasses:
- Tree: tree
- TreeNode: treenode
- Expander:TreeNodeExpander: treetoggler
- Spacer:TreeNodeSpacer: treenodespacer
```

Beispiel 51.23: HTML Tree

51.1.5 CustomWebResolver – TreeTable

TreeTables (Baumtabellen) sind eine Kombination aus Tabelle und Baum. Dafür mappen Sie `TreeTable` analog zu einer normalen Tabelle. Zusätzlich müssen Sie die ”Baumknoten Öffnen” bzw. ”Schließen”-Buttons zuweisen.

QF-Test geht davon aus, dass die erste Spalte der Tabelle die Baum-Komponenten enthält. Sie können dieses Verhalten über die Konfigurationskategorie `treetableResolver`(1096) im CustomWebResolver installieren anpassen.

Falls QF-Test im Baum die Einrückungsebenen nicht richtig erkennt oder Probleme beim Öffnen eines Knotens hat, können Sie die Konfigurationskategorie `treeResolver`(1095)

im CustomWebResolver installieren entsprechend parametrisieren. Alternativ können Sie Das `TreeIndentationResolver` Interface[1186] zur Konfiguration der Einrückungserkennung verwenden.

Klasse	Notwendige Komponenten / Unterelemente
TreeTable	Stellt den TreeTable dar, beinhaltet die Einträge.
TableRow	Stellt eine Tabellenzeile dar.
TableCell	Stellt eine Tabellenzelle bzw. einen Baumknoten dar.
TableHeader	Stellt die Zeile mit den Überschriften dar.
TableHeaderCell	Stellt eine Überschrift dar.
Expander:TreeNodeExpander	Stellt den Baumknoten Öffnen bzw. Schließen Button dar.
	Optionale Unterelemente
TreeNode	(Optional) Repräsentiert einen Baumknoten, muss innerhalb einer TableCell liegen.
Spacer:TreeNodeSpacer	(Optional) Wird verwendet um die Einrückung auf die richtige Ebene zu bewerkstelligen.
CheckBox:TreeNodeCheckBox	(Optional) Stellt eine CheckBox unter einem Baumknoten dar.
Icon:TreeNodeIcon	(Optional) Stellt ein Icon unter einem Baumknoten dar.

Tabelle 51.3: Mapping von TreeTables

Beispiel:

Ein TreeTable könnte im HTML-Code etwa so aussehen:

```
<table id="DetailsTreeTable" class="treetable">
  <thead>
    <tr>
      <th>Information</th>
      <th>Details</th>
    </tr>
  </thead>
  <tbody>
    <tr class="treenode odd>
      <td>
        <span class="treetoggler"></span>
        <span>Item 1</span>
      </td>
      <td>Lorem ipsum</td>
    </tr>
    <tr class="treenode odd>
      <td>
        <span class=treenodespacer">      </span>
        <span>Item 1.1</span>
      </td>
      <td>Dolor sit amet</td>
    </tr>
  </tbody>
</table>
```

Beispiel 51.24: HTML TreeTable

Die zugehörige Konfiguration des `CustomWebResolver installieren` Knotens könnte folgendermaßen aussehen:

```
genericClasses:
- TreeTable: treetable
- TableRow: <TR>
- TableCell: <TD>
- TableHeader: <THEAD>
- TableHeaderCell: <TH>
- Expander:TreeNodeExpander: treetoggler
```

Beispiel 51.25: TreeTable CWR-Konfiguration

51.1.6 CustomWebResolver – Liste

Für das Mappen von Listen ist es notwendig sowohl die Listenkomponente selbst, also diejenige, welche alle Einträge beinhaltet, sowie die einzelnen Listeneinträge zu mappen.

Klasse	Notwendige Komponenten / Unterelemente
List	Stellt die Liste dar, beinhaltet alle Listeneinträge.
Item:ListItem	Stellt einen Listeneintrag dar.
	Optionale Unterelemente
CheckBox:ListItemCheckBox	(Optional) Stellt eine CheckBox unter einem Listeneintrag dar.
Icon:ListItemIcon	(Optional) Stellt ein Icon unter einem Listeneintrag dar.

Tabelle 51.4: Mapping von Listen

Im Falle von ComboBoxen können Sie auch die spezielle ComboBox Liste spezifizieren mittels `List:ComboBoxList` und `Item:ComboBoxListItem`.

Beispiel:

Folgender HTML-Code bildet eine Liste ab:

```
<ul class="datalist">
  <li class="list-item">Eintrag A</li>
  <li class="list-item">Eintrag B</li>
  <li class="list-item">Eintrag C</li>
  <li class="list-item">Eintrag D</li>
  <li class="list-item">Eintrag E</li>
</ul>
```

Beispiel 51.26: HTML List

Hier hat man die Auswahl, ob man die HTML-Tags mappt oder die CSS-Klassen. Es ist sicherer, die CSS-Klassen zu nehmen. Bei `datalist` spricht vieles dafür, dass sich diese Klasse nur auf Listen im Sinne von QF-Test bezieht. Bei `list-item` können wir sicherheitshalber `ancestor: List` hinzufügen, um sicherzugehen, dass wir hier nicht Unterelemente anderer komplexer Komponenten mappen. (z.B. bei hier nicht dargestellten Tabellen, deren Zellen durchaus auch die CSS-Klasse `list-item` haben könnten.)

Man könnte auch die HTML-Tags mappen. Dies birgt nur die große Gefahr, dass diese im HTML-Code an anderer Stelle für andere Komponenten verwendet werden und dann das Mapping in QF-Test nicht passt. Wenn Ihnen so etwas passiert und Sie nicht herausfinden, warum ein Mapping nicht greift, ist es hilfreich, sich die vorhandenen Mappings über die Prozedur `qfs.web.cwr.dumpConfiguration` anzeigen zu lassen.

```
genericClasses:
- List: datalist
- Item:ListItem:
    class: list-item
    ancestor: List
```

Beispiel 51.27: HTML Liste

51.1.7 CustomWebResolver – Combobox

Comboboxen bieten die Herausforderung, dass es einen Teil für die Texteingabe bzw. das Öffnen einer Auswahlliste gibt sowie die angezeigte Liste mitsamt ihren Einträgen. Eine Listauswahl wird entsprechend immer mit zwei Mausklicks aufgezeichnet. Der erste Klick, der die Liste öffnet, und der zweite Klick, der den Listeneintrag auswählt.

Ein HTML `SELECT` Knoten wird automatisch zu einer ComboBox gemappt und die Listenauswahl wird als ein Auswahl Knoten aufzeichnet.

Klasse	Notwendige Komponenten / Unterelemente
ComboBox	Container Element, das sowohl das Textfeld der ComboBox wie auch den Button beinhaltet.
List:ComboBoxList	Stellt die Liste dar, beinhaltet alle Listeneinträge.
Item:ComboBoxListItem	Stellt einen Listeneintrag dar.
	Optionale Unterelemente
Button:ComboBoxButton	(Optional) Stellt den Button dar, der die Liste öffnet.
TextField:ComboBoxTextField	(Optional) Stellt das TextField dar, das Texteingaben empfängt.
CheckBox:ComboBoxListItemCheckBox	(Optional) Stellt eine CheckBox unter einem Listeneintrag dar.
Icon:ComboBoxListItemIcon	(Optional) Stellt ein Icon unter einem Listeneintrag dar.

Tabelle 51.5: Mapping von ComboBoxen

Für die Liste würde es auch ausreichen reine Listen und Listeneinträge zu mappen, also `List` und `Item:ListItem`, siehe Abschnitt 51.1.6(1103).

Beispiel:

HTML-Code für eine ComboBox:

```
<div class="combobox-wrapper">
  <div role="combobox" aria-expanded="true" aria-owns="ex1-listbox"
    aria-haspopup="listbox" id="ex1-combobox">
    <input type="text" aria-autocomplete="list" aria-controls="ex1-listbox"
      id="ex1-input" aria-activedescendant="">
  </div>
  <ul aria-labelledby="ex1-label" role="listbox" id="ex1-listbox"
    class="listbox">
    <li class="result" role="option" id="result-item-0">Leek</li>
    <li class="result" role="option" id="result-item-1">Lemon</li>
  </ul>
</div>
```

Beispiel 51.28: HTML ComboBox

Eine `ComboBox` besteht aus dem Container-Element, in dem die aktuelle Auswahl angezeigt wird, sowie aus einer Liste, die die auswählbaren Werte enthält. In unserem Beispiel ist diese Liste im dem `DIV`-Objekt enthalten, das auch die `ComboBox` selbst enthält. Oft ist die Liste jedoch an einer ganz anderen Stelle im DOM definiert. Diese finden Sie, indem Sie die Liste öffnen und sie dann im Analyse-Tool des Browsers betrachten. Die CSS-Klassen des Listenobjekte und der Listeneinträge können Sie auch herausfinden, indem Sie mit QF-Test einen Klick auf die Liste aufnehmen und die `qfs:class` Informationen in der Weitere Merkmale Tabelle der aufgenommenen Komponente auswerten.

```
genericClasses:
- List:ComboBoxList: listbox
- Item:ComboBoxListItem:
    class: result
    ancestor: List:ComboBoxList
- ComboBox: role=combobox
```

Beispiel 51.29: HTML ComboBox

Bei dieser `ComboBox` haben wir verschiedene Möglichkeiten, diese in QF-Test zu definieren. Wenn man Kontakt zu den Anwendungsentwicklern hat, ist es am sichersten, bei diesen nachzufragen, welches Kriterium eine bestimmte Komponentenklasse identifiziert. Ist dies nicht der Fall, sollte man sich für ein aller Wahrscheinlichkeit nach eindeutiges Kennzeichen entscheiden und dieses bei anderen Komponenten der gleichen Klasse gegenprüfen. Außerdem sollte man bei ähnlichen Komponentenklassen nachsehen, ob diese nicht dummerweise ebenfalls das ausgewählte Kennzeichen verwenden. In diesem Fall wählt man dann ein anderes, hoffentlich eindeutiges Kennzeichen oder konkretisiert die Zuweisung durch Hinzufügen von `ancestor:`...

Konkret bei diesem Beispiel ist `role=combobox` für die ComboBox selbst vermutlich eindeutig. Ebenso `role=listbox` oder alternativ `css: listbox` für die Liste. Für

welches der beiden man sich entscheidet, dürfte gleichgültig sein. Bei Überschneidungen mit normalen Listen kann auch der Klassentyp weggelassen werden. D.h. die Zuweisungen wären nur `List: listbox` und `Item:ListItem:`. Auf die Funktionalität innerhalb QF-Test hat dies keinen Einfluss.

Bei dem Listeneintrag könnten jedoch sowohl `class=result` als auch `role=option` bei anderen Elementen eingesetzt sein. Daher wird sicherheitshalber `ancestor: List:ComboBoxList` hinzugefügt, um eine Eindeutigkeit herzustellen.

51.1.8 CustomWebResolver – TabPanel und Accordion

Für das Mappen von TabPanels bzw. Accordions ist es wiederum notwendig sowohl die Hauptkomponente selbst, diejenige, die alle Einträge enthält, wie auch die einzelnen Elemente zu mappen.

Klasse	Notwendige Komponenten / Unterelemente
TabPanel	Stellt das TabPanel dar, beinhaltet die Einträge.
Item:TabPanelItem	Stellt einen Tab dar.
	Optionale Unterelemente
Panel:TabPanelContent	(Optional) Stellt ein Panel dar, das den Inhalt des angezeigten Tabs enthält.
Closer:TabPanelCloser	(Optional) Stellt einen Closer Button innerhalb des Tabs dar.
CheckBox:TabPanelCheckBox	(Optional) Stellt eine CheckBox unter einem Listeneintrag dar.
Icon:TabPanelIcon	(Optional) Stellt ein Icon unter einem Listeneintrag dar.

Tabelle 51.6: Mapping von TabPanels

Im Falle von Accordions lauten die Klassen entsprechend Accordion, Item:AccordionItem etc.

Beispiel:

Der folgende HTML-Code bildet ein TabPanel ab. Wie man auch in vielen Implementierungen von Webseiten findet, sind etliche zusätzliche Ebenen eingezogen, die für die Funktionalität der Tabs völlig irrelevant sind. Einige der Ebenen haben Merkmale, die man für die Zuweisung nutzen kann, für alle TabPanel-Elemente sogar redundant. Sie finden im Anschluss daher zwei Varianten eines CustomWebResolver installieren Knoten, die völlig unterschiedlich konfiguriert sind, für die Komponentenaufnahme bei QF-Test jedoch das gleiche Ergebnis liefern.

```
<div role="tab-container">
  <div class="tabs">
    <div>
      <div>
        <ul>
          <div class="tab-bar">
            <div>
              <li>
                <div type="tab">Tab1</div>
              </li>
              <li>
                <div type="tab">Tab2</div>
              </li>
              <li>
                <div type="tab">Tab3</div>
              </li>
            </div>
          </div>
        </ul>
        <div class="content">
          <div role="tab-content" id="Tab1">
            <label>Dies ist das erste TabPanel</label>
          </div>
        </div>
        <div class="content">
          <div role="tab-content" id="Tab2">
            <label>Dies ist das zweite TabPanel</label>
          </div>
        </div>
        <div class="content">
          <div role="tab-content" id="Tab3">
            <label>Dies ist das dritte TabPanel</label>
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

Beispiel 51.30: HTML TabPanel

```
genericClasses:
- TabPanel: tabs
- Panel:TabPanelContent: content
- Item:TabPanelItem: type=tab
ignoreTags:
- <DIV>
- <SPAN>
- <UL>
- <LI>
```

Beispiel 51.31: HTML TabPanel Variante 1

Der zweite Aufruf ist zwar anders parametrisiert, bewirkt aber die gleiche Komponentenaufnahme seitens QF-Test:

```
genericClasses:
- TabPanel: role=tab-container
- Panel:TabPanelContent: role=tab-content
- Item:TabPanelItem:
    tag: li
    ancestor: TabPanel
ignoreTags:
- <DIV>
- <SPAN>
- <UL>
```

Beispiel 51.32: HTML TabPanel Variante 2

51.1.9 Beispiel für den ”CarConfigurator Web”

Nachdem die theoretische Beschreibung des Vorgehens nicht einfach zu verstehen ist, zeigt dieser Abschnitt eine Beispielimplementierung für das ”CarConfigurator Web” Demo. Sie finden die dazugehörige Testsuite unter `qftest-9.0.0/demo/carconfigWeb/carconfigWeb_de.qft`.

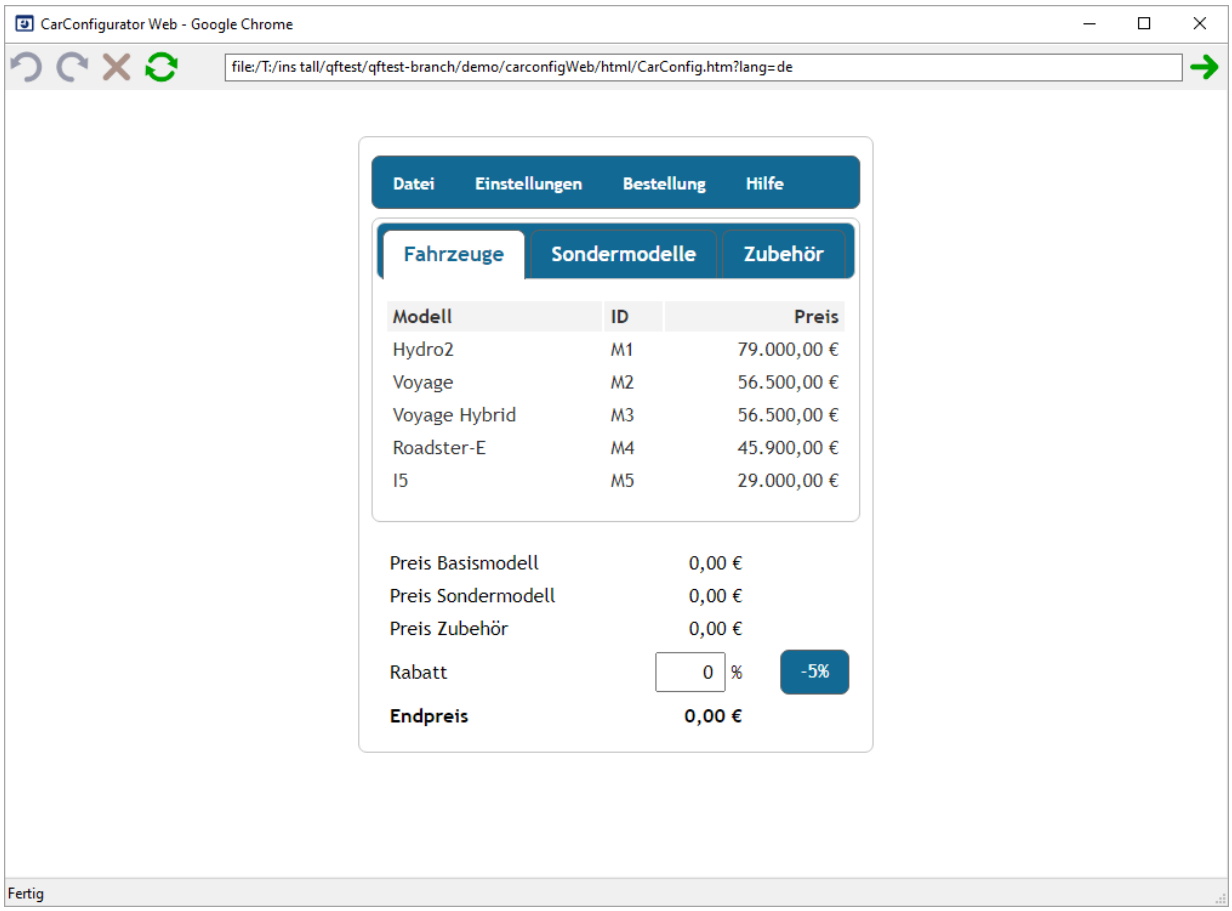
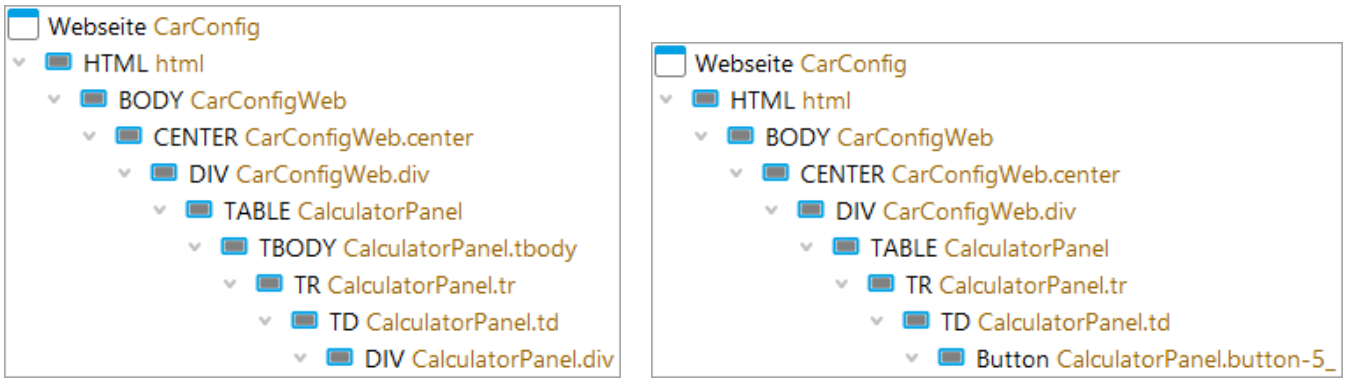


Abbildung 51.8: CarConfigurator Web

Wie im vorigen Abschnitt besprochen, müssen wir zuerst herausfinden, welches Attribut uns die relevanten Informationen liefert, um nachher zu den generischen Klassen von QF-Test zu kommen.

Einfache Klassenzuweisung auf Basis des `class` Attributs

Für den Anfang wollen wir den '-5%' Button in der rechten unteren Ecke auf die generische Klasse 'Button' abbilden. Die aufgeführten Tabelle visualisiert unsere Zielsetzung. Auf der linken Seite finden Sie die aktuelle Aufzeichnung und auf der rechten Seite finden Sie unsere Wunschaufzeichnung.



Aktuelle Aufzeichnung Verbesserte Aufzeichnung

Abbildung 51.9: Verbesserung durch einfache Klassenzuweisung

Zuerst sollten Sie einen Text-Check oder einen Mausklick auf diesen Button aufzeichnen. Danach springen Sie mittels Rechtsklick und Auswahl von Komponente finden zur aufgezeichneten Komponente. Hier können wir sehen, dass wir eine Komponente von der Klasse `DIV` mit einem leeren Namen aufgezeichnet haben. In den weiteren Merkmalen können wir nichts Nützliches finden. Besonders auffallend ist, dass nirgends die Information über den aktuellen Text von '-5%' auffindbar ist. Demzufolge hat QF-Test keine guten Anhaltspunkte um diese Komponente stabil wiederzuerkennen. Es bleiben nur die Geometrie- und Strukturinformationen. Jetzt versuchen wir diese Komponenten zum einen für QF-Test stabiler erkennbar und zum anderen ihre Benennung lesbarer zu machen.

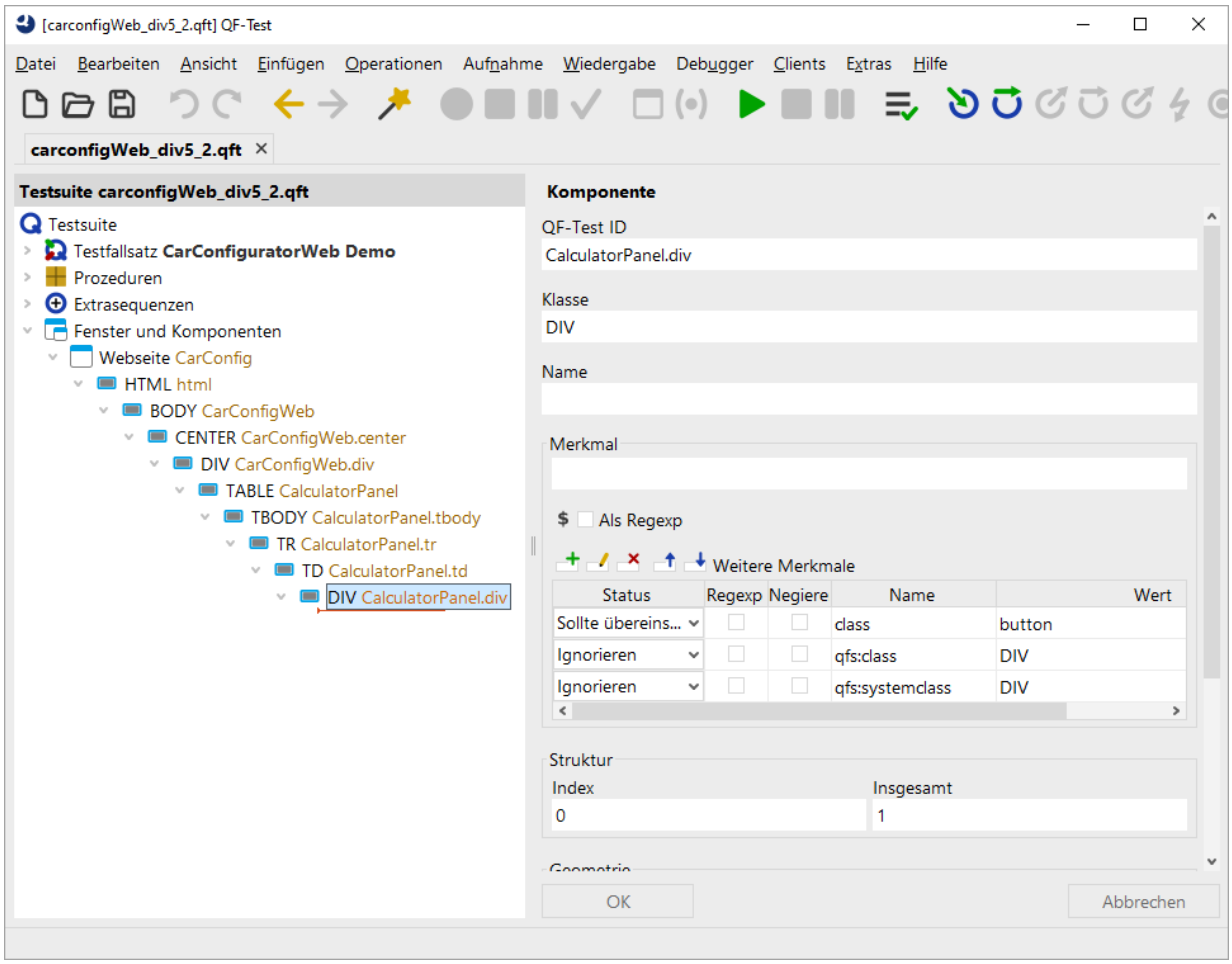


Abbildung 51.10: Aufzeichnung des ’-5%’ Buttons im ”CarConfigurator Web”

Wenn wir uns die Komponente näher anschauen, können wir in den weiteren Merkmalen das Merkmal `class` mit dem Wert `button` finden. Wir können also annehmen, dass jeder Button dieses spezielle Attribut besitzt. Insbesondere, wenn wir unsere Annahme an weiteren Buttons der Applikation überprüft haben.

Nun fügen wir einen CustomWebResolver installieren Knoten unter den Extrasequenzen ein. Nachdem wir vorhin herausgefunden haben, dass das `class` Attribut die relevanten Informationen beinhaltet, können wir der Kategorie `genericClasses` die Zuweisung `Button: button` hinzufügen. Der Ausdruck `Button: button` bedeutet, dass jede Komponente, die den Wert `button` im Attribut `class` besitzt, von nun an die generische Klasse `Button` zugewiesen bekommt. Diese Zuweisung wird bei der nächsten Aufzeichnung QF-Test nun veranlassen, das Standardverhalten für Buttons bei dieser Komponente heranzuziehen. Führen Sie hierzu den CustomWebResolver installieren Knoten aus und zeichnen die Komponente nochmals auf. Sie werden jetzt folgende Aufzeichnung erhalten:

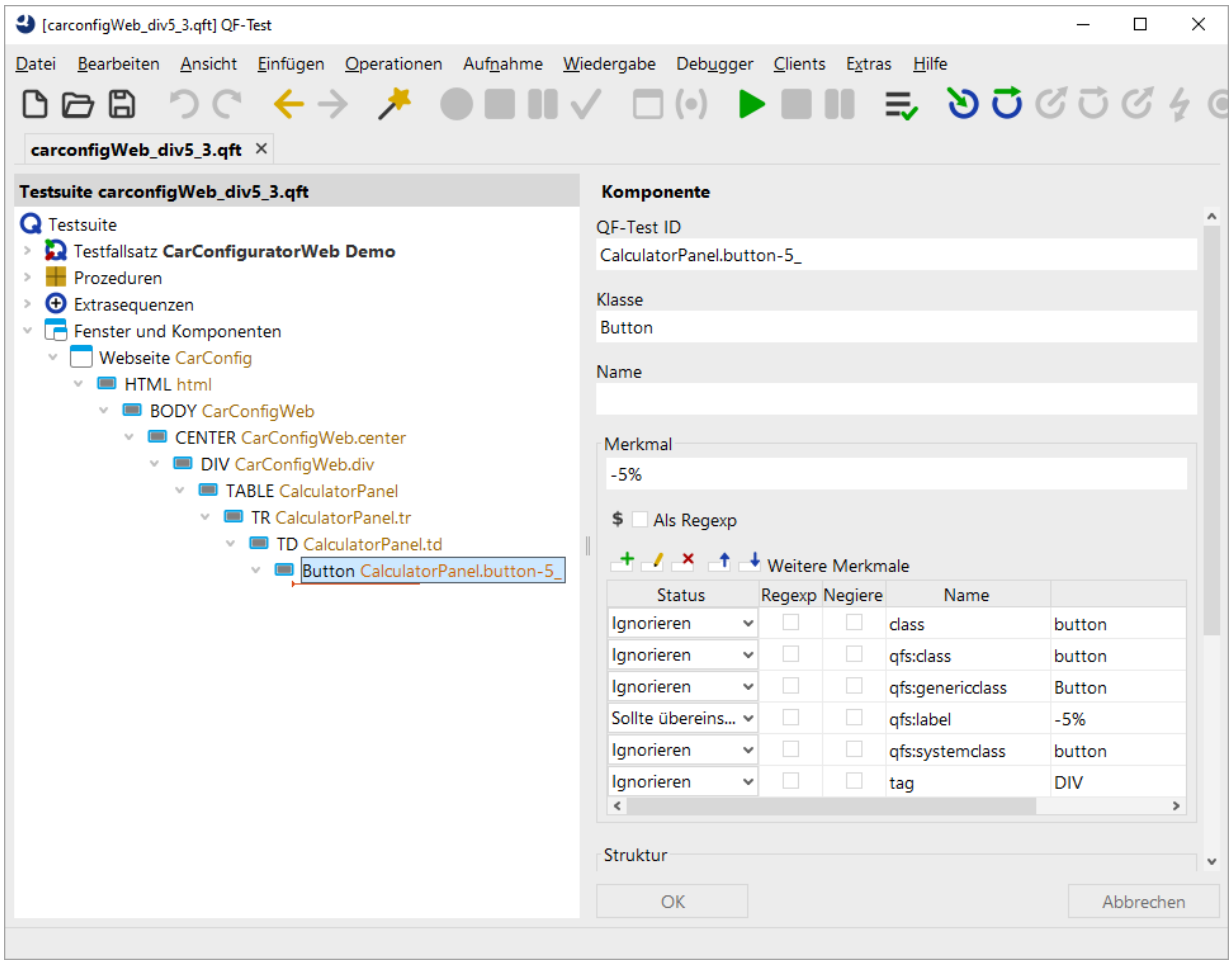


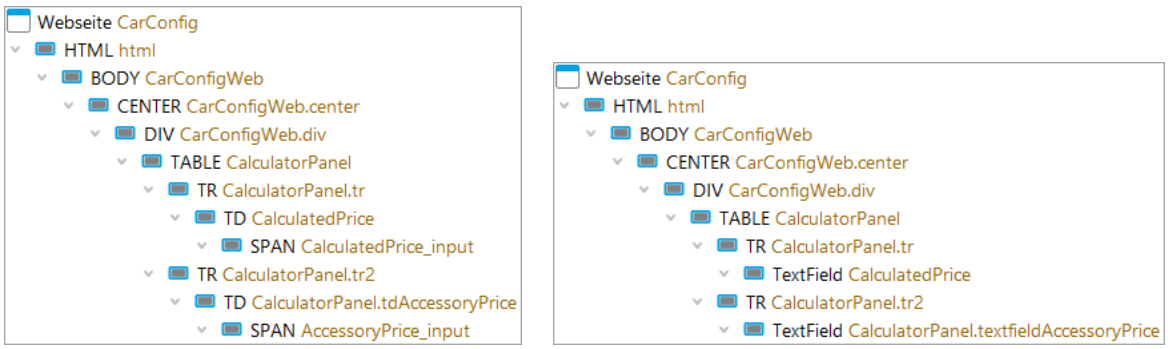
Abbildung 51.11: Aufzeichnung mit genericClasses im ”CarConfigurator Web”

Bei dieser neuen Aufzeichnung erhalten Sie einen Klick auf eine Komponente mit der QF-Test ID `button-5_`. Diese Komponente hat nun die Klasse `Button` und der Text ’-5%’ wurde als Merkmal und sogar als weiteres Merkmal `qfs:label` erkannt. Diese Komponente wird von QF-Test ab jetzt als Button behandelt. Natürlich sollten Sie den Entwicklern immer noch vorschlagen, eine vernünftige ID zu vergeben und somit eine Eindeutigkeit bei der Komponentenerkennung zu erreichen, weil die ID von QF-Test als Name der Komponente verwendet wird.

Diese ’einfache’ Zuweisung kann in vielen Fällen bereits ausreichend sein, besonders für Buttons, MenuItems oder Checkboxen. Falls in Ihrem Fall das Attribut nicht `class` sondern z.B. `role` ist, dann fügen Sie der Kategorie `genericClasses` eine Zuweisung wie `Button: role=button` hinzu. In einigen Fällen kann aber die Information über die Klasse nicht in der letzten Komponente im Baum, sondern in einer Komponente weiter oben in der Hierarchie zu finden sein. Wie man so eine Situation auflöst, finden Sie im nächsten Abschnitt.

Fortgeschrittene Klassenzuweisung

Nach dem einfachen Beispiel im vorherigen Abschnitt, schauen wir uns jetzt ein komplexeres Szenario an. Hierfür versuchen wir die Textfelder, die die Preisinformationen beinhalten, z.B. das Endpreisfeld, zu analysieren. Wie auch vorher sollten wir nun ein paar Mausklicks bzw. Checks auf diese Felder aufzeichnen und uns im Anschluss die aufgezeichneten Komponenten ansehen. Auch hier finden wir die aktuelle Situation und die Zielsetzung in der unteren Tabelle.



Aktuelle Aufzeichnung | Verbesserte Aufzeichnung

Abbildung 51.12: Verbesserung durch fortgeschrittene Klassenzuweisung

Es wurden einige `SPAN` Knoten aufgezeichnet. Diese Komponenten haben allerdings kein `class` Attribut. Jedoch sehen wir, dass diese zumindest ein `id` Attribut zugewiesen bekommen haben. In diesem Fall können wir sagen, dass diese `id` sehr spezifisch für das jeweilige Textfeld ist. Wenn Sie die Elternkomponente, den `TD` Knoten, selektieren, finden Sie allerdings wieder das `class` Attribut mit dem Wert `textfield`. Wenn diese Komponente selektiert ist, hebt QF-Test auch das gesamte Textfeld auf der Webseite hervor. Wir können also annehmen, dass eine solche Komponente mit dem Wert `textfield` für das `class` Attribute für ein echtes Textfeld steht.

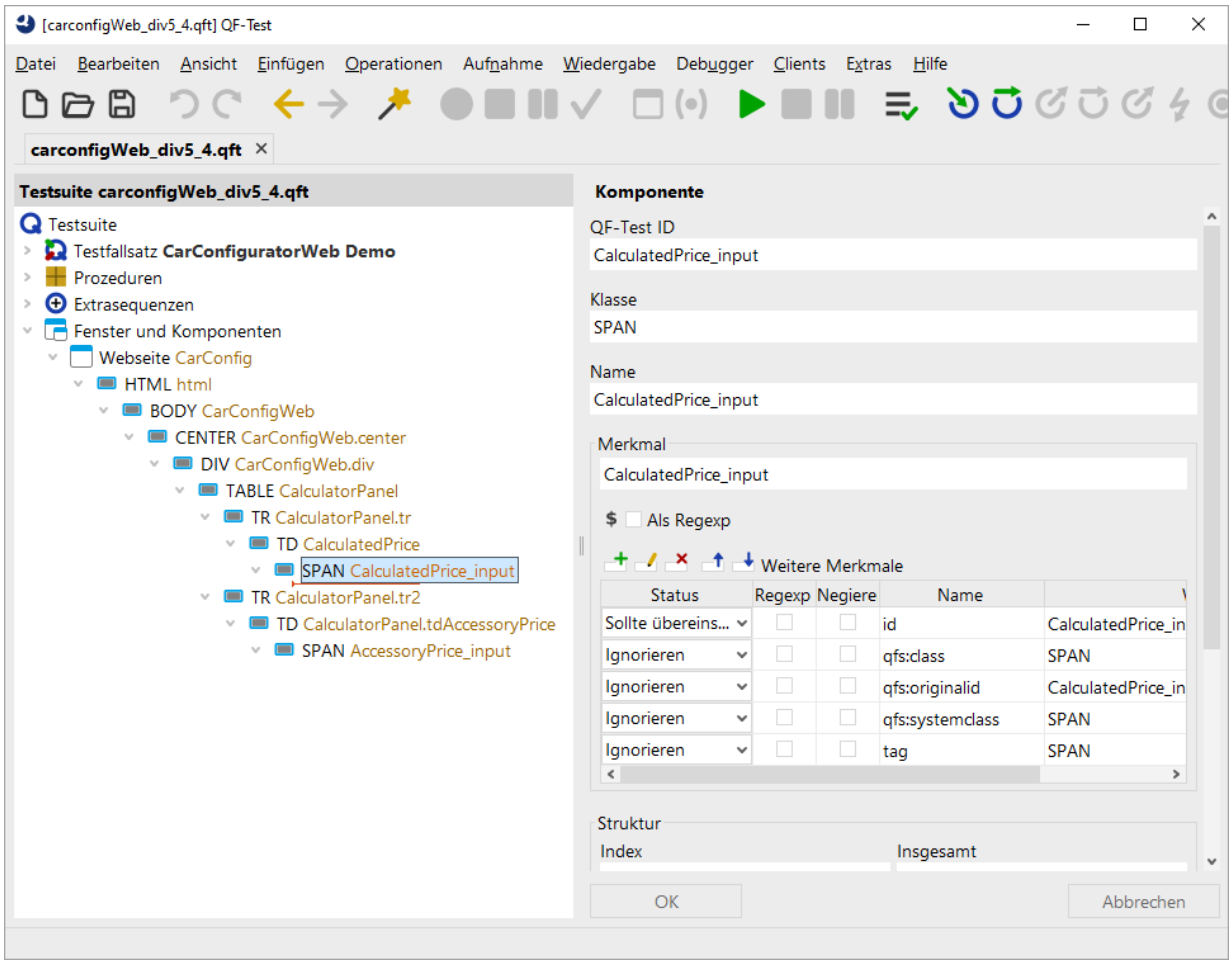


Abbildung 51.13: Aufzeichnung der `SPAN` Komponenten

Nun müssen wir wieder die Konfiguration im CustomWebResolver installieren Knoten anpassen. In diesem Fall müssen wir der Klasse `textfield` von unserer Webseite die generischen Klasse `TextField` zuweisen. Hierfür erweitern wir die Kategorie `genericClasses` um den Eintrag `TextField: textfield`.

Jetzt löschen Sie die zuvor aufgezeichneten Komponenten, führen den CustomWebResolver installieren Knoten aus und zeichnen die Komponente nochmals auf. Jetzt erhalten Sie folgende Aufzeichnung:

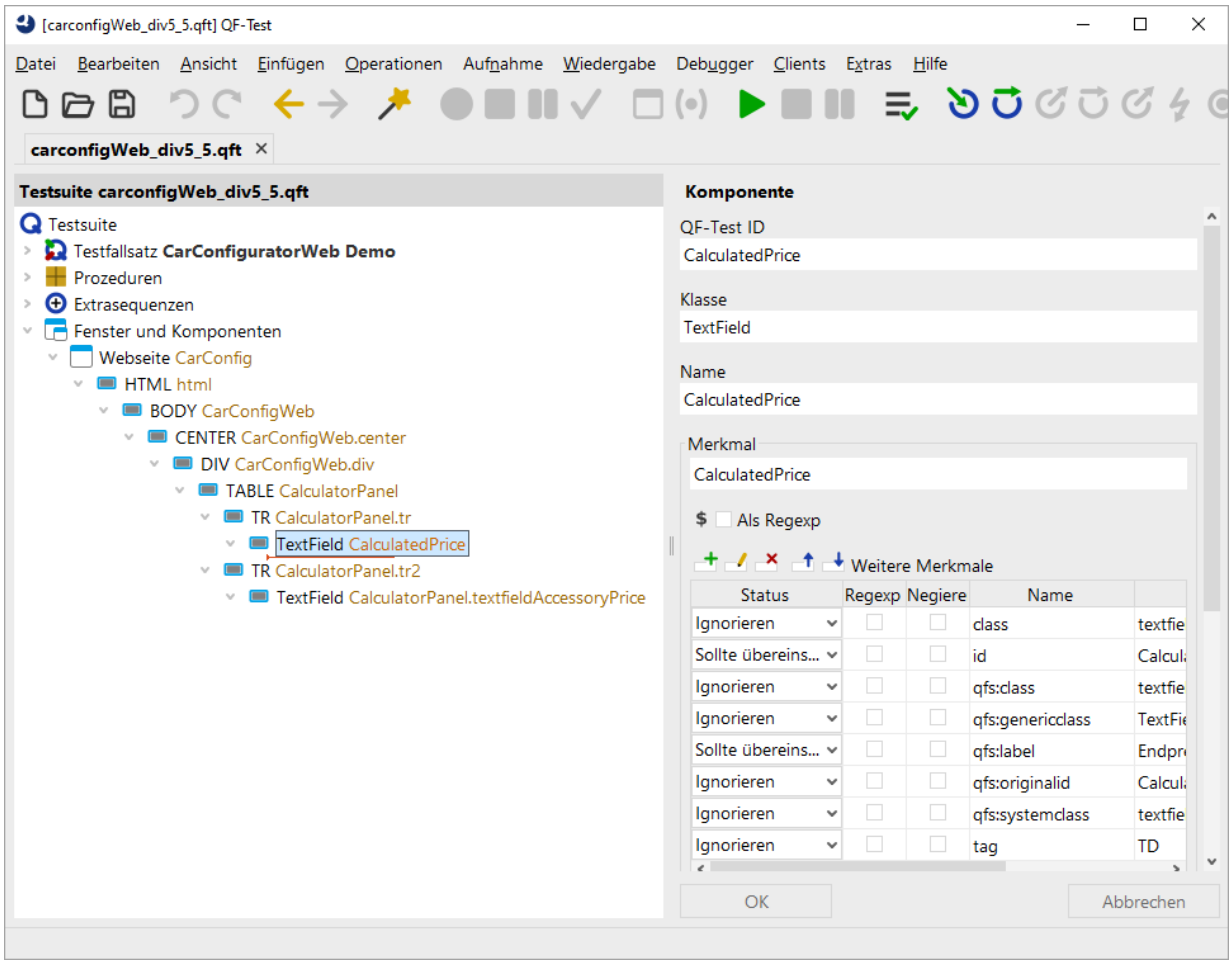


Abbildung 51.14: Aufzeichnung der Textfelder des ”CarConfigurator Web”

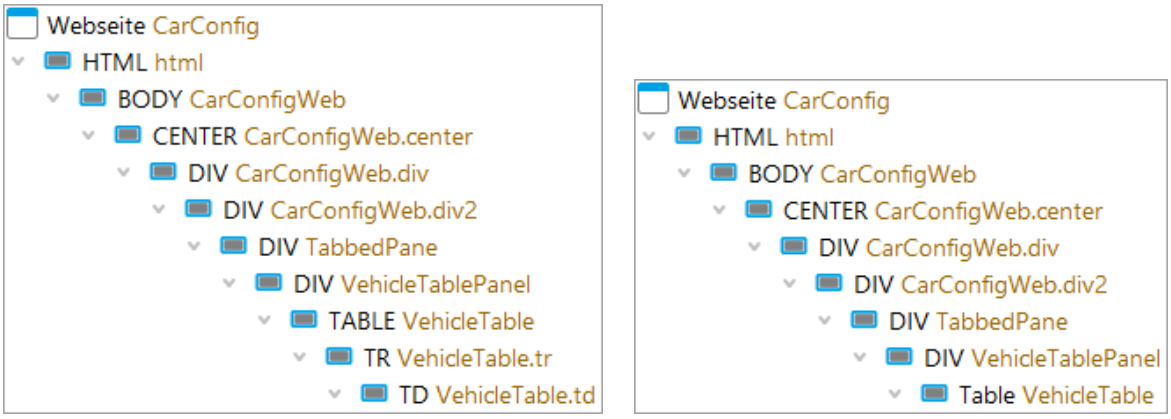
Es wurden die Textfelder wie erwartet aufgezeichnet und es gibt sogar eine Ebene in der Komponentenhierarchie weniger. Zudem haben die Textfelder jetzt die für QF-Test typischen Attribute, wie das weitere Merkmal `qfs:label`.

Im nächsten Abschnitt widmen wir uns der Übersetzung von Komponenten, die Daten beinhalten und deren Daten wir gerne direkt ansprechen würden. Solche Komponenten sind typischerweise Tabellen, Bäume oder Listen und werden als komplexe Komponente in QF-Test bezeichnet.

Klassenzuweisung für komplexe Komponenten wie Tabellen

Der Ansatz aus den vorherigen Abschnitten funktioniert sehr gut für jede Art von Standardkomponente, wie Buttons oder Checkboxen. Aber neben diesen Komponenten, gibt es noch komplexe Komponenten in einer GUI. Diese Komponenten

beinhalten Datensätze, welche wir meistens direkt ansprechen sollten, z.B. mit der Elementsyntax von QF-Test. Solche Komponenten können Tabellen, Bäume oder Listen sein. Hierfür müssen wir sowohl der Komponente wie auch den Datensätzen selbst generische Klassen zuweisen. Welche Einzelkomponenten das im Detail sind, wird in den folgenden Abschnitten beschrieben: CustomWebResolver – Combobox[1105], CustomWebResolver – Liste[1103], CustomWebResolver – Tabelle[1096], CustomWebResolver – TabPanel und Accordion[1107], CustomWebResolver – Baum (Tree)[1099].



Aktuelle Aufzeichnung | Verbesserte Aufzeichnung

Abbildung 51.15: Verbesserung durch Zuweisung komplexer Komponenten

Unser Beispiel bezieht sich auf die Tabelle, die die Fahrzeuge beinhaltet. Anfangs müssen wir wieder einige Mausklicks bzw. Checks auf die angezeigten Fahrzeuge aufzeichnen. Jetzt erhalten wir eine Aufzeichnung wie diese:

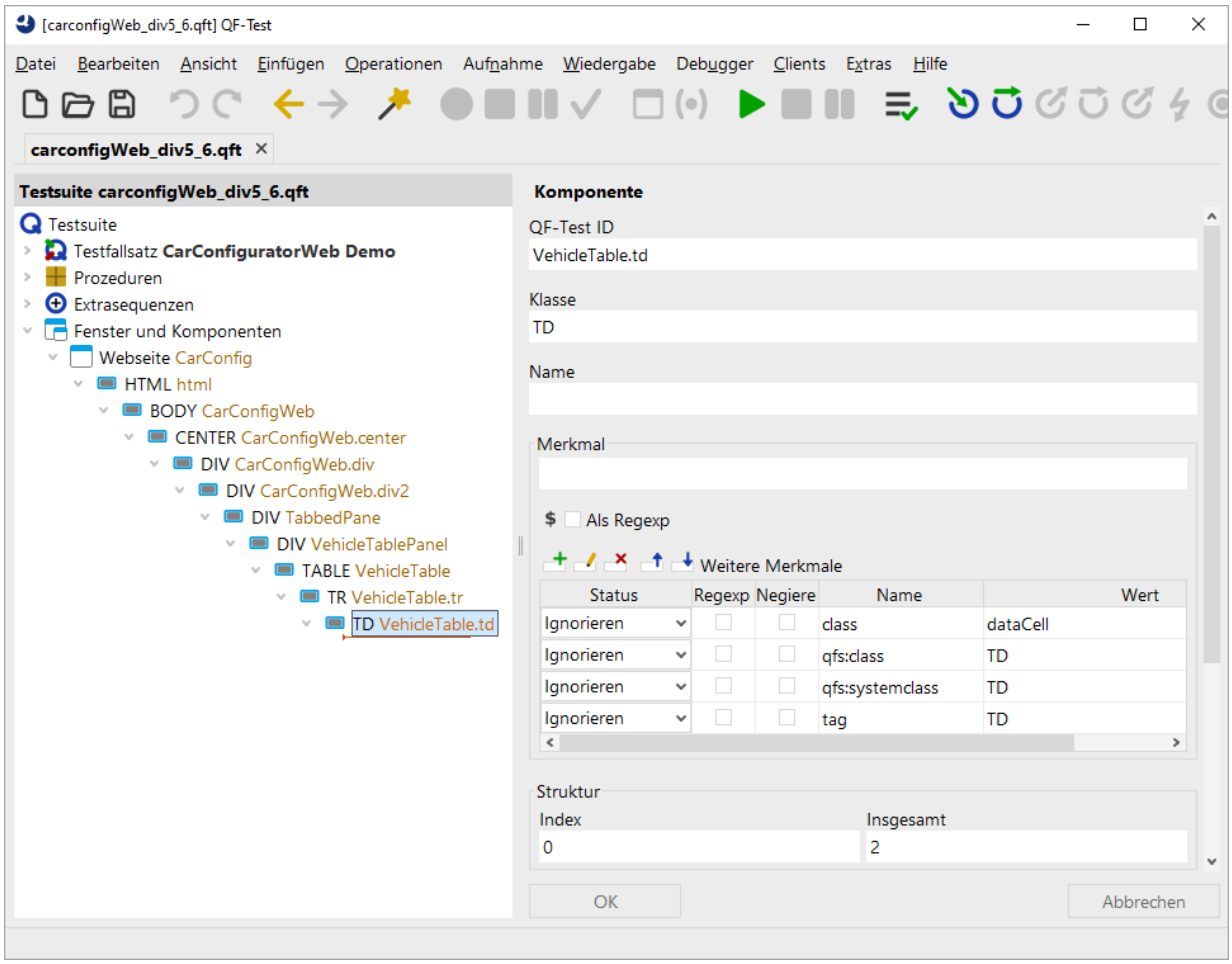


Abbildung 51.16: Aufzeichnung einer Tabelle im ”CarConfigurator Web”

Es wurde ein Klick auf eine `TD` Komponente, die unter einer `TR` Komponente, welche unter einer `TABLE` Komponente liegt, aufgezeichnet. Die aufgezeichnete `TD` Komponente beinhaltet auch das weitere Merkmal `class` mit dem Wert `dataCell`. Bei der `TR` Komponente steht dort `dataRow`, bei der `TABLE` steht dort `dataTable`.

Wenn wir nun schrittweise diese Komponenten selektieren und dabei beobachten, welche Komponente im SUT hervorgehoben werden, kommen wir zu folgendem Schluss:

Eine `TD` Komponente steht hier für eine Tabellenzelle, eine `TR` Komponente steht für eine Tabellenzeile und die `TABLE` Komponente steht für die gesamte Tabelle. Außerdem werden für die korrekte Indexierung über Spaltentitel auch die Überschriftenzeile sowie die einzelne Überschrift benötigt. Die ersten drei Komponenten sehen wir uns jetzt näher an, um diesen eine generische Klassen zuweisen zu können. QF-Test möchte nämlich für Tabellen genau diese fünf Klassen auflösen, siehe CustomWebResolver – Tabelle(1096).

Fangen wir mit dem TABLE Knoten an. Hier gibt es wieder ein class Attribut mit dem Wert dataTable. Das ist ein klares Zeichen, dass dieser Wert dataTable eine Tabelle repräsentiert. Nun gehen wir also wieder zum CustomWebResolver installieren Knoten und ergänzen die Kategorie genericClasses um Table: dataTable.

Nun kommt die Tabellenzeile an die Reihe. Auf der TR Komponente finden wir wieder das class Attribut, diesmal mit dem Wert dataRow. Das ist wiederum ein klares Zeichen. Jetzt müssen wir also wieder den CustomWebResolver installieren Knoten anpassen und ergänzen die Kategorie genericClasses um TableRow: dataRow.

Anschließend nehmen wir uns die TD Komponente vor. Auch hier finden wir wieder das class Attribut, diesmal mit dem Wert dataCell. Also ergänzen wir wieder die genericClasses Kategorie des Knotens, nun um TableCell: dataCell.

Nun möchten wir QF-Test noch bekannt geben, wie die Spaltenüberschriften zu erkennen sind, damit QF-Test bei der Aufnahme diese als Textindex verwenden kann. Wieder gibt es das class Attribut, diesmal mit dem Wert headerRow für die Überschriftenzeile und headerCell für die einzelne Spaltenüberschrift. Wir ergänzen wieder die genericClasses Kategorie des Knotens. Sie lautet nun:

```
genericClasses:
- TableHeader: headerRow
- TableHeaderCell: headerCell
- TableCell: dataCell
- TableRow: dataRow
- Table: dataTable
- TextField: textfield
- Button: button
```

Beispiel 51.33: Kategorie genericClasses

Der nächste Schritt ist die zuvor aufgezeichneten Komponenten zu löschen, den CustomWebResolver installieren Knoten auszuführen, die Webseite neuzuladen und die Klicks nochmals aufzuzeichnen.

Als Resultat bekommen wir eine für QF-Test typische Aufzeichnung eines Elements, wie VehicleTable@Modell&0 (oder jede andere Zeile, je nachdem was Sie angeklickt haben). Die aufgezeichneten Komponenten beinhalten nun nur noch eine Table Komponente ohne weitere Unterkomponenten, denn diese werden ab jetzt als Elemente von QF-Test behandelt.

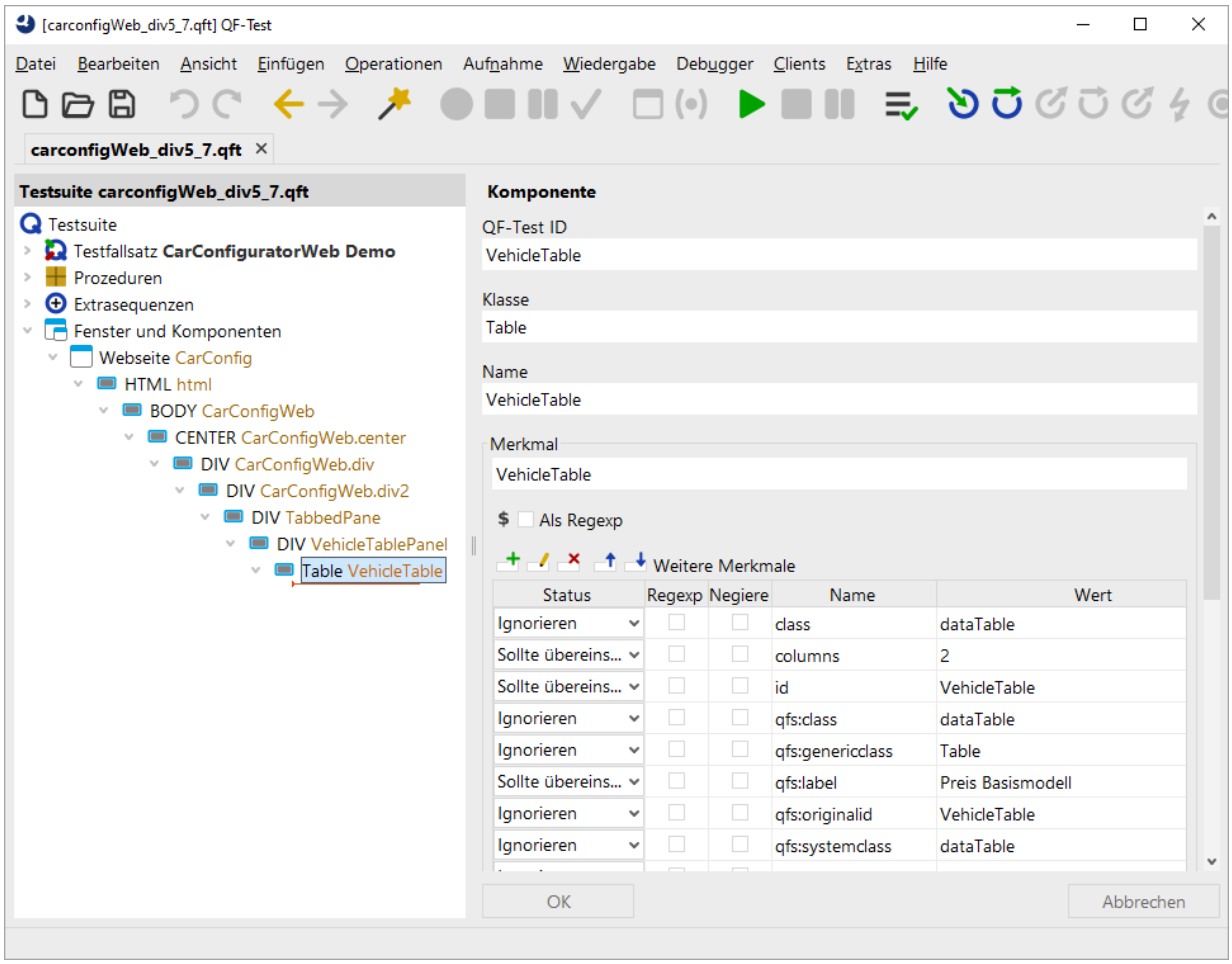


Abbildung 51.17: Aufzeichnung einer aufgelösten Tabelle im ”CarConfigurator Web”

Nachdem wir nun eine komplexe Komponente auflösen konnten, finden Sie eine Beschreibung des weiteren Vorgehens im nächsten Abschnitt.

Weiteres Vorgehen

Wie Sie in den vorigen Abschnitten gesehen haben, besteht also die erste Aufgabe eines Webprojektes, daraus, herauszufinden wie QF-Test die Komponenten der Webseite erkennt und entsprechend diese Übersetzungstabelle zu erstellen. Diese Aufgabe sieht auf den ersten Blick schwierig aus, kann Ihnen aber im späteren Verlauf Ihres Projektes viel Wartungsarbeit bei Änderungen in der Komponentenhierarchie ersparen, weil sich QF-Test eben nur mehr auf die wesentlichen Eigenschaften Ihrer HTML-Seite konzentriert und nicht mehr alle möglichen Informationen auswertet.

Im Kapitel 61 (1329) finden Sie eine vollständige Aufführung aller generischen Klassen für

Komponenten oder komplexe Komponenten wie Listen oder Bäume. Hier sollten Sie natürlich nicht alles mögliche versuchen zu mappen, sondern eher bedarfsorientiert vorgehen. Sie sollten also nur Komponenten, die auch wirklich benötigt werden, versuchen für QF-Test zu übersetzen. Eine spätere Erweiterung des Aufrufes ist möglich, falls neue Komponenten dazu gekommen sind.

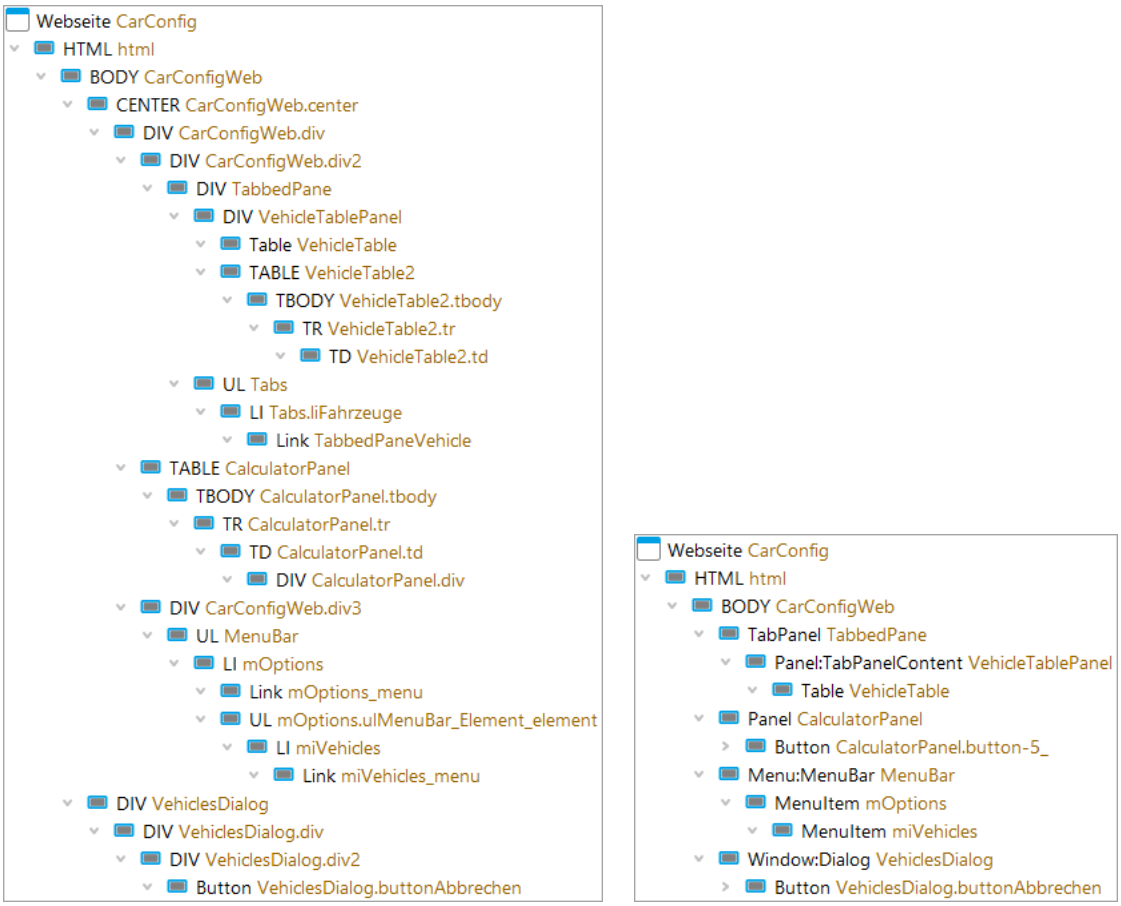
Im vorherigen Beispiel würde man nun die nächsten Komponenten wie Menüs oder Tabs analysieren. Nachdem dies den Rahmen dieses Handbuchs sprengen würde, finden Sie ein Beispiel für eine vollständige Konfiguration des CustomWebResolver installieren Knoten in unserer Demo-Testsuite `qftest-9.0.0/demo/carconfigWeb/carconfigWeb_de.qft` in der Prozedur `startStop.start` in der Sequenz `CustomWebResolver installieren`.

Damit Ihre Übersetzungen zukünftig bei jedem Start der Web-Anwendung verwendet werden, sollten Sie den CustomWebResolver installieren Knoten direkt nachdem der Browser geöffnet wurde ausführen. Wenn Sie den Start Ihrer Browser-Anwendung über den QF-Test Schnellstartassistenen erstellt haben, finden Sie in der erzeugten Vorbereitungssequenz bereits den Knoten in der Sequenz `CustomWebResolver installieren`. Dieser kann dann entsprechend konfiguriert werden.

Abschließende Schritte

Neben der reinen Übersetzung der Webseite nach QF-Test ist es auch möglich, bestimmte Komponenten für die Aufzeichnung zu ignorieren. Hierfür sind die Kategorien `ignoreTags` bzw. `ignoreByAttributes` zuständig. Dieses Ignorieren sollten Sie aber erst einführen, wenn Sie die meisten Komponenten bereits übersetzt haben.

Abschließend möchten wir Ihnen die Unterschiede zwischen der ursprünglichen Aufnahme und so wie der Aufzeichnungen mit zeigen. In der nachfolgenden Tabelle finden Sie wieder links die Aufzeichnungen ohne und rechts finden Sie dieselben Aufzeichnungen mit .



Aktuelle Aufzeichnung Verbesserte Aufzeichnung

Abbildung 51.18: Verbesserte Komponentenaufzeichnung am Beispiel des ”CarConfigurator Web”

51.2 Besondere Unterstützung für verschiedene Web-Komponentenbibliotheken

Modernen Web-Anwendungen sind oft sehr interaktiv und kommen vom Look&Feel her schon fast an Desktop-Applikationen heran. Hinter diesen Anwendungen steckt allerdings ein ganzer Zoo von Frameworks, auf denen diese basieren. Jedes davon setzt eigene Schwerpunkte und kommt mit einem eigenen Satz von Widgets. Solche Webframeworks stellen für QF-Test und jedes andere automatisierte Testwerkzeug aus folgenden Gründen ein Problem dar:

- Die Komponentenhierarchie wird automatisch aus abstrakten Widgets wie But-

ton oder List generiert. Ein Widget wird dabei oft mittels diverser DIV Knoten implementiert. Dies führt zu einer sehr tief verschachtelten Hierarchie mit geringer Struktur.

- IDs werden entweder gar nicht zugewiesen oder automatisch generiert, was für Regressionstests noch problematischer ist.
- Die asynchrone Kommunikation mit dem Webserver und die dynamische Erstellung von DOM Knoten kann Probleme beim Timing verursachen.

Es gibt kein Allheilmittel, mit dem sich diese Probleme generisch lösen lassen. In vielen Fällen kann QF-Test trotzdem ohne weiteres Zutun mit Webframeworks interagieren. Dabei sind Wiedererkennung und Performanz allerdings nicht ideal. Optimale Testbarkeit kann nur mit Hilfe von Spezialfunktionen erreicht werden, die genau auf ein Webframework zugeschnitten sind und sich dessen Eigenheiten zu Nutze machen.

Video

Das Video

'Die Explosion der Komplexität in der Web Testautomatisierung eindämmen'
https://www.qftest.com/de/yt/web-testautomatisierung-40.html

zeigt eindrucksvoll die Reduktion tief geschachtelter DOM-Strukturen durch QF-Test.

Für eine große Anzahl von Webframeworks bietet QF-Test optimierte CustomWebResolver:

Framework Name	Webseite	Kurzname
Angular Material	material.angular.io	`angular`
Ext JS	sencha.com/products/extjs	`extjs`
Fluent UI React	developer.microsoft.com/en-us/fluentui#/	`fluentui`
Flutter Web	flutter.dev/multi-platform/web	`flutter`
Google Web Toolkit (GWT)	gwtproject.org	`gwt`
ICEfaces	icesoft.org	`icefaces`
jQuery UI	jqueryui.com	`jqueryui`
jQuery EasyUI	jeasyui.com	`jeasyui`
Kendo UI for jQuery	www.telerik.com/kendo-jquery-ui	`kendoui`
Prime Faces	primefaces.org	`primefaces`
Qooxdoo	qooxdoo.org	`qooxdoo`
Rich Ajax Platform (RAP)	eclipse.org/rap	`rap`
RichFaces	jboss.org/richfaces	`richfaces`
Smart GWT	smartclient.com	`smartgwt`
Vaadin	vaadin.com	`vaadin`
W3C ARIA	w3.org/WAI/ARIA/apg	`aria`
ZK	zkoss.org	`zk`

Tabelle 51.7: Unterstützte Webframeworks

Der angegebene Kurzname kann in CustomWebResolver installieren Knoten in der Kategorie `base` angegeben werden, siehe Abschnitt 51.1.2[1087]. QF-Test ist sogar in der Lage, automatisch festzustellen, ob Ihre Web-Anwendung eines dieser Frameworks benutzt und einen passenden Resolver, wie unten beschrieben, einzuhängen. Verwenden Sie dazu den Kurznamen `autodetect`.

51.2.1 Konzepte für Webframework-Resolver

Ein Webframework-Resolver besteht aus verschiedenen Resolvern und anderen Funktionen, die speziell für ein bestimmtes Framework entwickelt werden. Vor allem versucht QF-Test, den DOM-Knoten Klassen zuzuweisen, die den zugehörigen abstrakten Widgets entsprechen und Knoten auf Zwischenebenen, die nur ein Detail der Framework-Implementierung sind, aus der Hierarchie herauszufiltern. Die Attribute Name, Merkmal und Weiteres Merkmal werden passend zum Framework ermittelt und Events werden so auf den korrekten DOM-Knoten wiedergegeben, dass dies möglichst exakt einer Aktion von einem echten Anwender entspricht. Diese Maßnahmen reduzieren die Hierarchie der Komponenten drastisch und erhöhen Performanz und Zuverlässigkeit der Wiedererkennung und Wiedergabe gleichermaßen. Zudem werden Timing und Synchronisation besonders unterstützt.

Zwangsläufig unterscheiden sich Komponenten und Events für eine Web-Anwendung mit und ohne aktiviertem Framework Resolver signifikant und sind nicht zueinander kompatibel. Daher sollte die Entscheidung, ob ein Framework Resolver verwendet wird, so früh wie möglich getroffen werden. Andernfalls müssen entweder bestehende Tests neu implementiert werden, nachdem der Resolver aktiviert wird oder Tests mit und ohne Resolver müssen sauber voneinander getrennt werden. Wenn ein Resolver für Ihre Anwendung verfügbar ist, sollten Sie ihn praktisch immer auch einsetzen, außer Ihre Tests sind bereits zu umfangreich, weitgehend vollständig und stabil.

Die Implementierung eines Webframework Resolvers ist ein fortlaufender Prozess. Wenn ein Webframework weiterentwickelt wird, so muss evtl. auch der zugehörige CustomWebResolver angepasst werden. Daher sind die eingebauten CustomWebResolver mit einer Versionsnummer versehen, welche der Versionnummer des Frameworks entspricht, für welche der Resolver ursprünglich entworfen wurde. So lange es nicht zu inkompatiblen Änderungen kommt, kann dieser CustomWebResolver auch für neuere Versionen des Frameworks verwendet werden.

Ältere CustomWebResolver in QF-Test setzen noch auf ein Versionsschema auf, welches von der Version des Web-Frameworks entkoppelt ist. Bei Aktualisierungen wird das neue Versionsschema verwendet.

Webframework-Resolver werden über den CustomWebResolver installieren Knoten aktiviert, bei dem Sie auch die zu verwendende Version mit "." getrennt angeben können. Sie können wahlweise nur die Major Version angeben, in welchem Fall QF-Test die

höchste verfügbare Medium.Minor Version für diese Major Version verwendet. Dies ist üblicherweise die beste Option und wird in der mit Hilfe des Schnellstart-Assistenten erstellten Startsequenz verwendet (vgl. Kapitel 3(32)). Alternativ können Sie die Major.Medium Version festlegen, oder sogar die exakte Major.Medium.Minor Version und so Ihre Tests mit der Version des Resolvers ausführen, mit der sie erstellt wurden.

51.2.2 Eindeutige Bezeichner setzen

Jedes Webframework hat seine eigene Methoden, um eindeutige IDs zu vergeben. In diesem Abschnitt finden Sie die entsprechenden Methoden für die unterstützten Frameworks von QF-Test:

Angular

Die einfachste Variante ist es, das Attribut 'ID' `<div id=”myId”/>` für alle notwendigen Komponenten zu setzen.

Ext JS

IDs können mittels folgendem Code gesetzt werden:

```
var container = Ext.create('Ext.container.Container', {
id: 'MyContainerId',
... });.
```

Als Alternative können Sie auch `container.getEl().set({ 'qfs-id': 'myId' });` aufrufen. In diesem Fall müssen Sie allerdings einen NameResolver implementieren, der 'qfs-id' als Namen für QF-Test setzt.

GWT

Die einfachste Variante ist die Methode `widget.getElement().setId(”myId”);` für die erzeugten Elemente aufzurufen.

Alternativ können Sie `widget.ensureDebugId(”myId”)` aufrufen. Allerdings müssen Sie in diesem Fall die Datei `xxx.gwt.xml` anpassen, damit diese IDs auch wirklich gesetzt werden. Hierfür müssen Sie `<inherits name=”com.google.gwt.user.Debug”/>` eintragen.

Es ist auch möglich ein eigenes Attribut 'qfs-id' für die Identifikation zu setzen. Hierfür rufen Sie die Methode `setAttribute(”qfs-id”, ”myId”)` auf. Dieses Attribut kann nun mittels NameResolver für QF-Test verwendet werden.

ICEfaces

Die einfachste Variante ist es, das Attribut 'ID' `<p:inputText id="myId"/>` für alle notwendigen Komponenten in der xhtml Definition zu setzen.

jQuery UI

Die einfachste Variante ist es, das Attribut 'ID' `<p:inputText id="myId"/>` für alle notwendigen Komponenten zu setzen, bei einem existierenden Element wie folgt: `$(element).attr("id","myId");`

jQuery EasyUI

Die einfachste Variante ist es, das Attribut 'ID' `<p:inputText id="myId"/>` für alle notwendigen Komponenten zu setzen.

Kendo UI für jQuery

Sie müssen das Attribut 'ID' im Sourcecode oder über den grafischen Editor setzen.

PrimeFaces

Die einfachste Variante ist es, das Attribut 'ID' `<p:inputText id="myId"/>` für alle notwendigen Komponenten in der xhtml Definition zu setzen.

Qooxdoo

Es gibt keinen Standardweg. Sie können entweder ein eigenes Attribut in der Generierung des DOMs an die Knoten oder eine Eigenschaft an die `setData` Methode des Qooxdoo Widgets hängen. Diese Eigenschaft können Sie dann per Resolver auswerten.

RAP

Ab RAP Version 2.2 wird ein Name, der mit `widget.setData("name", "myId")` vergeben wird, genau wie bei SWT direkt von QF-Test ausgewertet. Das Feld kann nur benutzt werden wenn es zuvor über `WidgetUtil.registerDataKeys("name");` registriert wurde.

Für RAP Versionen älter als 3.0.0 funktioniert auch die folgende Technik, die aber nicht empfohlen wird, da sie vom SWT Standard abweicht und zusätzlich erfordert, den Webserver mit einer besonderen Option zu starten.:

Rufen Sie die Methode `widget.setData(WidgetUtil.CUSTOM_WIDGET_ID, "myId");` für jedes notwendige Element auf.

Nun müssen Sie noch Ihre Webserver Umgebung mit folgendem zusätzlichen Parameter starten `-Dorg.eclipse.rap.rwt.enableUITests=true`.

Bitte beachten: Der VM Parameter wurde ab RAP 2.0 umbenannt, für RAP Versionen vor 2.0 lautet er `-Dorg.eclipse.rwt.enableUITests=true`.

RichFaces

Sie müssen das Attribut 'ID' im Sourcecode oder über den grafischen Editor setzen.

Smart GWT

Die einfachste Variante ist es, die Methode `widget.setID("id")` für alle notwendigen Komponenten aufzurufen.

Vaadin

Die einfachste Variante ist es, die Methode `widget.setId("id")` (`widget.setDebugId("id")` für Vaadin Version < 7) für alle notwendigen Komponenten aufzurufen.

Sie können auch eine eigene Stylesheet Klasse mittels `widget.setStyleName("qfs-id=myId")` hinzufügen. Diese muss dann in einem NameResolver für QF-Test ausgewertet werden.

ZK

QF-Test verwendet die Widget-ID, welche auch in den `zul` Dateien verwendet wird. Daher sollten die meisten IDs nutzbar sein.

Das ZK Framework bietet auch einen eigenen IDGenerator an, um IDs zu vergeben. Dies kann allerdings ziemlich aufwendig sein, hier ist es besser sich auf die Standardalgorithmen von QF-Test zu verlassen.

51.3 Browser Verbindungsmodus

Um Zugriff auf den Browser zu erlangen, verfolgt QF-Test verschiedene Ansätze, die im Folgenden erläutert werden.

4.1+ Da der QF-Driver Ansatz zur Browser-Anbindung mit Hilfe von Embedding leider nicht von allen Browserherstellern weiter gepflegt bzw. überhaupt unterstützt wird, wurde für QF-Test 4.1 ein neuer Mechanismus implementiert, um auch zukünftige Browser und Browser-Versionen für die Tests verwenden zu können. Dieser Mechanismus verwendet als Bindeglied zwischen dem Browser und QF-Test den Selenium WebDriver.

5.3+ Darüber hinaus existiert mit dem CDP-Driver für Chromium-basierte Browser eine effiziente Alternative zum WebDriver-Ansatz.

Nachfolgende Tabelle listet die Browser mit dem jeweils möglichen Verbindungsmodus auf. QF-Test versucht automatisch den richtigen Modus zu wählen. Sie können die Kontrolle darüber mit Hilfe des Attributs Verbindungsmodus für den Browser(738) im Web-Engine starten(737) Knoten übernehmen.

Browser	Verbindungsmodus	Bemerkung
Chrome	QF-Driver	Eine aktuelle stabile Chromium Version, ist Teil der QF-Test Installation (nur Windows)
Chrome	CDP-Driver	Experimentelle Unterstützung auch für neuere Versionen (siehe Unterstützte Technologien - zu testende Systeme[4])
Chrome	WebDriver	Unterstützung über automatischen ChromeDriver Download. Konkrete Versionen siehe Unterstützte Technologien - zu testende Systeme[4]
Chrome (headless)	CDP-Driver	
Chrome (headless)	WebDriver	
Firefox	WebDriver	Wird vom mitgelieferten GeckoDriver unterstützt, aktuell 102esr und höher
Firefox (headless)	WebDriver	
Microsoft Edge	CDP-Driver	siehe Chrome (CDP-Driver)
Microsoft Edge	WebDriver	
Microsoft Edge (headless)	CDP-Driver	
Microsoft Edge (headless)	WebDriver	
Opera	CDP-Driver	siehe Chrome (CDP-Driver)
Opera	WebDriver	`abgekündigt`
Safari	WebDriver	

Tabelle 51.8: Verbindungsmodus für Browser

51.3.1 QF-Driver Verbindungsmodus

Bei diesem Ansatz wird der auf dem Rechner des Anwenders installierte Browser in ein sogenanntes Wrapper-Fenster eingebunden. Man spricht bei diesem Ansatz auch von Embedding. QF-Test bindet in dieses Fenster den lokal installierten Web-Browser nativ ein und erlangt somit Zugriff auf die Automatisierungs-Interfaces des jeweiligen Browsers. Über diese kann QF-Test dann die benötigten Events vom Browser abgreifen aber auch umgekehrt Events in den Browser einspeisen.

51.3.2 CDP-Driver Verbindungsmodus

5.3+

In Browsern, die auf *Chromium* basieren (Google Chrome, Microsoft Edge und Opera), steht für Test- und Debuggingzwecke eine spezielle Schnittstelle zur Verfügung, welche zum Beispiel auch durch die im Browser eingebetteten Entwicklungswerkzeuge verwendet wird. QF-Test kann ab Version 5.3 diese Schnittstelle und das Chrome DevTools Protokoll nutzen, um mit dem Browser zu kommunizieren und zu interagieren. Für Firefox steht aktuell leider noch keine vollwertige Implementierung der Schnittstelle durch Mozilla zur Verfügung, gleiches gilt für Safari.

51.3.3 WebDriver Verbindungsmodus

WebDriver entwickelt sich zu einem W3C-Standard für die Steuerung von Web-Browsern. (http://www.w3.org/TR/webdriver). Er ist definiert als sogenanntes Remote-Control-Interface, das es erlaubt, die im Browser anzeigten Inhalte zu analysieren und den Browser zu steuern. WebDriver stellt hierfür ein plattform- und sprachunabhängiges Netzprotokoll zur Verfügung.

Die verschiedenen Browser-Hersteller haben sich auf diesen Quasi-Standard geeinigt, was dazu führt, dass die WebDriver Integration direkt von den Browser-Herstellern mitentwickelt wird. Die Integration erfolgt teilweise über Plugins, bei manchen Herstellern ist die Integration bereits direkt mit in der Standard-Installation des Browsers enthalten.

QF-Test nutzt die WebDriver Interfaces um mit dem Browser zu interagieren. Da der WebDriver Ansatz nur teilweise zu den Konzepten von QF-Test passt, wurde die Web-Engine von QF-Test so erweitert, dass auch bei der Anbindung per WebDriver ein möglichst großer Funktionsumfang von QF-Test genutzt werden kann und dem Anwender die Mehrwert-Features von QF-Test wie z.B. Synchronisation, Abstraktion von Komponenten usw. wie gewohnt zur Verfügung stehen.

Hinweis Der Selenium WebDriver setzt ein Java ab Version 8 und höher voraus.

51.3.4 Bekannte Einschränkungen des WebDriver Modus

Die WebDriver Anbindung an QF-Test befindet sich aktuell noch in der aktiven Entwicklung. Daher stehen leider einige Funktionen, die von der QF-Driver Anbindung bekannt sind, noch nicht zur Verfügung. In den meisten Fällen sind diese Einschränkungen dem Umfang der WebDriver Spezifikation zuzuschreiben.

- Keine Unterstützung von Datei-Downloads und HTTP-Authentifizierungs-Dialogen.
- Keine Möglichkeit, HTTP-Requests direkt aufzuzeichnen bzw. abzuspielen.

- wd.getComponent(WebElement) funktioniert bisher noch nicht bei Elementen in verschachtelten Frames.
- Events, die zum Laden einer neuen Webseite führen, werden teilweise nicht richtig aufgezeichnet.
- Event-Synchronisierung ist teilweise verzögert.

51.4 Web – Pseudoattribute

Web

Pseudoattribute sind dafür gedacht, Resolver zu vereinfachen, die über JavaScript Werte aus dem Browser ermitteln. Ein Pseudoattribut wird in QF-Test registriert und erhält seinen Wert aus dem Ergebnis einer Javascript-Code-Ausführung.

Attribute von HTML-Elementen können in einem SUT-Skript mit der Methode `getAttribute()` ausgelesen werden (Auf diese Weise wertet auch der CustomWebResolver (siehe Abschnitt 51.1.2[1082]) die `genericClasses`-Kategorie aus). Pseudoattribute werden über den gleichen Mechanismus ausgelesen, verhalten sich also aus der Sicht von QF-Test wie normale Attribute. Sie sind jedoch nicht über den HTML-Quelltext definiert oder mit `node.setAttribute()` explizit zugewiesen, sondern werden über den nachfolgend beschriebenen Mechanismus gesetzt.

Bei der Definition eines Pseudoattributes wird angegeben, ob es gecached werden soll. In diesem Fall wird der Wert, den Javascript beim ersten Zugriff auf die Komponente ermittelt hat, bis zum nächsten Komplett-Scan der Seite zwischengespeichert. Das kann die Ausführung insgesamt beschleunigen. Bei veränderbaren Werten, beispielsweise dem Check-Status einer Checkbox, sollte das Pseudoattribut aber nicht gecached werden, da diese Werteänderung sonst nicht für QF-Test ”sichtbar” ist. Bei ”ungecachten” Pseudoattributen findet keine längere Zwischenspeicherung des Wertes statt, sondern dieser wird jedes Mal neu ausgelesen (und dann für die Verarbeitung kurzzeitig gespeichert), wenn ein Event aufgenommen wird, wie zum Beispiel bei einem Mausklick die Einzelevents ”moved”, ”pressed”, ”released” und ”clicked”, oder wenn eine Komponente gesucht wird.

Im nachfolgenden Beispiel wird ein Pseudoattribut für alle HTML-Elemente mit dem Tag ”ICON” oder ”IMAGE” definiert, das mittels JavaScript den Wert von `iconname` ermittelt (Technisch wird hier dazu die beispielhafte Methode `inspect`, die vom Framework für das HTML-Element definiert wurde, aufgerufen).

```
import de.qfs.apps.qftest.client.web.dom.DomNodeAttributes
import de.qfs.apps.qftest.client.web.dom.FunctionalPseudoAttribute
def attr = new FunctionalPseudoAttribute("js_icon",
                      "try {return _qf_node.inspect('iconname')} catch(e){}", true)
DomNodeAttributes.registerPseudoAttributes("ICON", attr)
DomNodeAttributes.registerPseudoAttributes("IMAGE", attr)
```

Beispiel 51.34: Groovy SUT-Skript zum Registrieren eines Pseudoattributs

Dieses Pseudoattribut kann dann zum Beispiel in einem Merkmal-Resolver genutzt werden, wobei im Gegensatz zu einem direkten Aufruf von `node.callJS()` im Resolver automatisch die internen Caching-Mechanismen von QF-Test genutzt werden:

```
def getFeature(node, feature):
    iconname = node.getAttribute("js_icon")
    return iconname
resolvers.addResolver("iconFeature", getFeature, "ICON", "IMAGE")
```

Beispiel 51.35: Nutzung eines Pseudoattributs im Resolver (Jython SUT-Skript)

Im folgenden Skript wird das Pseudoattribut wieder deregistriert.

```
import de.qfs.apps.qftest.client.web.dom.DomNodeAttributes
DomNodeAttributes.unregisterPseudoAttributes("ICON", "js_icon")
DomNodeAttributes.unregisterPseudoAttributes("IMAGE", "js_icon")
```

Beispiel 51.36: Deregistrierung eines Pseudoattributs (Groovy SUT-Skript)

Ein Pseudoattribut wird über einen der folgenden Konstruktoren von `de.qfs.apps.qftest.client.web.dom.FunctionalPseudoAttribute` oder `de.qfs.apps.qftest.client.web.dom.PseudoAttribute` definiert:

`PseudoAttribute FunctionalPseudoAttribute(String name, String javaScriptFunction, Boolean cached)`

Definiert ein Pseudoattribut.

Parameter

`name`	Der Name für das Pseudoattribut.
`javaScriptFunction`	Der JavaScript Code, der beim Zugriff auf das Pseudoattribut in einer Funktion ausgeführt werden soll. `_qf_node` ist dabei das Objekt, worüber auf das HTML-Element zugegriffen werden kann. Die Auswertung erfolgt analog zur `DomNode.callJS`-Methode.
`cached`	`true`, wenn der Wert nach dem ersten Zugriff auf das Pseudoattribut behalten werden soll, andernfalls `false`.
Rückgabewert	Ein Pseudoattribut, das anschließend registriert werden kann.

`PseudoAttribute PseudoAttribute(String name, String javaScriptCode, Boolean cached)`

Definiert ein Pseudoattribut.

Parameter

`name`	Der Name für das Pseudoattribut.
`javaScriptCode`	Der JavaScript Code, der beim Zugriff auf das Pseudoattribut ausgeführt werden soll. `_qf_node` ist dabei das Objekt, worüber auf das HTML-Element zugegriffen werden kann. Die Auswertung erfolgt analog zur `DomNode.evalJS`-Methode.
`cached`	`true`, wenn der Wert nach dem ersten Zugriff auf das Pseudoattribut behalten werden soll, andernfalls `false`.
Rückgabewert	Ein Pseudoattribut, das anschließend registriert werden kann.

Anschließend muss das `Pseudoattribut` über die folgende Methode aus `de.qfs.apps.qftest.client.web.dom.DomNodeAttributes` registriert werden:

`void registerPseudoAttributes(String tag, PseudoAttribute pseudoAttribute)`

Registriert das Pseudoattribut für HTML-Elemente mit dem angegebenen Tag.

Parameter

`tag`	Das Tag der HTML-Elemente, für die das Pseudoattribut registriert werden soll. Wenn das Pseudoattribut für HTML-Elemente mit unterschiedlichen Tags registriert werden soll, muss der Aufruf für die einzelnen Tags durchgeführt werden. Wenn es für alle HTML-Elemente registriert werden soll, lautet das Tag `"<QF_ALL>"`.
`pseudoAttribute`	Das zuvor definierte Pseudoattribut.

`void unregisterPseudoAttributes(String tag, String name)`

Deregistriert das Pseudoattribut für HTML-Elemente mit dem angegebenen Tag. Die Deregistrierung ist nicht zwingend notwendig. Mit der Beendigung von QF-Test geschieht dies automatisch.

Parameter

`tag`	Der Tag-Name der HTML-Elemente, für die das Pseudoattribut deregistriert werden soll.
`name`	Der Name des Pseudoattributs.

51.5 Zugriff auf unsichtbare Felder einer Webseite

Web

Unsichtbare Felder (hidden fields) werden standardmäßig nicht aufgezeichnet und somit auch nicht unterhalb des Fenster und Komponenten(942) Knotens abgelegt.

Wenn Sie regelmäßig auf unsichtbare Felder zugreifen müssen, können Sie die Option Sichtbarkeit von DOM-Elementen berücksichtigen(569) deaktivieren.

Einen andere Möglichkeit bietet das folgende Vorgehen:

- Aktivieren Sie den Modus Komponenten aufnehmen(44).
- Navigieren Sie den Mauszeiger zum Eltern-Element der unsichtbaren Feldes (in den meisten Fällen wohl ein FORM Element).
- Betätigen Sie die rechte Maustaste und wählen Sie Komponente mit Kindern aus dem Kontextmenü.
- Deaktivieren Sie den Modus Komponenten aufnehmen(44).

- Bearbeiten→Vorherigen Knoten anwählen bringt Sie direkt zum aufgenommenen Komponentenknoten. Eine Suche[22] innerhalb von Fenster und Komponenten[942] nach z.B. 'HIDDEN' schnell zur Zielkomponente führen.

Das Auslesen der Attribute des unsichtbaren Feldes (z.B. das Attribut 'value') kann durch ein einfaches SUT-Skript[720] geschehen, wie es unten gezeigt wird. Details zu Skripting im Allgemeinen, die genutzten Methoden und Parameter finden Sie entsprechend in Skripting[186], Die API des Runcontexts[1030] und Pseudo DOM API für Web-Anwendungen[1253].

```
node = rc.getComponent('id of the hidden component', hidden=1)
          node.getAttribute('value')
```

Beispiel 51.37: Auslesen des Attributs 'value' eines unsichtbaren Feldes

51.6 WebDriver mit Safari

Zum Testen mit Safari benötigen Sie macOS und Safari ab Version 12. Die Tests erfordern eine vorherige Einrichtung durch den Benutzer: Wählen Sie in den Safari-Einstellungen unter "Erweitert" den Punkt "Menü Entwickler in der Menüleiste anzeigen". In diesem Menü aktivieren Sie dann den Menüpunkt "Entfernte Automation erlauben". Anschließend müssen Sie noch einmalig ein Terminal-Fenster öffnen und dort den Befehl `/usr/bin/safaridriver -p 0` ausführen, um die anschließende Autorisierungs-Anfrage zu bestätigen.

Hinweis

Aufgrund der speziellen Sicherheitseinstellungen, die Apple seinem SafariDriver auferlegt, gibt es bei Tests mit Safari folgende Einschränkungen:

- Tests können nur abgespielt aber nicht aufgenommen werden
- Harte Events sind nicht möglich
- Es ist nur eine Browser-Instanz zugelassen

Kapitel 52

Steuern und Testen von nativen Windows-Anwendungen - ohne Verwendung der QF-Test `win` Engine

Hinweis

Die `win` Engine von QF-Test ist im Kapitel Testen nativer Windows-Anwendungen(234) beschrieben.

Grundsätzlich sollte für Tests und die Steuerung von nativen Windows-Anwendungen die `win` Engine von QF-Test verwendet werden. Hierfür ist eine passende Lizenz notwendig. Falls diese nicht vorhanden ist und native Windows Anwendungen nur in sehr geringem Umfang testet oder gesteuert werden müssen, kann auf die im Folgenden beschriebene Bibliothek zurückgegriffen werden.

Sie erlaubt über die Microsoft UI Automation Schnittstelle den Zugriff auf Elemente. Aktionen können angestoßen und ausgewählte Werte überprüft werden. Informationen zu der Microsoft Schnittstelle finden Sie auf https://en.wikipedia.org/wiki/Microsoft_UI_Automation

Das Ansprechen der Bibliothek erfolgt über Prozeduren, die Teil der Standardbibliothek(183) qfs.qft sind.

Die direkte Aufnahme von Aktionen oder Checks (Capturing) ist nicht möglich. Der bzw. die Parameter zur Identifikation des GUI Objekts müssen bestimmt und passend an die entsprechende Prozedur übergeben werden.

Das Abspielen der Aktionen passiert primär über ”harte” Systemevents. Dies führt zu einem anderen Wiedergabeverhalten als man es bei Java oder Web-Anwendungen mit QF-Test sonst gewohnt ist.

Trotz dieser Einschränkungen kann die Bibliothek ein hilfreiches Werkzeug für einfache Test- oder Steuerungsaufgaben von nativen Windows-Anwendungen sein.

52.1 Vorgehensweise

Die Microsoft UI Automation ist ein Accessibility- und Test-Framework, das den programmatischen Zugriff auf GUI-Elemente von nativen Windows-Anwendungen erlaubt. In QF-Test kann dieses Framework in Skriptknoten über das Jython-Modul `uiauto` angesprochen werden (alternativ `de.qfs.UIAuto` für Groovy, `uiauto` für Javascript).

Zur direkten und einfachen Nutzung für die Testerstellung bietet QF-Test in der Standardbibliothek ein Package, das QF-Test Prozeduren für häufig benötigte Interaktionen mit GUI-Elementen zur Verfügung stellt. Dieses wird im Folgenden beschrieben.

Die für die Ansteuerung von nativen Windows Elementen relevanten Prozeduren befinden sich im Package `qfs.autowin`. Sie sehen in dem Package etliche Prozeduren, die als überholt (deprecated) gekennzeichnet sind. Diese sind durch normale Knoten der `win` Engine von QF-Test abgelöst worden und werden nicht mehr weiterentwickelt. Dies bezieht sich zum Beispiel auf Themen bei der Bildschirmskalierung. Prinzipiell können Sie die Prozeduren so wie sie sind weiter verwenden. Wenn es aber zu Problemen kommt, dann ist es empfehlenswert auf die `win` Engine umzusteigen.

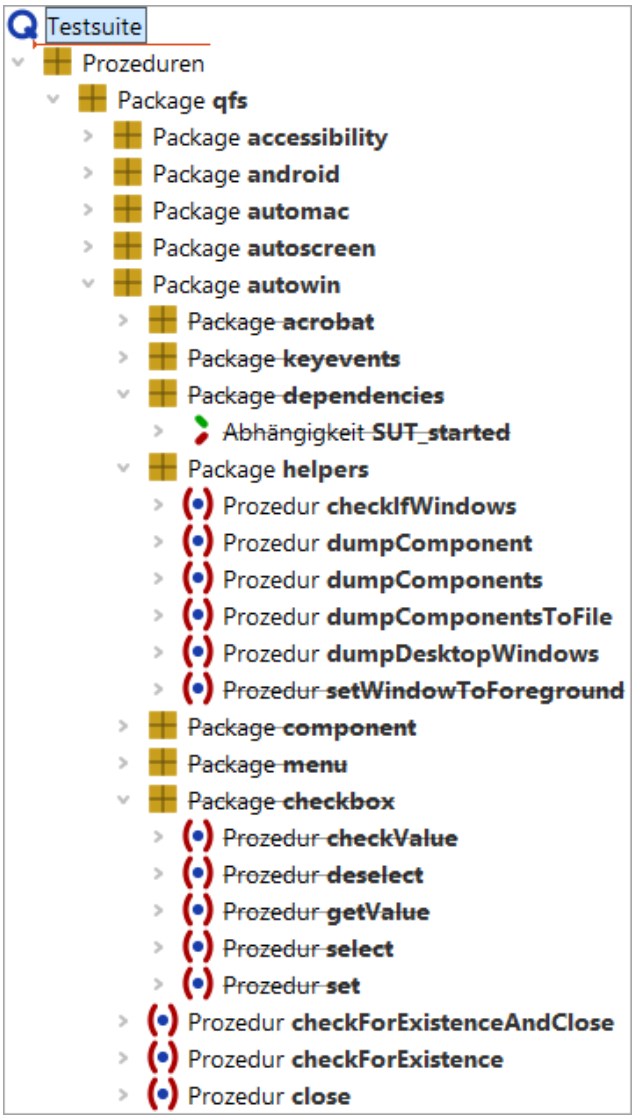


Abbildung 52.1: UI Automation Prozeduren in der Standardbibliothek

Die grundsätzliche Vorgehensweise bei der Erstellung von Tests für native Windows-Anwendungen ist folgende:

- Start der Anwendung
- Identifikation der Zugriffsparameter für die GUI-Elemente
- Erstellung von Tests unter Verwendung der ermittelten Zugriffsparameter

52.1.1 Start der Anwendung

Die zu testende Anwendung kann, muss aber nicht, über QF-Test gestartet werden.

Wenn die Anwendung über QF-Test gestartet wird, erscheint der gestartete Prozess in der Liste der QF-Test Clients im Menü Clients und auch die Beendigung des Prozesses durch QF-Test ist möglich.

Zur Prüfung, ob die zu testende Anwendung gestartet ist, wird die Prozedur `qfs.autowin.checkForExistence` verwendet.

Ein Beispiel zum Anwendungsstart finden Sie in Abschnitt 52.2.1(1143).

52.1.2 Auflistung aller GUI-Elemente eines Fensters

Bevor Tests erstellt werden können, ist es notwendig, sich einen Überblick über die in der Anwendung vorhandenen GUI-Elemente zu verschaffen. Hierzu stehen die Prozeduren `qfs.autowin.helpers.dumpComponents` und `qfs.autowin.helpers.dumpComponentsToFile` zur Verfügung. Diese listen die GUI-Elemente für das angegebene Fenster auf. Erstere gibt das Ergebnis im QF-Test Terminal, letztere in eine Datei aus.

Mit der Prozedur `qfs.autowin.helpers.dumpDesktopWindows` können die Titel der aktuell geöffneten Windows Fenster ausgegeben werden.

`qfs.autowin.helpers.dumpComponents` gibt den Namen (Name), die Klasse (ClassName), den Komponententyp (ControlType) und die Id (AutomationId) des GUI-Elements aus, soweit diese für das GUI-Element implementiert wurden.

Alle auf dem Desktop sichtbaren GUI-Elemente sind in einer Baumstruktur organisiert, mit dem Desktop als Wurzelelement. Im Aufruf der Dump-Prozedur wird das gewünschte Fenster angegeben. Die Verschachtelung der GUI-Elemente wird in der Ausgabe mittels Einrückungen dargestellt.

Hinweis Die Ausgabe der Prozedur dumpComponents() wird im QF-Test Terminal, das standardmäßig unten im QF-Test Fenster angezeigt wird, ausgegeben. Im Client-Terminal und den Skript-Konsolen, die separat geöffnet werden können, wird das Ergebnis dieses Dump-Befehls nicht angezeigt.

Ein Beispiel finden Sie in Abschnitt 52.2.2(1144).

52.1.3 Informationen zu einzelnen GUI-Elementen

In der Standardbibliothek steht die Prozedur `qfs.autowin.helpers.dumpComponent` zur Verfügung, um sich zu einem

einzelnen GUI-Element weitere Informationen ausgeben zu lassen. Dies umfasst die Liste der zur Verfügung stehenden Methoden sowie einzelne Attributwerte.

52.1.4 Zugriff auf GUI Elemente

Alle in der Standardbibliothek für die Ausführung von Aktionen auf nativen Windows-Anwendungen zur Verfügung gestellten Prozeduren ermitteln im ersten Schritt das jeweilige GUI-Element und führen dann die entsprechende Aktion darauf aus. Diese sind im Package `qfs.autowin.component` zusammengefasst. Das GUI-Element wird über die Prozedur `qfs.autowin.component.getControl` ermittelt. Die Parameter dieser Hilfsprozedur gelten daher für alle Prozeduren, die eine Aktion auf einem GUI-Element ausführen.

Folgende Parameter(-kombinationen) sind möglich (in der Reihenfolge der Auswertung):

- AutomationId
- ControlType und Name
- ControlType und Index
- ClassName und Name
- ClassName und Index
- Name

AutomationId
Über die AutomationId ist die eindeutige Identifikation eines GUI-Elements innerhalb eines Fensters möglich. Sie muss jedoch bei der Anwendungsentwicklung explizit vergeben werden und ist deshalb in vielen Fällen leider nicht verfügbar.

Name
Der Name entspricht in der Regel dem angezeigten Text. Namen müssen nicht eindeutig sein. Daher kann es notwendig sein, zusätzlich zum Namen den ControlType oder ClassName anzugeben. Namen werden als reguläre Ausdrücke ausgewertet. Nähere Infomationen zu regulären Ausdrücken finden Sie in Reguläre Ausdrücke - *Regexps*[1023].

ControlType
Der ControlType ist ein Wert aus einer vordefinierten Liste von Komponententypen, zum Beispiel : Button, CheckBox, ComboBox, DataGrid, Edit, List, Tab, Text. Die Prozedur `qfs.autowin.helpers.dumpComponent`

zeigt den Namen und den numerischen Wert des jeweiligen ControlTypes an. Um ein GUI-Element an Hand seines ControlTypes zu identifizieren, muss entweder der Name oder der Index, bezogen auf den ControlType, zusätzlich angegeben werden, außer es gibt im Fenster nur ein GUI-Element dieses ControlTypes.

ClassName
Der ClassName eines GUI-Elements ist Framework-spezifisch. Zusätzlich zu dem ClassName kann auch der Name oder der Index, bezogen auf die Klasse, des GUI-Elements angegeben werden.

52.1.5 Ausführen von Aktionen auf GUI-Elementen

Für die häufigsten Aktionen finden Sie vorgefertigte Prozeduren in der Standard-Bibliothek qfs.qft im Package `qfs.autowin.component`. Sie können das Package nach Belieben unter Verwendung des Jython-Moduls ”uiauto” erweitern. Hierbei empfiehlt es sich, die Erweiterungen nicht direkt in der qfs.qft vorzunehmen, da wir die Standardbibliothek kontinuierlich weiterentwickeln und mit jeder QF-Test Version neu ausliefern.

Mausklick
Prozedur: `qfs.autowin.component.click`

Zunächst versucht die Prozedur ein Klick-Event auszulösen. Dies ist jedoch häufig nicht implementiert. Falls dies nicht möglich ist, wird ein harter Mausklick auf die Position des GUI-Elements abgespielt.

Warten auf Komponente
Prozedur: `qfs.autowin.component.waitForComponent`

Die Prozedur wartet auf die angegebene Komponente und sobald diese gefunden wurde, wird die Kontrolle an den rufenden Knoten zurückgegeben. Die angegebene Wartezeit (in Millisekunden) ist die maximale Zeit, die gewartet wird. Die Prozedur wirft eine Exception, wenn die Komponente nicht innerhalb der angegeben Zeit gefunden wird.

Warten auf Fenster
Prozedur: `qfs.autowin.checkForExistence`

Die Prozedur wartet auf das angegebene Fenster und sobald dieses gefunden wurde, wird die Kontrolle an den rufenden Knoten zurückgegeben. Die angegebene Wartezeit (in Millisekunden) ist die maximale Zeit, die gewartet wird. Die Prozedur wirft eine Exception, wenn das Fenster nicht innerhalb der angegeben Zeit gefunden wird.

Texteingabe
Prozedur: `qfs.autowin.component.setText`

Häufig ist setText() für die jeweilige Komponente nicht implementiert. In diesem Fall steht meistens setValue() zur Verfügung.

Tastatur-Events
Im Package `qfs.autowin.keyevents` stehen Prozeduren für das Abspielen der Tastatur-Events ENTER, TAB und DELETE zur Verfügung. Über die Prozedur `qfs.autowin.keyevents.sendKey` können einzelne Tastatureingaben, auch in Kombination mit Strg, Alt und der Umschalttaste, erfolgen, z.B. einzelne Buchstaben, Ziffern, Funktionstasten. Die Eingabe geht auf die Komponente, die im angegebenen Fenster den Fokus hat.

Text auslesen
Prozedur: `qfs.autowin.component.getText`

Einen direkten Zugriff auf den Text eines GUI-Elements gibt es nicht. Viele GUI-Elemente haben einen Namen, der in der Regel dem angezeigten Text entspricht. Manchmal hat der Name nur einen Bezug zum angezeigten Text, entspricht ihm aber nicht wörtlich. Manche Elemente haben unabhängig vom Namen einen Wert, der den Text widergibt. Die Prozedur getText() versucht zunächst, den Wert des Elements zu ermitteln und, falls dies fehlschlägt oder der Wert leer ist, den Namen des Elements.

Die Prozeduren getName() und getValue() stehen zusätzlich zur Verfügung.

Geometrie auslesen
Prozedur: `qfs.autowin.component.getGeometry`

Text prüfen
Prozedur: `qfs.autowin.component.checkText`

Die Prozedur holt sich den Text des GUI-Elements über die Prozedur getText() und prüft den erhaltenen Wert mit dem übergebenen Sollwert.

Die Prozeduren checkName() und checkValue() stehen zusätzlich zur Verfügung.

Geometrie prüfen
Prozedur: `qfs.autowin.component.getGeometry`

Die ermittelten Geometriedaten werden mit den übergebenen Sollwerten verglichen.

Bildvergleich

Prozedur: `qfs.autowin.component.checkImage`

Für das Prüfen des Abbilds wird ein Referenzbild benötigt. Dieses muss in einer Datei im png-Format bereitgestellt werden. Über `qfs.autowin.component.getGeometry` werden die Bildschirmkoordinaten des zu prüfenden Bereichs ermittelt. Die eigentliche Prüfung wird über die Prozedur getPositionOfImage() aus dem Package `qfs.autoscreen` der Standardbibliothek durchgeführt.

Menüeintrag auswählen

Procedure: `qfs.autowin.menu.selectItem`

Beim Debuggen eines Tests über Einzelschritte ist es häufig sehr störend, wenn vor dem Klick auf einen Eintrag das zugehörige Menü zuklappt, weil der Fokus an QF-Test übergeben wurde. Hier hilft die Prozedur selectItem(), mit der das Öffnen des Menüs und der Klick auf den Eintrag in einem Debug-Schritt ausgeführt werden kann.

52.2 Beispiel

Die Beispieltestsuiten finden Sie in den Unterverzeichnissen `demo/carconfigWpf` und `demo/carconfigForms` des QF-Test Installationsverzeichnisses.

52.2.1 Start der Anwendung

Die Beispieltestsuiten verwenden für den Start der Anwendung die in der Standardbibliothek mitgelieferte Abhängigkeit `SUT_started` (SUT = System Under Test) aus dem Package `qfs.autowin.dependencies`.

Der Start der Demoapplikation wird über den Knoten

- Shell-Kommando ausführen

angestoßen. Anschließend muss auf die Applikation gewartet werden, bevor die Tests abgespielt werden können:

- Prozeduraufruf: `qfs.autowin.checkForExistence`

Die Demo-Testsuite verwendet die Funktionalität des Abhängigkeit Knotens, womit sich die Testvor- und -nachbedingungen sehr effizient gestalten lassen. Ganz kurz zusammengefasst besteht die Funktionalität eines Abhängigkeit Knotens darin, dass er den darin enthaltenen Vorbereitung Knoten vor dem Start des Testfalls ausführt, damit die implementierte Vorbedingung hergestellt wird. Nach Ausführung des Testfalls wird der Aufräumen Knoten der Abhängigkeit standardmäßig nicht ausgeführt. Dies geschieht erst bei Bedarf bei der Ausführung des Abhängigkeit Knotens des nachfolgenden Testfalls. Der Bedarf ergibt sich dann, wenn der nachfolgende Testfall eine andere Abhängigkeit ruft oder dieselbe Abhängigkeit mit anderen charakteristischen Variablen. Weitere Informationen zu Abhängigkeiten finden Sie Tutorial sowie im Handbuch im Kapitel Abhängigkeit Knoten(161).

Weitere Informationen hierzu finden Sie in Abschnitt 52.1.1(1139).

52.2.2 Übersicht über die GUI-Elemente der Anwendung

Wenn das SUT läuft, ist der nächste Schritt, dass man sich dessen GUI-Elemente anzeigen lässt. Im vorliegenden Fall wurde die Prozedur `qfs.autowin.helpers.dumpComponents` verwendet. Das Ergebnis wird in das QF-Test Terminal geschrieben.

Wir wollen uns als nächstes für einige GUI-Elemente der WPF Demo-Anwendung die Informationen, die zur Identifikation zur Verfügung stehen, ansehen.

Abbildung 52.2: Die WPF Demo-Applikation

Fenster (1)

Name: CarConfiguratorNet WPF
ClassName: Window
ControlType: Window (#50032)
Das Fenster kann über seinen Namen angesprochen werden, da kein weiteres GUI-Element diesen Namen verwendet.

Menü (2)

Name: Hilfe
ClassName: MenuItem
ControlType: MenuItem (#50011)
AutomationId: mHelp

Das GUI-Element kann über die eindeutige AutomationId oder seinen Namen angesprochen werden. Noch eine Alternative ist der ControlType oder ClassName ”MenuItem” zusammen mit dem Namen oder dem Index ”4”.

Tabellenzelle (3)

Name: Rolo
ClassName: DataGridCell
ControlType: Custom (#50025)

Die Tabellenzelle kann über den Namen oder den ControlType zusammen mit dem Namen identifiziert werden. Wenn die Tabellenzelle über einen Index ange-

sprochen werden soll, ist es besser, den ClassName anstelle des ControlType zu verwenden, weil der ControlType ”Custom” auch für andere GUI-Elemente zum Einsatz kommt. Da bei der Entwicklung keine AutomationId vergeben wurde, steht diese nicht zur Verfügung.

Eingabefeld (4)

ClassName: TextBox
ControlType: Edit (#50004)
AutomationId: textBoxDiscountPrice

Das Eingabefeld kann über die AutomationId oder den ControlType zusammen mit dem Index 0 angesprochen werden.

Anzeigefeld (5)

Name: 12.300,00 €
ClassName: Text
ControlType: Text (#50020)
AutomationId: labelCalculatedPriceOutput

Neben der AutomationId und dem ControlType steht hier zwar auch ein Name zur Verfügung. Dieser eignet sich jedoch nicht zur Identifikation des GUI-Elements, da dieser je nach angezeigtem Text variiert.

Kapitel 53

Steuern und Testen von nativen MacOS-Anwendungen

Aktuell unterstützt QF-Test primär das funktionale Testen von Java und Web-Anwendungen. Die gleichwertige Erweiterung für native MacOS-Anwendungen mit äquivalenten Abläufen und Features ist in der Entwicklung. Als Übergangslösung für Situationen, in denen neben Java und Web-Anwendungen auch in geringerem Maß auf native MacOS-Anwendungen steuernd zugegriffen werden soll, wurde die im Folgenden beschriebene Bibliothek integriert.

Sie erlaubt über die Accessibility Schnittstelle den Zugriff auf Elemente. Aktionen können angestoßen und ausgewählte Werte überprüft werden. Weiterführende Informationen zur Accessibility Schnittstelle finden Sie in der Dokumentation von Apple.

Das Ansprechen der Bibliothek erfolgt über Prozeduren, die Teil der Standardbibliothek(183) qfs.qft sind.

Die direkte Aufnahme von Aktionen oder Checks (Capturing) ist nicht möglich. Der bzw. die Parameter zur Identifikation des GUI Objekts müssen bestimmt und passend an die entsprechende Prozedur übergeben werden.

Das Abspielen der Aktionen passiert primär über ”harte” Systemevents. Dies führt zu einem anderen Wiedergabeverhalten als man es bei Java oder Web-Anwendungen mit QF-Test sonst gewohnt ist.

Trotz dieser Einschränkungen kann die Bibliothek ein hilfreiches Werkzeug für einfache Test- oder Steuerungsaufgaben von nativen MacOS-Anwendungen sein.

53.1 Vorgehensweise

Die Accessibility-Schnittstelle erlaubt den programmatischen Zugriff auf GUI-Elemente von nativen MacOS-Anwendungen. In QF-Test kann dieses Framework in Skriptknoten über das Jython-Modul `automac` angesprochen werden (alternativ `de.qfs.automac` für Groovy, `automac` für Javascript).

Zur direkten und einfachen Nutzung für die Testerstellung bietet QF-Test in der Standardbibliothek ein Package, das QF-Test Prozeduren für häufig benötigte Interaktionen mit GUI-Elementen zur Verfügung stellt. Dieses wird im Folgenden beschrieben.

Die für die Ansteuerung von nativen MacOS-Elementen relevanten Prozeduren befinden sich im Package `qfs.automac`.

Die grundsätzliche Vorgehensweise bei der Erstellung von Tests für native MacOS-Anwendungen ist folgende:

- Start der Anwendung
- Identifikation der Zugriffsparameter für die GUI-Elemente
- Erstellung von Tests unter Verwendung der ermittelten Zugriffsparameter

53.1.1 Start der Anwendung

Die zu testende Anwendung kann, muss aber nicht, über QF-Test gestartet werden. Auf jeden Fall muss eine ”Verbindung” zur Anwendung hergestellt werden. Dies geschieht über die Prozedur `qfs.automac.app.connect`, wobei der Zugriff auf die Accessibilty-Schnittstelle der Anwendung von QF-Test in einer Variablen abgespeichert werden. Diese Jython Variable wird in den Prozeduren, die Aktionen auf der Anwendung ausführen, genutzt. Für das Auffinden der Anwendung oder ggf. deren Start stehen mehrere Möglichkeiten zur Verfügung.

Über den Parameter `bundleId` wird die für die Anwendung eindeutige Bundle Id angegeben, z.B. com.apple.Calculator. Falls die Anwendung bereits gestartet ist, wird ein Zugriff auf diese Anwendung abgespeichert, ansonsten wird die Anwendung gestartet.

Über den Parameter `bundleFile` wird für die Anwendung das Bundle file angegeben unter dem die Anwendung zu finden ist, z.B. /Applications/Calculator.app. Falls die Anwendung bereits gestartet ist, wird ein Zugriff auf diese Anwendung abgespeichert, ansonsten wird die Anwendung gestartet. In letzterem Fall wird auch ein QF-Test Prozess gestartet, der in der Liste der QF-Test Clients im Menü Clients erscheint. Durch Beendigung dieses Prozesses wird auch die Applikation selbst beendet.

Über den Parameter `title` wird der Zugriff auf die Anwendung mit dem angegebenen Fenstertitel abgespeichert. In diesem Fall muss die Anwendung bereits anderweitig

gestartet worden sein, z.B. über den Knoten Shell-Kommando ausführen.

Über den Parameter `processId` wird der Zugriff auf die Anwendung mit der angegebenen Prozessidentifikationsnummer (PID) abgespeichert. In diesem Fall muss die Anwendung bereits anderweitig gestartet worden sein, z.B. über den Knoten Shell-Kommando ausführen.

Die Prozedur `qfs.automac.helpers.DumpDesktopWindows` gibt den Titel, die PID, Bundle id und Bundle file aller gestarteten Applikationen im Terminal aus.

53.1.2 Auflistung aller GUI-Elemente eines Fensters

Bevor Tests erstellt werden können, ist es notwendig, sich einen Überblick über die in der Anwendung vorhandenen GUI-Elemente zu verschaffen. Hierzu stehen die Prozeduren `qfs.automac.helpers.dumpComponents` und `qfs.automac.helpers.dumpComponentsToFile` zur Verfügung. Diese listen die GUI-Elemente für das angegebene Fenster auf. Erstere gibt das Ergebnis im QF-Test Terminal, letztere in eine Datei aus.

`qfs.automac.helpers.dumpComponents` gibt Label, Titel, Rolle, Subrole, Typ und Identifier des GUI-Elements aus, soweit diese für das GUI-Element implementiert wurden.

Alle auf dem Desktop sichtbaren GUI-Elemente sind in einer Baumstruktur organisiert, mit dem Desktop als Wurzelelement. Die Verschachtelung der GUI-Elemente wird in der Ausgabe mittels Einrückungen dargestellt. Die Dump-Prozedur bezieht sich auf die verbundene Applikation oder auf die im Aufruf der Prozedur spezifizierte.

Hinweis Die Ausgabe der Prozedur dumpComponents() wird im QF-Test Terminal, das standardmäßig unten im QF-Test Fenster angezeigt wird, ausgegeben. Inm Client-Terminal und den Skript-Konsolen, die separat geöffnet werden können, wird das Ergebnis dieses Dump-Befehls nicht angezeigt.

53.1.3 Informationen zu einzelnen GUI-Elementen

In der Standardbibliothek steht die Prozedur `qfs.automac.helpers.dumpComponent` zur Verfügung, um sich zu einem einzelnen GUI-Element weitere Informationen ausgeben zu lassen. Dies umfasst die Liste der zur Verfügung stehenden Methoden sowie einzelne Attributwerte.

Über den Accessibility Inspector von XCode können ebenfalls die Parameter eines GUI-Elements angezeigt werden.

53.1.4 Zugriff auf GUI Elemente

Alle in der Standardbibliothek für die Ausführung von Aktionen auf nativen MacOS-Anwendungen zur Verfügung gestellten Prozeduren ermitteln im ersten Schritt das jeweilige GUI-Element und führen dann die entsprechende Aktion darauf aus. Diese sind im Package `qfs.automac.component` zusammengefasst. Das GUI-Element wird über die Prozedur `qfs.automac.component.getControl` ermittelt. Die Parameter dieser Hilfsprozedur gelten daher für alle Prozeduren, die eine Aktion auf einem GUI-Element ausführen.

Folgende Parameter sind möglich:

- label
- title
- identifier
- role
- roleType
- subrole
- index

Bei der Angabe mehrerer Parameter wird das GUI-Element gesucht, auf das alle Werte zutreffen.

`identifier`
: Über `identifier` ist die eindeutige Identifikation eines GUI-Elements innerhalb eines Fensters möglich. Er muss jedoch bei der Anwendungsentwicklung explizit vergeben werden und ist deshalb nicht immer verfügbar.

`label`
: `label` entspricht in der Regel dem angezeigten Text. `label` muss nicht eindeutig sein. Daher kann es notwendig sein, zusätzlich andere Parameter, z.B. `role` anzugeben. Falls die GUI-Elemente mit dem Accessibility Inspector untersucht werden, heißt dieses Attribut dort entweder `Label` oder `AXDescription`.

`title`
: `title` entspricht in der Regel dem angezeigten Text. `title` muss nicht eindeutig sein. Daher kann es notwendig sein, zusätzlich andere Parameter, z.B. `role` anzugeben.

role
`role` ist ein Wert aus einer vordefinierten Liste von Komponententypen, z.B. AXButton.

roleType
`roleType` bezeichnet den Typ des GUI-Elements. Falls die GUI-Elemente mit dem Accessibility Inspector untersucht werden, heißt dieses Attribut dort entweder `Type` oder `AXRoleDescription`.

subrole
Eine zusätzliche Spezifizierung zu `role`.

index
Falls es mehrere GUI-Elemente gibt, auf die die angegebenen Parameter zutreffen, kann über `index` das gewünschte GUI-Element spezifiziert werden. `index` beginnt mit 0.

53.1.5 Ausführen von Aktionen auf GUI-Elementen

Für die häufigsten Aktionen finden Sie vorgefertigte Prozeduren in der Standard-Bibliothek qfs.qft im Package `qfs.automac.component`. Sie können das Package nach Belieben unter Verwendung des Skripting-Moduls ”automac” erweitern. Hierbei empfiehlt es sich, die Erweiterungen nicht direkt in der qfs.qft vorzunehmen, da wir die Standardbibliothek kontinuierlich weiterentwickeln und mit jeder QF-Test Version neu ausliefern.

Mausklick
Prozedur: `qfs.automac.component.click`

Die Prozedur wartet auf die angegebene Komponente und sobald diese gefunden wurde, wird ein Klick-Event abgespielt.

Warten auf Komponente
Prozedur: `qfs.automac.component.waitForComponent`

Die Prozedur wartet auf die angegebene Komponente und sobald diese gefunden wurde, wird die Kontrolle an den rufenden Knoten zurückgegeben. Die angegebene Wartezeit (in Millisekunden) ist die maximale Zeit, die gewartet wird. Die Prozedur wirft eine Exception, wenn die Komponente nicht innerhalb der angegeben Zeit gefunden wird.

Texteingabe
Prozedur: `qfs.automac.component.setValue`

Die Prozedur wartet auf die angegebene Komponente und sobald diese gefunden wurde, wird über die Methode setValue() der übergebene Wert eingetragen.

Tastatur-Events
Prozedur: `qfs.automac.sendKey`

Es können einzelne Tastatureingaben, auch in Kombination mit STRG, ALT, CMD und SHIFT, abgespielt werden, z.B. einzelne Buchstaben, Ziffern, Funktionstasten. Die Eingabe geht auf die Komponente, die in der Applikation aktuell den Fokus hat.

Text auslesen
Prozedur: `qfs.automac.component.getValue`

Die Prozedur wartet auf die angegebene Komponente und sobald diese gefunden wurde, wird über die Methode getValue() deren Wert ermittelt und zurückgeliefert.

Geometrie auslesen
Prozedur: `qfs.automac.component.getGeometry`

Es werden die x- und y-Bildschirmkoordinate der oberen linken Ecke sowie Breite und Höhe des angegebenen GUI-Elements ermittelt und als komma-separierter Text zurückgegeben.

Text prüfen
Prozedur: `qfs.automac.component.checkValue`

Die Prozedur holt sich den Text des GUI-Elements über die Prozedur getValue() und prüft den erhaltenen Wert mit dem übergebenen Sollwert.

Geometrie prüfen
Prozedur: `qfs.automac.component.getGeometry`

Die ermittelten Geometriedaten werden mit den übergebenen Sollwerten verglichen.

Bildvergleich
Prozedur: `qfs.automac.component.checkImage`

Für das Prüfen des Abbilds wird ein Referenzbild benötigt. Dieses muss in einer Datei im png-Format bereitgestellt werden. Über `qfs.automac.component.getGeometry` werden die Bildschirmkoordinaten

des zu prüfenden Bereichs ermittelt. Die eigentliche Prüfung wird über die Prozedur getPositionOfImage() aus dem Package `qfs.autoscreen` der Standardbibliothek durchgeführt.

Menüeintrag auswählen

Procedure: `qfs.automac.menu.selectItem`

Beim Debuggen eines Tests über Einzelschritte ist es häufig sehr störend, wenn vor dem Klick auf einen Eintrag das zugehörige Menü zuklappt, weil der Fokus an QF-Test übergeben wurde. Hier hilft die Prozedur selectItem(), mit der das Öffnen des Menüs und der Klick auf den Eintrag in einem Debug-Schritt ausgeführt werden kann.

Kapitel 54

APIs für Erweiterungen

QF-Test bietet einige APIs an, über die Sie seine Funktionalität erweitern können. Die Interfaces können bequem in Jython oder Java implementiert werden. In letzterem Fall sollten Sie die kompilierten Klassen in eine jar Datei packen und diese in das Plugin Verzeichnis stellen (siehe Abschnitt 50.2(1029)), so dass die Verbindung mittels Jython oder Groovy hergestellt werden kann.

54.1 Das `resolvers` Modul

Mit diesem Erweiterungs-API können Sie Einfluss darauf nehmen, wie QF-Test Komponenten und Unterelemente erkennt und aufzeichnet. Dies ist ein sehr mächtiges Feature, mit dem Sie volle Kontrolle über das Komponenten-Management von QF-Test erhalten.

Video

Video zum Thema:

'Resolver in QF-Test'
https://www.qftest.com/de/yt/resolver-46.html

54.1.1 Verwendung

Hinweis

Beim Registrieren eines Resolvers ist es wichtig, das GUI-Engine(722) Attribut für den SUT-Skript(720) Knoten korrekt anzugeben. Wird die falsche Engine gesetzt, funktioniert der Resolver einfach nicht. Ist gar keine Engine angegeben, wirkt der Resolver bei allen Engines und kann zu Verwirrung führen und die Wiedergabe in einer Engine stören, für die er nicht gedacht war.

Zum besseren Verständnis zur Verwendung von Resolvern hier eine kurze Beschreibung der Komponentenerkennung durch QF-Test. Sie läuft grob in vier Schritten ab:

1. Einlesen der Komponentenobjekte aus dem GUI.
2. Datenextraktion für die Einzelkomponente: z.B. Komponentenklasse, Id, Koordinaten, Komponententext.
3. Beziehungsanalyse der Komponenten zueinander: z.B. Strukturinformationen (Index), Bestimmung einer der Komponente zugehörigen Beschriftung `qfs:label*`-Varianten(74).
4. Aufnahme: Generierung eines Komponente Knotens und Speicherung der erhaltenen Daten in den Details des Knotens.
   Wiedergabe: Abgleich der erhaltenen Daten mit den Details des Knotens, auf den die Aktion abgespielt werden soll.

QF-Test verwendet für die in Schritt 2 und 3 zu erledigenden Aufgaben Resolver. Über die API können Methoden dieser Resolver erweitert und so Einfluss auf die Komponentenerkennung genommen werden.

Sollen Werte bei der Aufnahme beeinflusst werden, so ist dies nur über einen Resolver möglich. Bei der Wiedergabe können Komponenteninformationen auch über andere Wege (z.B. ein Skript oder einen regulären Ausdruck (vgl. Abschnitt 49.3(1023)) in den Details des Komponente Knotens) erlangt werden.

Web

Für Web-Anwendungen steht eine spezielle Schnittstelle zur Verfügung, in der die Funktionalität der hier beschriebenen Resolver zusammengefasst und leichter konfigurierbar ist. Siehe Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077). Der dort beschriebene CustomWebResolver installieren Knoten ist für Web-Elemente optimiert und somit wesentlich performanter als der Einsatz der Resolver dieses Abschnitts. Nur in Spezialfällen ist für Web-Komponenten der Einsatz der hier beschriebenen Resolver sinnvoll.

Es folgt eine Aufstellung zur Verfügung stehender `resolver`.

- `NameResolver` Abschnitt 54.1.7(1163)
- `GenericClassNameResolver` Abschnitt 54.1.8(1165)
- `ClassNameResolver` Abschnitt 54.1.9(1166)
- `FeatureResolver` Abschnitt 54.1.10(1167)
- `ExtraFeatureResolver` Abschnitt 54.1.11(1168)
- `ItemNameResolver` Abschnitt 54.1.12(1174)
- `ItemValueResolver` Abschnitt 54.1.13(1175)
- `TreeTableResolver` Abschnitt 54.1.14(1176)

- InterestingParentResolver Abschnitt 54.1.15(1178)
- TooltipResolver Abschnitt 54.1.16(1179)
- IdResolver Abschnitt 54.1.17(1180)
- EnabledResolver Abschnitt 54.1.18(1181)
- VisibilityResolver Abschnitt 54.1.19(1182)
- MainTextResolver Abschnitt 54.1.20(1183)
- WholeTextResolver Abschnitt 54.1.21(1184)
- BusyPaneResolver Abschnitt 54.1.22(1184)
- GlassPaneResolver Abschnitt 54.1.23(1185)
- TreeIndentationResolver Abschnitt 54.1.24(1186)
- EventSynchronizer Abschnitt 54.1.25(1187)
- BusyApplicationDetector Abschnitt 54.1.26(1187)
- ExtraFeatureMatcher Abschnitt 54.1.27(1188)

54.1.2 Implementierung

Bei der Implementierung eines Resolvers sind folgende beide Schritte nötig:

1. Implementierung des Resolver-Interfaces.
2. Registrierung des Interfaces unter Angabe eines Namens und der Komponentenklasse(n), für die es gilt.

In den meisten Fällen besteht das Interface aus einer einzigen Methode, sodass ein typisches Beispiel wie folgt aussieht (Jython-Skript):

```
def getName(menuItem, name):
    if not name:
        return menuItem.getLabel()
resolvers.addResolver("menuItems", getName, "MenuItem")
```

Beispiel 54.1: Einfacher Jython NameResolver für MenuItems

Die ersten drei Zeilen definieren die Methode des Resolver Interfaces. Über den Namen der Methode leitet sich ab, um welchen Resolvertyp es sich handelt, d.h. welcher Wert eines Komponente Knotens beeinflusst wird. In unserem Fall heißt die Methode `getName`. Es handelt sich also um einen `NameResolver`. Die vierte Zeile ruft die `addResolver` Funktion im `resolvers` Modul und registriert den Resolver.

Die meisten Resolver-Methoden haben nur zwei Parameter: Erstens die Komponete, für die die Komponentenerkennung in diesem Moment ausgeführt wird. Zweitens das in dieser Methode behandelte Feld bzw. Objekt. Bei einem `NameResolver` ist dies der vom QF-Test Standard-`NameResolver` ermittelte Name. Bei einem `FeatureResolver` das ermittelte Merkmal usw. Eine ausführliche Beschreibung der einzelnen Resolver Interfaces finden Sie in den Kapiteln Abschnitt 54.1.7(1163) bis Abschnitt 54.1.26(1187).

Der Name, unter dem ein Resolver registriert wird, muss eindeutig sein. Er wird benötigt, wenn der Resolver verändert wurde und die alte Version durch die neue ersetzt werden soll oder wenn der Resolver explizit mittels `resolvers.removeResolver("resolvername")` (vgl. removeResolver(1161)) entfernt werden soll. Die Namen aller registrierten Resolver können mit Hilfe der Funktion `resolvers.listNames()` abgerufen werden (vgl. listNames(1162)).

Nach der Änderung eines Resolver-Skripts muss dieses erneut registriert werden, um die Änderung zu aktivieren. Eine vorherige Deregistrierung des alten Standes ist nicht notwendig, solange der Name, unter dem der Resolver registriert wurde, unverändert bleibt.

Alle Arten von Resolvern können wahlweise für individuelle Komponenten, für spezifische Klassen oder Generische Klassen(62) registriert werden. Resolver für individuelle Komponenten werden nur aufgerufen, wenn ihre Information für genau diese Komponente benötigt wird. Resolver, die für eine Klasse registriert sind, werden für alle Objekte dieser oder einer davon abgeleiteten Klasse aufgerufen.

Ein Resolver kann für eine oder mehrere individuelle Komponenten und/oder Klassen registriert werden. Falls der entsprechende Parameter nicht spezifiziert wird, gilt der Resolver für alle Klassen. Dies ist z.B. möglich für `NameResolver`, `FeatureResolver` und `TreeTableResolver`. Sie werden dann für jeden zu ermittelnden Namen bzw. Merkmal oder TreeTable aufgerufen. Dies ist äquivalent zu - aber effektiver als - eine Registrierung für die Klasse `java.lang.Object` bei Java-Applikationen.

Es können verschiedene Resolver mit unterschiedlichen Aufgaben erstellt und zur Laufzeit registriert werden. Um den Resolver permanent zu installieren, verschieben Sie den SUT-Skript Knoten direkt hinter den Warten auf Client(758) Knoten in Ihrer Sequenz zum Start des SUT.

Sind mehrere Resolver für ein bestimmtes Objekt, eine bestimmte Klasse oder global registriert, wird der zuletzt registrierte Resolver zuerst aufgerufen. Der erste Resolver, der einen nicht-null Wert zurückliefert, bestimmt das Ergebnis.

Da Resolver für jede im GUI angezeigte Instanz der Komponente bzw. Klasse aufgerufen werden, für die sie registriert wurden, empfiehlt es sich, zeitsparende Algorithmen bei der Implementierung zu verwenden. In einem Jython Skript ist z.B. die Ausführung von `string[0:3] == "abc"` deutlich schneller als etwa `string.startswith("abc")`.

Alle Exceptions, die während der Ausführung eines Resolvers auftreten, werden von der `ResolverRegistry` abgefangen. Es wird allerdings nur eine kurze Meldung und kein Stacktrace ausgegeben, weil insbesondere globale Resolver sehr oft aufgerufen werden können. Somit würde ein Resolver, der einen Bug hat, durch die Ausgabe von Stacktraces für jeden Fehler das Client-Terminal überfluten. Daher sollten Resolver ihre eigenen Fehlerbehandlungsroutinen enthalten. Dabei können zwar immer noch extrem viele Ausgaben erzeugt werden, aber diese sind dennoch hilfreicher als Java-Stacktraces.

Das `resolvers` Modul ist immer automatisch in allen SUT-Skript Knoten verfügbar.

Die meisten Beispiele sind in Jython implementiert. Im Kapitel Abschnitt 54.1.7(1163) finden Sie ein Beispiel für einen Groovy SUT-Skript Knoten.

54.1.3 `addResolver`

Die zentrale Funktion des `resolvers` Modul ist die generische Funktion `addResolver`, die anhand des Namens der definierten Methode sowie deren Parameter das jeweils passende Objekt identifiziert und dessen spezifische Funktion zur Registrierung des Resolvers aufruft.

`void addResolver(String resolverName, Method method, Object target=None, ...)`

Registriert den über die übergebene Methode festgelegten `Resolver` für die angegebenen Ziele. Falls bereits ein Resolver unter dem angegebenen Resolver-Namen registriert war, wird dieser zunächst deregistriert.

Parameter

`name`	Der Name unter dem der Resolver registriert werden soll.
`method`	Die Methode, welche die Methode des Resolvers implementiert. Der Name dieser Methode definiert den Typ des registrierten Resolvers, d.h. bei Groovy ist hier eine MethodClosure anzugeben. Zulässige Werte sind z.B.: `getName`, `getClassName`, `getGenericClassName`, `getFeature`, `getExtraFeatures`, `getItemName`, `getItemValue`, `getItemNameByIndex`, `getTree` und `getTreeColumn`, `isInterestingParent`, `getTooltip`, `getId`, `isEnabled`, `isVisible`, `getMainText`, `matchExtraFeature`, `isBusy`, `isGlassPaneFor`, `sync` und `applicationIsBusy`.
`target`	Ein oder mehrere optionale Ziele für die der Resolver registriert werden soll. Für jedes Ziel gibt es folgende Varianten: • Eine individuelle Komponente • Der Name einer Klasse Ist kein Ziel angegeben, wird der Resolver global für alle Komponenten registriert.

void addResolver(String resolverName, Object object, Object target=None, ...)
Registriert den oder die über die Methoden des Objekts festgelegten `Resolver` für die angegebenen Ziele. Falls bereits ein Resolver unter dem angegebenen Resolver-Namen registriert war, wird dieser zunächst deregistriert.

Parameter

name	Der Name unter dem der Resolver registriert werden soll.
object	Ein Objekt oder eine Klasse, die eine oder mehrere Resolver-Methoden bereitstellt. Anhand der Methodennamen werden entsprechende Resolver registriert. Die zulässigen Methodennamen sind identisch mit denen zuvor definierten.
target	Ein oder mehrere optionale Ziele für die der Resolver registriert werden soll. Für jedes Ziel gibt es folgende Varianten: • Eine individuelle Komponente • Der Name einer Klasse Ist kein Ziel angegeben, wird der Resolver global für alle Komponenten registriert.

Historie

Resolver haben in QF-Test bereits eine lange Historie. Bis QF-Test Version 4.1 war es notwendig, die jeweils spezifische Funktion zur Registrierung der Resolver Interfaces aufzurufen. Diese können weiterhin verwendet werden, sind hier aber nicht mehr beschrieben. Die flexible `addResolver`-Funktion ersetzt dabei u.a. diese bisherigen Funktionen des `resolvers` Moduls:

- `addNameResolver2(String name, Method method, Object target=None, ...)`
- `addClassNameResolver(String name, Method method, Object target=None, ...)`
- `addGenericClassNameResolver(String name, Method method, Object target=None, ...)`
- `addFeatureResolver2(String name, Method method, Object target=None, ...)`

- addExtraFeatureResolver(String name, Method method, Object target=None, ...)
- addItemNameResolver2(String name, Method method, Object target=None, ...)
- addItemValueResolver2(String name, Method method, Object target=None, ...)
- addTreeTableResolver(String name, Method getTable, Method getColumn=None, Object target=None)
- addTooltipResolver(String name, Method method, Object target=None, ...)
- addIdResolver(String name, Method method, Object target=None,...)

54.1.4 removeResolver

Die über das resolvers Modul registrierten Resolver können mittels der Funktion removeResolver deregistriert werden.

Resolver werden oft direkt nach dem Start der Applikation registriert und bleiben während der gesamten Testausführung aktiv. Es gibt jedoch auch Fälle, in denen ein Resolver nur bei der Arbeit mit einer bestimmten Komponente aktiviert und anschließend wieder entfernt werden soll. Sei es aus Performanzgründen oder weil die Wirkung des Resolvers nur in bestimmten Situationen gewünscht ist.

Es stehen zwei Funktionen zur Verfügung. Die erste, removeResolver deregistriert einen einzelnen Resolver, die zweite, removeAll entfernt alle vom Benutzer registrierten Resolver.

void removeAll()
Deregistriert alle über das resolvers Modul registrierten Resolver von allen Zielen, für die sie registriert waren.

void removeResolver(String name)
Deregistriert einen Resolver von allen Zielen, für die er registriert war.

Parameter

name	Der Name, unter dem der Resolver registriert war.

Im Beispiel wird zunächst ein unter dem Namen ”menuItems” registrierter Resolver entfernt, danach alle über das resolvers-Modul registrierten Resolver.

```
resolvers.removeResolver("menuItems")
resolvers.removeAll()
```

Beispiel 54.2: SUT-Skript zur Deregistrierung eines Resolvers

54.1.5 `listNames`

Gibt eine Liste der Namen der Resolver zurück, die über das `resolvers` Modul registriert wurden.

`List<String> listNames()`

Listet die Namen der registrierten Resolver auf.

Im Beispiel wird überprüft, ob ein spezifischer Resolver registriert wurde. Falls nicht wird dem Protokoll eine Fehlermeldung hinzugefügt.

```
if (! resolvers.listNames().contains("specialNames")) {
    rc.logError("Special Names Resolver nicht registriert!")
}
```

Beispiel 54.3: Groovy SUT-Skript zur Abfrage registrierter Resolver über das `resolvers` Modul

54.1.6 Zugriff auf die beste Beschriftung

Wenn aus einem Resolver auf die Beste Beschriftung[76] zugegriffen werden soll, dann die Methode `rc.engine.helper.getBestLabel()` genutzt werden.

`String getBestLabel(Component node)`

Gibt die Beste Beschriftung[76] zurück.

Parameter

`node`	Die Komponente, für die die Beschriftung ermittelt werden soll.
Rückgabewert	Die beste Beschriftung der Komponente.

Das Beispiel zeigt einen NameResolver, der die beste Beschriftung in das Name Attribut überträgt.

```
_h = rc.engine.helper
def getName(node, name):
    label = _h.getBestLabel(node)
        return label
resolvers.addResolver("labelAsName", getName, "TextField", "TextArea")
```

Beispiel 54.4: NameResolver für die beste Beschriftung

54.1.7 Das NameResolver Interface

3.1+

Der NameResolver beeinflusst den Name Attributwert eines Komponente Knotens.

Nachdem QF-Test den Namen einer Komponente ermittelt hat, erhalten die registrierten NameResolver die Chance, diesen zu überschreiben oder zu unterdrücken. Der erste Resolver, der einen nicht-null Wert zurückliefert, bestimmt das Ergebnis. Sind keine Resolver registriert oder liefern alle Resolver null, wird der ursprüngliche Name benutzt.

Ein NameResolver kann den Namen einer Komponente, der normalerweise mit setName() bei AWT/Swing, setId() oder das fx:id Attribut bei JavaFX, setData() bei SWT oder dem 'ID' Attribut eines DOM-Knotens bei einer Web-Anwendung gesetzt wird, verändern (oder überhaupt erst definieren). Dies kann äußerst nützlich sein, wenn der Source code nicht geändert werden kann, sei es weil fremder Code eingesetzt wird oder weil auf Bestandteile von komplexen Komponenten kein Zugriff besteht. Ein Beispiel für letzteren Fall ist der JFileChooser Dialog von Swing. Für diesen bringt QF-Test einen eigenen NameResolver mit, über den Sie im Kapitel "Die Standardbibliothek" des Tutorials weitere Informationen finden.

In einzelnen Fällen kann es sinnvoll sein, den Namen einer Komponente zu unterdrücken, z.B. wenn er nicht eindeutig ist, oder - wesentlich schlimmer - wenn er während der Laufzeit variiert. Um das zu erreichen muss getName den leeren String zurückliefern.

Technologien: AWT/Swing, JavaFX, SWT, Windows, Android, iOS. Für Web-Anwendungen sollte der in Verbesserte Komponentenerkennung mittels CustomWebResolver(1077) beschriebene CustomWebResolver installieren Knoten genutzt werden. Er ist für Web-Elemente optimiert. Aus Performance-Gründen sollte daher der NameResolver nur verwendet werden, wenn die dort bereitgestellte Funktionalität nicht ausreicht.

Ein NameResolver muss folgende Methode implementieren:

String getName(Object element, String name)

Legt den Namen einer Komponente fest.

Parameter

element	Das GUI Element dessen Name ermittelt werden soll.
name	Der ursprüngliche Name, den QF-Test ohne Resolver verwenden würde.
Rückgabewert	Der zu verwendende Name oder null, falls der Resolver das Element nicht behandelt. Durch Rückgabe des leeren Strings wird der ursprüngliche Name der Komponente unterdrückt.

Das erste Beispiel zeigt einen `NameResolver`, der für Komponenten der generischen Klasse `MenuItems`, für die der QF-Test Standard-Resolver keinen Namen ermitteln konnte, den Text des Menüeintrags als Namen der Komponente definiert.

```
def getName(menuItem, name):
    if not name:
        return menuItem.getLabel()
resolvers.addResolver("menuItems", getName, "MenuItem")
```

Beispiel 54.5: Jython `NameResolver` für `MenuItems`

Probieren Sie es aus! Kopieren Sie das obige Beispiel in einen SUT-Skript(720) Knoten und führen Sie diesen aus. Falls Ihre Anwendung auf SWT basiert, ersetzen Sie `getLabel()` durch `getText()`. Nehmen Sie dann einige Menü Aktionen in eine neue, leere Testsuite auf. Sie werden sehen, dass alle Komponenten ohne eigenen Namen für Menüeinträge nun einen Namen entsprechend ihrer Beschriftung erhalten. Falls `setName` in Ihrer Anwendung nicht verwendet wird und die Menü-Bezeichner weitgehend statisch sind, kann das sogar eine recht nützliche Sache sein.

Das zweite Beispiel zeigt einen `NameResolver`, der einem teilweise dynamischen Namen (z.B. ”lfd. Nr: 100478”) den festen Wert (”laufende Nummer”) zuweist. Er wird für eine spezifische Java-Swing-Klasse registriert.

```
def getName(menuItem, name):
    if name and name[0:7] == "lfd. Nr":
        return "laufende Nummer"
resolvers.addResolver("lfdNr", getName, "javax.swing.JMenuItem")
```

Beispiel 54.6: Jython `NameResolver` für eine spezifische Klasse

Der folgende `NameResolver` ist ein Groovy-Beispiel, das in einer SWT-Anwendung den Text des Menüpunktes als Namen einsetzt, sofern nicht bereits ein Name vom QF-Test

Standard-NameResolver vergeben wurde.

```
def getName(def menuItem, def name) {
    if (name == null) {
        return menuItem.getLabel()
    }
}
resolvers.addResolver("menuItems", this.&getName, "MenuItem")
// Hier ginge auch verkürzt:
//   resolvers.addResolver("menuItems", this, "MenuItem")
// da jedes Groovy-Skript ein Objekt darstellt und
// addResolver(...) auf Objekten alle Methoden des Objekts,
// wenn möglich, als Resolver registriert.
```

Beispiel 54.7: Einfacher Groovy Resolver für MenuItems

Ein Resolver kann gleichzeitig für mehrere Klassen von Elementen registriert werden:

```
def getName(com, name):
    return com.getText()
resolvers.addResolver("labels", getName, "Label", "Button")
```

Beispiel 54.8: Registrieren eines Resolver für mehrere Klassen

54.1.8 Das GenericClassNameResolver Interface

4.0+

Ein GenericClassNameResolver kann die generische Klasse (Kapitel 61[1329]) bestimmen, die QF-Test für eine Komponente aufzeichnet. Er kann dazu genutzt werden aufgezeichnete Komponenten lesbarer zu gestalten und auch um gezielt weitere Resolver auf die neu erstellten Klassen zu registrieren.

Technologien: alle

Web

Dieser Resolver sollte für Web-Anwendungen nur dann verwendet werden, wenn die Möglichkeiten des in Verbesserte Komponentenerkennung mittels CustomWebResolver[1077] beschriebenen CustomWebResolver installieren Knoten nicht ausreichen.

Nachdem QF-Test den generischen Klassennamen einer Komponente ermittelt hat, erhalten die registrierten GenericClassNameResolver die Chance, diesen zu überschreiben. Der erste Resolver, der einen nicht-null Wert zurückliefert, bestimmt das Ergebnis. Sind keine Resolver registriert oder liefern alle Resolver null, wird der ursprüngliche generische Klassenname benutzt.

Aus Performanzgründen werden die Klassen zwischengespeichert, daher wird der Resolver für jede Komponente höchstens einmal aufgerufen. Falls Sie den Resolver an-

passen und testen wollen, müssen Sie den Bereich mit der Komponente neu laden bzw. schließen und erneut öffnen.

Web Falls ein Element bereits über einen CustomWebResolver eine Generische Klasse zugewiesen bekommen hat, werden für dieses Element keine GenericClassNameResolvers mehr aufgerufen.

Ein `GenericClassNameResolver` muss folgende Methode implementieren:

`String getGenericClassName(Object element, String name)`

Legt den generischen Klassennamen einer Komponente fest.

Parameter

`element`	Das GUI Element dessen generischer Klassenname ermittelt werden soll.
`name`	Der ursprüngliche generische Klassenname, den QF-Test ohne Resolver verwenden würde. Kann auch Null sein.
Rückgabewert	Der zu verwendende generische Name oder null, falls der Resolver das Element nicht behandelt. Durch Rückgabe des leeren Strings wird der ursprünglich ermittelte generische Name der Komponente unterdrückt.

54.1.9 Das `ClassNameResolver` Interface

3.1+

Ein `ClassNameResolver` kann die Klasse bestimmen, die QF-Test für eine Komponente aufzeichnet. Er kann dazu genutzt werden aufgezeichnete Komponenten lesbarer zu gestalten und auch um gezielt weitere Resolver auf die neu erstellten Klassen zu registrieren, wobei wir grundsätzlich die Verwendung von generischen Klassen empfehlen. Für die Registrierung von Generische Klassen(1329) sollten Sie den `GenericClassNameResolver` (Abschnitt 54.1.8(1165)) verwenden.

Technologien: alle

Web Dieser Resolver sollte für Web-Anwendungen nur dann verwendet werden, wenn die Möglichkeiten des in Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077) beschriebenen CustomWebResolver installieren Knoten nicht ausreichen.

Ein `ClassNameResolver` muss folgende Methode implementieren:

String getClassName(Object element, String name)

Legt den Klassennamen einer Komponente fest.

Parameter

element	Das GUI Element dessen Klassenname ermittelt werden soll.
name	Der ursprüngliche Klassenname, den QF-Test ohne Resolver verwenden würde.
Rückgabewert	Der zu verwendende Klassenname oder null, falls der Resolver das Element nicht behandelt.

Nachdem QF-Test den Klassennamen einer Komponente ermittelt hat, erhalten die registrierten `ClassNameResolver` die Chance, diesen zu überschreiben. Der erste Resolver, der einen nicht-null Wert zurückliefert, bestimmt das Ergebnis. Sind keine Resolver registriert oder liefern alle Resolver null, wird der ursprüngliche Klassenname benutzt. Ein solcher Resolver kann jeden beliebigen Klassennamen zurückliefern. Sämtliche QF-Test internen Methoden werden diese Klassen auch wie normale Klassen behandeln.

Aus Performanzgründen werden die Klassen zwischengespeichert, daher wird der Resolver für jede Komponente höchstens einmal aufgerufen. Falls Sie den Resolver anpassen und testen wollen, müssen Sie den Bereich mit der Komponente neu laden bzw. schließen und erneut öffnen.

54.1.10 Das `FeatureResolver` Interface

3.1+

Der `FeatureResolver` beeinflusst das Merkmal Attribut einer Komponente.

Nachdem QF-Test das Merkmal einer Komponente ermittelt hat, erhalten die registrierten `FeatureResolver` die Chance, dieses zu überschreiben oder zu unterdrücken. Der erste Resolver, der einen nicht-null Wert zurückliefert, bestimmt das Ergebnis. Sind keine Resolver registriert oder liefern alle Resolver null, wird das ursprüngliche Merkmal benutzt.

Um das Merkmal einer Komponente zu unterdrücken, muss `getFeature` den leeren String zurückliefern.

Technologien: AWT/Swing, JavaFX, SWT, Windows, Android, iOS. Für Web-Anwendungen sollte der in Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077) beschriebene CustomWebResolver installieren Knoten genutzt werden. Er ist für Web-Elemente optimiert. Aus Performance-Gründen sollte daher der `FeatureResolver` nur verwendet werden, wenn die dort bereitgestellte Funktionalität nicht ausreicht.

Ein `FeatureResolver` muss folgende Methode implementieren:

String getFeature(Object element, String feature)

Legt das Merkmal einer Komponente fest.

Parameter

element	Das GUI Element dessen Merkmal ermittelt werden soll.
feature	Das ursprüngliche Merkmal, das QF-Test ohne Resolver verwenden würde.
Rückgabewert	Das zu verwendende Merkmal oder null, falls der Resolver das Element nicht behandelt. Durch Rückgabe des leeren Strings wird das ursprüngliche Merkmal der Komponente unterdrückt.

Das folgende Beispiel bezieht sich auf ein Java/Swing Panel, das eine Beschriftung in der Umrandung enthält. Diese Beschriftung soll als Merkmal gesetzt werden.

```
def getFeature(com, feature):
    try:
        title = com.getBorder().getInsideBorder().getTitle()
        if title != None:
            return title
    except:
        pass
resolvers.addResolver("paneltitle", getFeature, "Panel")
```

Beispiel 54.9: Ein FeatureResoler für Java/Swing Panels

54.1.11 Das ExtraFeatureResolver Interface

Der ExtraFeatureResolver kann ein Weiteres Merkmal in der Weitere Merkmale Tabelle einer Komponente ändern, hinzufügen oder löschen. Hierzu stellt das Interface eine Reihe von Methoden zur Verfügung.

Ein Objekt der Klasse de.qfs.apps.qftest.shared.data.ExtraFeature repräsentiert ein weiteres Merkmal eines GUI Elements, bestehend aus Name und Wert, sowie Informationen darüber, ob für das Feature ein Treffer erwartet wird, ob der Wert ein regulärer Ausdruck (vgl. Abschnitt 49.3(1023)) ist und ob die Aussage negiert werden soll. Für mögliche Zustände definiert die Klasse die Konstanten STATE_IGNORE, STATE_SHOULD_MATCH und STATE_MUST_MATCH.

Nachdem QF-Test die weiteren Merkmale eines GUI Elements ermittelt hat, erhalten die registrierten ExtraFeatureResolver die Chance, diese zu modifizieren. Im Gegensatz zu anderen Resolvern bricht QF-Test nicht ab, sobald ein ExtraFeatureResolver einen nicht-null Wert zurückliefert, sondern übernimmt diesen Wert als Parameter für den nächsten Resolver. Hierdurch können mehrere

ExtraFeatureResolver registriert werden, die jeweils ein spezielles Merkmal behandeln. Sind keine Resolver registriert oder liefern alle Resolver null, macht QF-Test mit dem ursprünglich ermittelten Satz von Merkmalen weiter.

Um die getExtraFeatures Methode implementieren zu können, müssen Sie natürlich die API der beteiligten Klassen ExtraFeature und ExtraFeatureSet kennen. Diese werden nach den Beispielen zum ExtraFeatureResolver beschrieben.

Technologien: AWT/Swing, JavaFX, SWT, Windows, Android, iOS. Für Web-Anwendungen sollte der in Verbesserte Komponentenerkennung mittels CustomWebResolver(1077) beschriebene CustomWebResolver installieren Knoten genutzt werden. Er ist für Web-Elemente optimiert. Aus Performance-Gründen sollte daher der ExtraFeatureResolver nur verwendet werden, wenn die dort bereitgestellte Funktionalität nicht ausreicht.

Um die qfs:label*-Varianten der weiteren Merkmale bei der Aufnahme und der Wiedergabe konsistent zu halten und sie auf SmartIDs abbilden zu können, ist es notwendig, bestimmte Regeln einzuhalten. Diese gelten nicht für das qfs:label Weiteres Merkmal, das für sich alleine steht.

Bei der Behandlung von qfs:label*-Varianten in einem ExtraFeatureResolver muss darauf geachtet werden, dass der komplette Variantensatz in sich konsistent bleibt. Das heißt:

- Es darf höchstens eine qfs:label*-Variante mit dem Status ”Sollte übereinstimmen” geben. Der Rest sollte auf ”Ignorieren” stehen.
- qfs:labelBest sollte entweder die ”Sollte übereinstimmen”-Variante sein oder den gleichen Wert wie die ”Sollte übereinstimmen”-Variante haben.

Bei der Ermittlung der zugehörigen Beschriftungen erstellt QF-Test die qfs:label*-Varianten gemäß dieser Regeln. Die ExtraFeatureSet Klasse berücksichtigt die Regeln ebenfalls, wenn von einem ExtraFeatureResolver aus darauf zugegriffen wird. Dadurch wird die Einhaltung der Regeln bei der Arbeit mit einem ExtraFeatureResolver erleichtert. Dies bedeutet:

- Wenn der Wert von qfs:labelBest geändert wird und es den Status ”Ignorieren” hat, wird der Wert der ”Sollte übereinstimmen”-Variante automatisch ebenfalls geändert.
- Wenn der Wert der ”Sollte übereinstimmen”-Variante geändert wird, wird der Wert von qfs:labelBest automatisch ebenfalls geändert.
- Wenn eine qfs:label*-Variante auf ”Sollte übereinstimmen” gesetzt wird, werden alle anderen auf ”Ignorieren” gesetzt und qfs:labelBest entsprechend aktualisiert.

Hinweis

Wenn qfs:label*-Varianten mit dem Status ”Muss übereinstimmen” oder mit einem regulären Ausdruck existieren, wird die Regelbehandlung deaktiviert.

Ein ExtraFeatureResolver muss folgende Methode implementieren:

ExtraFeatureSet getExtraFeatures(Object element, ExtraFeatureSet features)

Erzeugt ein ExtraFeatureSet, das weitere Merkmale einer Komponente festlegt.

Parameter

element	Das GUI Element dessen weitere Merkmale ermittelt werden sollen.
features	Ein Satz von weiteren Merkmale, die QF-Test selbst festgelegt hat, ein leerer Satz falls es keine solchen Merkmale gibt. Dieser Satz kann modifiziert oder ignoriert und durch einen anderen ersetzt werden.
Rückgabewert	Der modifizierte original Satz von Merkmalen oder ein neuer Satz, falls der Resolver das GUI Element behandelt, andernfalls null. Die ursprünglichen Merkmale können durch Rückgabe eines leeren Satzes unterdrückt werden.

Das erste Beispiel zeigt einen ExtraFeatureResolver, der den Titel eines Java/Swing Dialogs als Weiteres Merkmal mit dem Status ”muss übereinstimmen” (STATE_MUST_MATCH) hinzufügt. Dies ist sehr nützlich, wenn es bei der Komponentenerkennung eines Dialogs auf den richtigen Titel ankommt.

```
def getExtraFeatures(node, features):
    try:
        title = node.getTitle()
        features.add(resolvers.STATE_MUST_MATCH,"dialog.title", title)
        return features
    except:
        pass
resolvers.addResolver("dialog title", getExtraFeatures,"Dialog")
```

Beispiel 54.10: ExtraFeatureResolver der ein weiteres Feature für Java/Swing Dialoge erstellt

Das folgende Beispiel zeigt, wie man ein vorhandenes Weiteres Merkmal ändert. Da es sich um qfs:labelBest handelt, sorgt QF-Test wie oben beschrieben dafür, dass die qfs:label*-Varianten in sich konsistent bleiben.

```
def getExtraFeatures(node, features):
    label = features.get("qfs:labelBest")
    if label and label.getValue() == "unwanted":
        label.setValue("wanted")
        return features
resolvers.addResolver("change label", getExtraFeatures)
```

Beispiel 54.11: ExtraFeatureResolver, der ein Weiteres Merkmal ändert

Das nächste Beispiel zeigt, wie man den Status der qfs:label*-Variante mit dem Status ”Sollte übereinstimmen” auf ”Ignorieren” gesetzt:

```
def getExtraFeatures(node, features) {
    def labelFeature = features.getShouldMatchLabel()
    if (labelFeature) {
        labelFeature.setState(resolvers.STATE_IGNORE)
        return features
    }
}
resolvers.addResolver("get label example", this)
```

Beispiel 54.12: Ein ExtraFeatureResolver (in Groovy), welcher den Status einer qfs:label*-Variante ändert

Dank der oben beschriebenen Regeln für den Satz der qfs:label*-Varianten, kann ein einfacher ExtraFeatureResolver, mit dem folgenden Inhalt

```
def getExtraFeatures(node, features):
    label = features.get("qfs:labelBest")
    if label and label.getValue() == "unwanted":
        label.setValue("wanted")
        return features
resolvers.addResolver("change label", getExtraFeatures)
```

Beispiel 54.13: ExtraFeatureResolver changing an existing Weiteres Merkmal

einfach in Beispiel ExtraFeatureResolver, der ein Weiteres Merkmal ändert (1171) umgewandelt werden.

Im Folgenden finden Sie die Beschreibung der APIs der Klassen ExtraFeature und ExtraFeatureSet.

`ExtraFeature ExtraFeature(String name, String value)`

Erzeugt ein neues ExtraFeature mit Status STATE_IGNORE.

Parameter

`name`	Der Name des ExtraFeatures.
`value`	Der Wert des ExtraFeatures.

`ExtraFeature ExtraFeature(int state, String name, String value)`

Erzeugt ein neues ExtraFeature mit gegebenem Status.

Parameter

`state`	Der Status des ExtraFeatures. Mögliche Werte sind resolvers.STATE_IGNORE, resolvers.STATE_SHOULD_MATCH, resolvers.STATE_MUST_MATCH.
`name`	Der Name des ExtraFeatures.
`value`	Der Wert des ExtraFeatures.

`String getName()`

Liefert den Namen des ExtraFeatures.

Rückgabewert	Des Name des ExtraFeatures.

`boolean getNegate()`

Liefert den negate Status des ExtraFeatures.

Rückgabewert	der negate Status des ExtraFeatures.

`boolean getRegexp()`

Liefert den regexp Status des ExtraFeatures.

Rückgabewert	der regexp Status des ExtraFeatures.

`int getState()`

Liefert den Status des ExtraFeatures.

Rückgabewert	Der Status des ExtraFeatures.

`String getValue()`

Liefert den Wert des ExtraFeatures.

Rückgabewert	Der Wert des ExtraFeatures.

`void setName(String name)`

Setze den Namen des ExtraFeatures.

Parameter

`name`	Der zu setzende Name.

`void setNegate(boolean negate)`

Setzt den negate Status des ExtraFeatures.

Parameter

`negate`	Der zu setzende negate Status.

`void setRegexp(boolean regexp)`

Setzt den regexp Status des ExtraFeatures.

Parameter

`regexp`	Der zu setzende regexp Status.

`void setState(int state)`

Setzt den Status des ExtraFeatures.

Parameter

`state`	Der zu setzende Status.

`void setValue(String value)`

Setzt den Wert des ExtraFeatures.

Parameter

`value`	Der zu setzende Wert.

Die Klasse `de.qfs.apps.qftest.shared.data.ExtraFeatureSet` bündelt `ExtraFeatures` in einen Satz:

`ExtraFeatureSet ExtraFeatureSet()`

Erzeugt ein neues, leeres ExtraFeatureSet.

`void add(ExtraFeature extraFeature)`

Fügt dem Satz ein ExtraFeature hinzu. Existiert schon ein ExtraFeature mit demselben Namen, wird dieses ersetzt.

Parameter

`extraFeature`	Das hinzuzufügende ExtraFeature.

`void add(String name, String value)`

Fügt dem Satz ein neues ExtraFeature mit Status STATE_IGNORE hinzu. Existiert schon ein ExtraFeature mit demselben Namen, wird dieses ersetzt.

Parameter

`name`	Der Name des ExtraFeatures.
`value`	Der Wert des ExtraFeatures.

`void add(int state, String name, String value)`
Fügt dem Satz ein neues ExtraFeature mit gegebenem Status hinzu. Existiert schon ein ExtraFeature mit demselben Namen, wird dieses ersetzt.

Parameter

`state`	Der Status des ExtraFeatures. Mögliche Werte sind resolvers.STATE_IGNORE, resolvers.STATE_SHOULD_MATCH, resolvers.STATE_MUST_MATCH.
`name`	Der Name des ExtraFeatures.
`value`	Der Wert des ExtraFeatures.

`ExtraFeature get(String name)`

Liefert ein ExtraFeature des Satzes.

Parameter

`name`	Der Name des abzufragenden ExtraFeatures.
Rückgabewert	Das ExtraFeature oder null, falls kein ExtraFeature unter diesem Namen im Satz gespeichert ist.

`ExtraFeature getShouldMatchLabel()`
Liefert die `qfs:label*` ExtraFeature Variante mit dem Status "Sollte übereinstimmen". Falls mehr als eine Variante diesen Status hat, wird die erste davon zurückgeliefert.

Rückgabewert	Das ExtraFeature oder null, falls es keine `qfs:label*` ExtraFeature Variante mit dem Status "Sollte übereinstimmen" gibt.

`ExtraFeature remove(String name)`

Entfernt ein ExtraFeature aus dem Satz.

Parameter

`name`	Der Name des zu entfernenden ExtraFeatures.
Rückgabewert	Das entfernt ExtraFeature oder null, falls kein ExtraFeature unter diesem Namen im Satz gespeichert war.

`ExtraFeature[] toArray()`

Liefert alle ExtraFeatures des Satzes.

Rückgabewert	Ein Array mit allen ExtraFeatures, nach Namen sortiert.

54.1.12 Das `ItemNameResolver` Interface

3.1+

Ein `ItemNameResolver` kann die Textdarstellung des Index zur Adressierung eines Unterelements einer komplexen Komponente verändern (oder überhaupt erst definieren).

Nachdem QF-Test einen Namen für den Index eines Unterelements ermittelt hat, erhalten die registrierten ItemNameResolver die Chance, diesen zu überschreiben. Der erste Resolver, der einen nicht-null Wert zurückliefert, bestimmt das Ergebnis. Sind keine Resolver registriert oder liefern alle Resolver null, wird der ursprüngliche Name benutzt.

Technologien: AWT/Swing, JavaFX, SWT, Windows, Android, iOS. Für Web-Anwendungen sollte der in Verbesserte Komponentenerkennung mittels CustomWebResolver[1077] beschriebene CustomWebResolver installieren Knoten genutzt werden. Er ist für Web-Elemente optimiert. Aus Performance-Gründen sollte daher der ItemNameResolver nur verwendet werden, wenn die dort bereitgestellte Funktionalität nicht ausreicht.

Ein ItemNameResolver muss folgende Methode implementieren:

String getItemName(Object element, Object item, String name)
Legt den Namen zur textuellen Repräsentation des Index eines Unterelements einer komplexen Komponente fest.

Parameter

element	Das GUI Element zu dem das Unterelement gehört.
item	Das Unterelement, dessen Name ermittelt wird. Sein Typ hängt von der Art des GUI Elements und der registrierten ItemResolver ab, wie in Abschnitt 54.3.5[1206] beschrieben.
name	Der ursprüngliche Name, den QF-Test ohne Resolver verwenden würde.
Rückgabewert	Der Name oder null, falls der Resolver dieses Element oder Unterelement nicht behandelt.

Es folgt ein Beispiel für einen ItemNameResolver, der die ID einer JTable Spalte für den Index zugänglich macht:

```
def getItemName(tableHeader, item, name):
    id = tableHeader.getColumnModel().getColumn(item).getIdentifier()
    if id:
        return str(id)
resolvers.addResolver("tableColumnId", getItemName,
                      "javax.swing.table.JTableHeader")
```

Beispiel 54.14: Ein ItemNameResolver für JTableHeader

54.1.13 Das ItemValueResolver Interface

3.1+

Der ItemValueResolver wird verwendet, um die Prüfung des Textes von Elementen zu optimieren.

Ein `ItemValueResolver` kann die Textdarstellung des Wertes eines Unterelements einer komplexen Komponente verändern (oder überhaupt erst definieren), der für einen Check Text oder Text auslesen Knoten verwendet wird.

Nachdem QF-Test einen Wert für ein Unterelement ermittelt hat, erhalten die registrierten `ItemValueResolver` die Chance, diesen zu überschreiben. Der erste Resolver, der einen nicht-null Wert zurückliefert, bestimmt das Ergebnis. Sind keine Resolver registriert oder liefern alle Resolver null, wird der ursprüngliche Wert benutzt.

Technologien: AWT/Swing, JavaFX, SWT, Windows, Android, iOS. Für Web-Anwendungen sollte der in Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077) beschriebene CustomWebResolver installieren Knoten genutzt werden. Er ist für Web-Elemente optimiert. Aus Performance-Gründen sollte daher der `ItemValueResolver` nur verwendet werden, wenn die dort bereitgestellte Funktionalität nicht ausreicht.

Ein `ItemValueResolver` muss folgende Methode implementieren:

`String getItemValue(Object element, Object item, String value)`
Legt den Wert der eines Unterelements einer komplexen Komponente fest, wie er für Check Text(806) oder Text auslesen(839) Knoten verwendet wird.

Parameter

`element`	Das GUI Element zu dem das Unterelement gehört.
`item`	Das Unterelement, dessen Wert ermittelt wird. Sein Typ hängt von der Art des GUI Elements und der registrierten `ItemResolver` ab, wie in Abschnitt 54.3.5(1206) beschrieben.
`value`	Der ursprüngliche Wert, den QF-Test ohne Resolver verwenden würde.
Rückgabewert	Der Wert oder null, falls der Resolver dieses Element oder Unterelement nicht behandelt.

54.1.14 Das `TreeTableResolver` Interface

Ein `TreeTableResolver` hilft QF-Test TreeTable Komponenten als solche zu erkennen. Eine TreeTable ist eine Mischung aus einer Tabelle und einem Baum. Sie ist keine Standard-Swing-Komponente, allerdings werden die meisten TreeTables ähnlich implementiert, indem ein Baum als Renderer für eine Spalte der Tabelle verwendet wird. Wenn QF-Test eine TreeTable identifiziert, behandelt es die Zeilenindizes aller Zellen der Tabelle wie Baumindizes, was in diesem Zusammenhang wesentlich bessere Ergebnisse liefert. Außerdem werden Geometrie Informationen für Zellen in der Spalte des Baums basierend auf Baumknoten statt auf Tabellenzellen ermittelt.

Technologien: AWT/Swing

Hinweis

Dieses Interface ist nur für AWT/Swing relevant. Mehrspaltige Bäume in SWT und JavaFX werden von QF-Test automatisch unterstützt. Für Web-Frameworks ist die TreeTable im entsprechenden (Custom-)Web-Resolver definiert.

Ein `TreeTableResolver` muss die beiden folgenden Methoden implementieren:

`JTree getTree(JTable table)`

Ermittelt die JTree Komponente mit deren Hilfe eine TreeTable implementiert ist.

Parameter

`table`	Die Tabelle für die der Baum ermittelt werden soll.
Rückgabewert	Der Baum oder null, falls es sich um eine normale Tabelle handelt.

`int getTreeColumn(JTable table)`

Ermittelt den Index der Spalte des Baums in einer TreeTable. Die meisten Implementierungen haben den Baum in der ersten Spalte. In diesem Fall muss 0 zurückgegeben werden.

Parameter

`table`	Die Tabelle für die der Spaltenindex des Baums ermittelt werden soll.
Rückgabewert	Der Spaltenindex oder -1, falls es sich um eine normale Tabelle handelt. Der Spaltenindex muss immer in Model-Koordinaten geliefert werden, nicht in View-Koordinaten.

Die meisten `TreeTableResolver` sind trivial zu implementieren. Das folgende Beispiel in Jython genügt bereits für die `org.openide.explorer.view.TreeTable` Komponente der populären netBeans IDE, vorausgesetzt, dass der Resolver für die TreeTable Klasse registriert wird.

```
def getTreeMethod(table):
        return table.getCellRenderer(0,0)
    def getColumn(table):
        return 0
    resolvers.addResolver("treetableResolver", getTreeMethod, \
    getColumn, "org.openide.explorer.view.TreeTable")
```

Beispiel 54.15: `TreeTableResolver` für die netBeans IDE

Das folgende Beispiel zeigt einen typischen `TreeTableResolver`.

```
def getTree(table):
    return table.getTree()
def getColumn(table):
    return 0
resolvers.addResolver("treeTable", getTree, getColumn,
                                "my.package.TreeTable")
```

Beispiel 54.16: `TreeTableResolver` für Swing TreeTable mit optionaler `getColumn` Methode

Da praktisch alle TreeTables den Baum in der ersten Spalte der Tabelle darstellen, ist die `getColumn` Methode optional. Wird keine übergeben, wird automatisch eine default Implementierung für die erste Spalte erstellt:

```
def getTree(table):
    return table.getTree()
resolvers.addResolver("treeTable", getTree, None,
                                "my.package.TreeTable")
```

Beispiel 54.17: Vereinfachter `TreeTableResolver`

Falls keine dedizierte `getTree` Methode vorhanden ist, hilft meist der `CellRenderer` der Spalte, die den Baum enthält (typischerweise 0), da dieser oft von `JTree` abgeleitet ist.

```
def getTree(table):
    return table.getCellRenderer(0,0)
resolvers.addResolver("treeTable", getTree,
                                "my.package.TreeTable")
```

Beispiel 54.18: Einfacher `TreeTableResolver`, der die `getCellRenderer` Methode nutzt

54.1.15 Das `InterestingParentResolver` Interface

Ein `InterestingParentResolver` kann beeinflussen, ob eine Komponente als für die Komponentenerkennung interessant bzw. uninteressant betrachtet wird. Dies wiederum legt fest, ob für die Komponente ein Komponente Knoten angelegt wird.

Technologien: AWT/Swing, JavaFX, SWT, Windows, Android, iOS. Für Web-Anwendungen sollte der in Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077) beschriebene CustomWebResolver installieren Knoten genutzt werden. Er ist für Web-Elemente optimiert. Aus Performance-Gründen sollte daher der

InterestingParentResolver nur verwendet werden, wenn die dort bereitgestellte Funktionalität nicht ausreicht.

Ein InterestingParentResolver muss folgende Methode implementieren:

Boolean isInterestingParent(Object parent, boolean interesting)

Liefert zurück, ob eine Komponente interessant bzw. uninteressant ist.

Parameter

parent	Die zu prüfende Komponente.
interesting	Ob QF-Test diese Komponente bis jetzt als interessant behandelt hat.
Rückgabewert	Boolean.TRUE wenn interessant, Boolean.FALSE wenn nicht, null, wenn dieser Resolver das nicht entscheiden soll.

54.1.16 Das TooltipResolver Interface

4.1+

Ein ToolTipResolver kann den Tooltip einer Komponente beeinflussen. Dieser Tooltip wird bei Prüfungen und im weiteren Merkmal 'qfs:labelTooltip' verwendet.

Technologien: AWT/Swing, JavaFX, SWT. Für Web-Anwendungen sollte der in Verbesserte Komponentenerkennung mittels CustomWebResolver(1077) beschriebene CustomWebResolver installieren Knoten genutzt werden. Er ist für Web-Elemente optimiert. Aus Performance-Gründen sollte daher der TooltipResolver nur verwendet werden, wenn die dort bereitgestellte Funktionalität nicht ausreicht.

Ein TooltipResolver muss folgende Methode implementieren:

String getTooltip(Object element, String tooltip)

Legt den Tooltip der Komponente fest.

Parameter

element	Das GUI Element dessen Tooltip ermittelt werden soll.
tooltip	Der ursprüngliche Tooltip, den QF-Test ohne Resolver verwenden würde.
Rückgabewert	Der zu verwendende Tooltip oder null, falls der Resolver das Element nicht behandelt. Durch Rückgabe des leeren Strings wird der ursprüngliche Tooltip der Komponente unterdrückt.

54.1.17 Das IdResolver Interface

Web

Ein IdResolver kann das Attribut 'ID' eines DOM-Knoten modifizieren bzw. unterdrücken. Wenn QF-Test die Knoten einer Webseite registriert, wird das Attribut 'ID' dieser Knoten gespeichert. Abhängig von der Option ID-Attribut als Name verwenden(567) wird der Wert dieses Attributes auch als Komponentenname für die Erkennung herangezogen. Da viele Webseiten bzw. Frameworks automatisch generierte IDs verwenden, ist oft eine Modifikation dieser IDs nötig, um eine stabile bzw. eindeutige Erkennung zu erhalten.

Es gibt drei Möglichkeiten mit solchen automatisch generierten IDs umzugehen:

- Die einfachste Variante solche automatischen IDs zu ignorieren, ist es, beim generierten CustomWebResolver installieren(902) Knoten die Kategorie autoIdPatterns zu setzen. Dort können Sie konkrete Werte, z.B. meineAutoId oder auch reguläre Ausdrücke (vgl. Abschnitt 49.3(1023)), z.B. auto.* spezifizieren, um alle IDs, die mit auto beginnen, zu ignorieren.
- Falls Sie ein eigenes Attribut eingeführt haben, welches anstatt des originalen Attributs 'ID' verwendet werden soll, dann rufen Sie den CustomWebResolver installieren(902) Knoten auf. Dort können Sie eigene Attribute in der Kategorie customIdAttributes spezifizieren. Diese Attribute werden nun anstatt des Attributs 'ID' verwendet werden.
- Sie können mit der Option Alle Ziffern aus 'ID'-Attributen eliminieren(568) konfigurieren, dass nur Ziffern aus den IDs gelöscht werden sollen.
- Falls Sie eine komplexere Logik implementieren wollen, brauchen Sie einen eigenen IdResolver.

Die oben genannten Methoden schließen einander nicht aus und können auch miteinander kombiniert werden. Falls Sie sich für eine eigene Logik per Resolver entscheiden, sollten Sie allerdings immer einen IdResolver verwenden, weil die ID eines Knotens an verschiedenen Stellen wieder auftauchen kann. Vor allem im Attribut Name(932) des Knotens (abhängig von der Option ID-Attribut als Name verwenden(567)), im Attribut Merkmal(932) und im Attribut Weitere Merkmale(933). Daher ist es viel effizienter, einmalig die ID mittels der oben genannten Möglichkeiten zu verändern, als getrennte Name-, Feature- und ExtraFeatureResolvers zu implementieren. Noch wichtiger ist der Umstand, dass die Veränderung der ID eines Knotens großen Einfluss auf die Eindeutigkeit dieser ID haben kann. Der Mechanismus zum Ermitteln von Namen auf Basis der ID nimmt darauf Rücksicht, so dass ein IdResolver auch nicht eindeutige IDs liefern darf. Ein NameResolver muss dagegen eindeutige Namen liefern.

Technologien: Web

Ein IdResolver muss folgende Methode implementieren:

String getId(DomNode node, String id)
Legt die ID eines DomNode Knotens fest. Die so bestimmte ID wird gespeichert und kann später mittels node.getId() ausgelesen werden. Dagegen ermittelt node.getAttribute("id") immer das ursprüngliche, unmodifizierte Attribut 'ID'.

Parameter

node	Der DomNode Knoten dessen ID ermittelt werden soll.
id	Die ID, die QF-Test für diesen Knoten ermittelt hat, möglicherweise nach Entfernen der darin enthaltenen Ziffern, abhängig von der Option Alle Ziffern aus 'ID'-Attributen eliminieren(568). Um den Resolver auf Basis des ursprünglichen 'ID' Attributs zu implementieren, ermitteln Sie dieses einfach via node.getAttribute("id").
Rückgabewert	Die ID oder null, falls keine ID festgelegt werden kann. Durch Rückgabe des leeren Strings wird der eigentliche ID der Komponente unterdrückt.

54.1.18 Das EnabledResolver Interface

4.1+

Ein EnabledResolver beeinflusst, wann eine Komponente als aktiv oder inaktiv angesehen wird. Bei AWT/Swing Komponenten kann dies direkt per Attribut erfolgen, Web und JavaFX benötigen dafür spezielle Stylesheet-Klassen, die dann mit Hilfe des EnabledResolvers ausgewertet werden.

Technologien: JavaFX, Web, Windows, Android, iOS

Ein EnabledResolver muss folgende Methode implementieren:

Boolean isEnabled(Object element, boolean enabled)
Legt fest, ob eine Komponente als aktiv interpretiert wird.

Parameter

element	Das GUI-Element, dessen Zustand bestimmt werden soll.
enabled	Der Zustand, den QF-Test ohne Hilfe des Resolvers ermittelt hätte.
Rückgabewert	True oder false, bzw. null, falls der Resolver das Element nicht behandelt.

Das folgende Beispiel bestimmt den Enabled-Zustand eines Webknotens anhand der CSS-Klasse v-disabled.

```
def isEnabled(element):
    try:
        return not element.hasCSSClass("v-disabled")
    except:
        return True
resolvers.addResolver("vEnabledResolver",isEnabled, \
                      "DOM_NODE")
```

Beispiel 54.19: Ein `EnabledResolver`

54.1.19 Das `VisibilityResolver` Interface

4.1+

Ein `VisibilityResolver` beeinflusst, wann ein Web-Element als sichtbar angesehen wird.

Technologien: Web, Windows, Android, iOS

Ein `VisibilityResolver` muss folgende Methode implementieren:

`Boolean isVisible(Object element, boolean visible)`

Bestimmt, ob eine Komponente als sichtbar angesehen wird.

Parameter

`element`	Der Web-Knoten, dessen Sichtbarkeit bestimmt werden soll.
`visible`	Der Zustand, den QF-Test ohne Hilfe des Resolvers ermittelt hätte.
Rückgabewert	True oder false, bzw. null, falls der Resolver das Element nicht behandelt.

Das folgende Beispiel setzt die Sichtbarkeit eines Web-Elementes zusätzlich auf false, wenn es durchsichtig ist.

```
import re
def getOpacity(element):
    style = element.getAttribute("style")
    if not style:
        return 1
    m = re.search("opacity:\s*([\d\.]+)", style)
    if m:
        return float(m.group(1)) == 0.4
    else:
        return 1
def isVisible(element,visible):
    while visible and element:
        visible = getOpacity(element) > 0
        element = element.getParent()
    return visible
resolvers.addResolver("opacityResolver",isVisible)
```

Beispiel 54.20: Ein `VisibilityResolver`

54.1.20 Das `MainTextResolver` Interface

4.1+

Ein `MainTextResolver` ermittelt den ”Haupttext” einer Komponente, standardmäßig die erste Zeile, der zum Beispiel für das Merkmal(932) oder die `qfs:label*`-Varianten(74) verwendet werden soll.

Technologien: AWT/Swing, JavaFX, SWT, Web, Windows, Android, iOS

Ein `MainTextResolver` muss folgende Methode implementieren:

`String getMainText(Object element, String text)`

Ermittelt den ”Haupttext” einer Komponente

Parameter

`element`	Das GUI-Element, dessen Text ermittelt werden soll.
`text`	Der Text, den QF-Test ohne den Resolver verwenden würde.
Rückgabewert	Der ”Haupttext” oder null, falls der Resolver das Element nicht behandelt. Durch Rückgabe des leeren Strings wird der Komponente kein Text zugeordnet.

Das folgende Beispiel entfernt aus dem ”Haupttext” aller Komponenten den String `TO-DO`.

```
def getMainText(element,text):
    if text:
        return text.replace("TO-DO","")
resolvers.addResolver("removeMarkFromText", getMainText)
```

Beispiel 54.21: Ein MainTextResolver

54.1.21 Das WholeTextResolver Interface

4.1+

Ein WholeTextResolver ermittelt den Text einer Komponente, der für Checks und ähnliches verwendet werden soll.

Technologien: AWT/Swing, JavaFX, SWT, Web, Windows, Android, iOS

Ein WholeTextResolver muss folgende Methode implementieren:

String getWholeText(Object element, String text)

Ermittelt den ”Gesamttext” einer Komponente

Parameter

element	Das GUI-Element, dessen Text ermittelt werden soll.
text	Der Text, den QF-Test ohne den Resolver verwenden würde.
Rückgabewert	Der ”Gesamttext” oder null, falls der Resolver das Element nicht behandelt. Durch Rückgabe des leeren Strings wird der Komponente kein Text zugeordnet.

Das folgende Beispiel entfernt für Textfelder aus allen für Checks etc. verwendeten Texten den String TO-DO.

```
def getWholeText(element,text):
    if text:
        return text.replace("TO-DO","")
resolvers.addResolver("removeMarkFromText", getWholeText, "TextField", "TextArea")
```

Beispiel 54.22: Ein WholeTextResolver

54.1.22 Der BusyPaneResolver Interfaces

4.1+

QF-Test wartet bei der Testausführung, bis verdeckende BusyPanes verschwinden, um dann in einem determinierten Zustand fortzufahren. Mit einem BusyPaneResolver kann man beeinflussen, ob eine Komponenten QF-Test als verdeckt angesehen wird.

Technologie: AWT/Swing, JavaFX

Ein BusyPaneResolver muss folgende Methode implementieren:

Boolean isBusy(Object element)
Bestimmt, ob eine Komponente aktuell von einer BusyPane oder vergleichbaren Komponente verdeckt wird.

Parameter

element	Das GUI-Element, dessen Zustand bestimmt werden soll.
Rückgabewert	True, wenn aktuell wegen einer BusyPane o.Ä. nicht auf das Element zugegriffen werden kann, false sonst. Null, falls der Resolver das Element nicht behandelt.

Das folgende Beispiel deaktiviert effektiv die Erkennung von BusyPanes für Komponenten des Typs ”my.special.Component”.

```
def isBusy():
    return false
resolvers.addResolver("neverBusyResolver",isBusy,"my.special.Component")
```

Beispiel 54.23: Ein BusyPaneResolver

54.1.23 Der GlassPaneResolver Interfaces

4.1+

Wenn Komponenten von anderen (evtl. transparenten) Komponenten verdeckt werden, so kann man QF-Test mit Hilfe eines GlassPaneResolvers diese Verbindung mitteilen und Events so zur korrekten Komponente umleiten.

Technologie: AWT/Swing

Ein GlassPaneResolver muss folgende Methode implementieren:

Object isGlassPaneFor(Object element, Object target)
Legt die Verbindung zwischen einer überlagernden Komponente und der eigentlichen Zielkomponente fest.

Parameter

element	Die GUI-Komponente, auf dem Events empfangen werden
target	Die GUI-Komponente an die QF-Test die Events ohne Resolver weiterleiten würde
Rückgabewert	Die Komponente, an welche die Events gesendet werden sollen oder null, falls der Resolver das Element nicht behandelt.

Das folgende Beispiel deaktiviert effektiv die Weiterleitung der Events durch GlassPanes:

```
def isGlassPaneFor(element):
    return element
resolvers.addResolver("noGlassPaneResolver", isGlassPaneFor)
```

Beispiel 54.24: Ein `GlassPaneResolver`

54.1.24 Das `TreeIndentationResolver` Interface

8.0+

Ein `TreeIndentationResolver` ermittelt die Einrückung von Baumknoten-Elementen in Bäumen. Verwenden Sie diesen Resolver, wenn QF-Test die Einrückung einzelner Knoten in einer Tree- oder TreeTable-Komponente nicht korrekt erkennt und die Möglichkeiten des Parameters "treeIndentationMode" der CustomWebResolver-Kategorie `treeResolver` nicht ausreichen.

Beachten Sie, dass der Rückgabewert des Resolvers wie eine Pixel-Angabe ausgewertet wird. Insbesondere bedeutet dies, dass unterschiedliche Einrückungsebenen sich standardmäßig um mindestens 2 Pixel unterscheiden müssen.

Technologien: Web

Ein `TreeIndentationResolver` muss folgende Methode implementieren:

`Integer getTreeIndentation(DomNode tree, DomNode treeNode)`
Ermittelt die Einrückung einer TreeNode-Komponente innerhalb eines Trees oder TreeTable.

Parameter

`tree`	Der Tree oder TreeTable der den Baumknoten enthält dessen Einrückung ermittelt werden soll.
`treeNode`	Der Baumknoten dessen Einrückung ermittelt werden soll.
Rückgabewert	Die Einrückung in Pixeln oder `null`, falls der Resolver das Element nicht behandelt.

Das folgende Groovy-Beispiel liest für die Einrückung aller TreeNodes das Attribut `aria-level` aus.

```
Integer getTreeIndentation(Object tree, Object treeNode) {
    def ariaLevel = treeNode.getAttribute('aria-level')
    return ariaLevel ? ariaLevel as Integer * 10 : null
}
resolvers.addResolver("TreeIndentationResolver-Tree", this, "Tree")
```

Beispiel 54.25: Ein `TreeIndentationResolver`

54.1.25 Das EventSynchronizer Interface

4.1+

Wenn QF-Test Events auf dem SUT wiedergibt, bzw. nachdem dies geschehen ist, wartet QF-Test auf die Synchronisation mit dem jeweiligen Event Dispatch Thread. Mit einem EventSynchronizer kann man QF-Test mitteilen, wann das SUT wieder Events entgegen nehmen kann. Dies sollte verwendet werden, wenn im SUT eine eigene Synchronisierung implementiert wurde.

Technologien: AWT/Swing, JavaFX, SWT, Web

Ein EventSynchronizer muss folgende Methode implementieren:

---

void sync(Object context)

Synchronisiert mit dem Event Dispatch Thread des SUT.

Parameter

context	Der Kontext, der bei der Registrierung des Resolvers angegeben wurde.

Das folgende akademische Beispiel hält die Ausführung auf dem Dispatch Thread bis zur nächsten vollen Sekunde an:

```
import time
def sync():
    t = time.time()
    full = int(t)
    delta = t - full
    time.sleep(delta)
resolvers.addResolver("timeSynchronizer",sync)
```

Beispiel 54.26: Ein EventSynchronizer

54.1.26 Das BusyApplicationDetector Interface

4.1+

Mit einem BusyApplicationDetector kann QF-Test erkennen, dass eine Anwendung aktuell ”beschäftigt” ist und keine Events entgegen nehmen kann.

Technologien: AWT/Swing, JavaFX, SWT, Web

Ein BusyApplicationDetector muss folgende Methode implementieren:

Boolean applicationIsBusy

Ermittelt, ob die Anwendung aktuell ”beschäftigt” ist.

Rückgabewert	True, wenn die Anwendung ”beschäftigt” ist, sonst false.

Das folgende Beispiel verwendet eine SUT-spezifische Methode, um QF-Test mitzuteilen, dass es beschäftigt ist:

```
def applicationIsBusy():
    return my.app.App.instance().isDoingDbSynchronization()
resolvers.addResolver("dbAccessDetector",applicationIsBusy)
```

Beispiel 54.27: Ein BusyApplicationDetector

54.1.27 Matcher

Matcher sind keine Resolver im eigentlichen Sinn, da sie nur bei der Wiedergabe greifen. Dennoch werden sie über das resolvers Modul registriert.

Matcher kann man speziell bei der Arbeit mit generischen Komponenten oder bei schlüsselwortgetriebenen Testen einsetzen, wenn keine Aufzeichnungen gemacht werden sollen.

Das ExtraFeatureMatcher Interface

4.1+

Mit einem ExtraFeatureMatcher kann beeinflusst werden, wann ein Weiteres Merkmal, welches für eine Komponente registriert wurde, als "passend" angesehen wird.

Technologien: AWT/Swing, JavaFX, SWT, Web, Windows, Android, iOS

Ein ExtraFeatureMatcher muss folgende Methode implementieren:

Boolean matchExtraFeature(Object element, String name, String value, boolean regexp, boolean negate)

Prüft ein ExtraFeature gegen eine Komponente

Parameter

element	Die GUI-Komponente, für die das Weiteres Merkmal geprüft werden soll.
name	Der Name des Weiteres Merkmal.
value	Der Wert des Weiteres Merkmal.
regexp	True, wenn value ein Regulärer Ausdruck ist (vgl. Abschnitt 49.3(1023)).
negate	True, wenn die Prüfung negiert werden soll.
Rückgabewert	True, falls das Weiteres Merkmal passt, sonst False. Null, falls der Resolver das Element nicht behandelt.

Das folgende Beispiel prüft den Wert des Weiteres Merkmal `my:label` gegen das `my-label` Attribut des HTML-Elementes.

```
import re
def matchExtraFeature(element, name, value, regexp, negate):
    if not name == "my:label":
        return None
    label = element.getAttribute("my-label")
    if label:
        if regexp:
            match = re.match(value,label)
        else:
            match = (value == label)
    else:
        match = False
    return (match and not negate) or (not match and negate)
resolvers.addResolver("myLabelResolver", matchExtraFeature)
```

Beispiel 54.28: Ein `ExtraFeatureMatcher`

Mit Hilfe der speziellen resolvers-Methode `addSpecificExtraFeatureMatcher` kann man den Matcher-Aufruf auch auf einen einzelnen Feature-Namen einschränken:

```
import re
def matchExtraFeature(element, name, value, regexp, negate):
    label = element.getAttribute("my-label")
    if label:
        if regexp:
            match = re.match(value,label)
        else:
            match = (value == label)
    else:
        match = False
    return (match and not negate) or (not match and negate)
resolvers.addSpecificExtraFeatureMatcher("myLabelResolver", \
                                  matchExtraFeature, "my:label")
```

Beispiel 54.29: Nutzung der Methode `addSpecificExtraFeatureMatcher`

54.1.28 Externe Implementierung

Möchte man seine Resolver nicht direkt in einem SUT-Skript implementieren, sondern zum Beispiel in eine JAR-Datei im Plugin-Verzeichnis zur Verfügung stellen, so ist es hilfreich, wenn die Resolver-Klassen direkt die oben aufgeführten Resolver-Interfaces implementieren (Prinzipiell kann ein Resolver auch allein anhand des Namens der implementierten Methode erkannt werden).

Dazu ist bei der Entwicklung die Datei qfsut.jar in den Classpath einzufügen. Die aufgeführten Interfaces befinden sich mehrheitlich im Paket de.qfs.apps.qftest.extensions, bei den Interfaces die zwei Methodenparameter erwarten ist an den eigentlichen Interfacenamen eine ”2” anzufügen. Die Interfaces, welche mit Item... bezeichnet sind, finden sich im Paket de.qfs.apps.qftest.extensions.items. Im SUT-Skript, welches den resolvers.addResolver-Aufruf durchführt, ist dann eine Instanz der selbstentwickelten Resolver-Klasse als Argument mitzugeben.

54.2 Die ResolverRegistry

Alle Arten von Resolvern können mittels resolvers Modul in Jython- oder Groovy-Skripten implementiert werden wie in Abschnitt 54.1(1154) beschrieben. Dies sollte für den normalen Gebrauch ausreichen. Nur wenn Sie im Detail verstehen wollen, wie die ResolverRegistry selbst funktioniert, oder wenn Sie Resolver in Java implementieren möchten, sollten Sie auch die folgenden Abschnitte lesen.

Das vorliegende Kapitel beschreibt, wie Sie Resolver direkt durch Java-Klassen implementieren. Diese Methode führt allerdings dazu, dass ein Teil Ihrer Applikation von den QF-Test Klassen abhängig wird. Auch aus diesem Grund ist es vorzuziehen, die Resolver Interfaces in Jython oder Groovy zu implementieren. Dadurch kann der gesamte Mechanismus strikt vom SUT getrennt bleiben und hat keinerlei Einfluss auf dessen Entwicklungsprozess.

Zusätzlich zu der hier beschriebenen Registrierung der Resolver müssen die entsprechenden Resolver Interfaces implementiert werden. Diese sind ab Abschnitt 54.1.7(1163) ff beschrieben.

Auf Java-Ebene werden Resolver dadurch kompliziert, dass die Methoden als Interface definiert und von einer Klasse implementiert werden müssen. Von dieser Klasse muss eine Instanz erzeugt und bei QF-Test registriert werden. Wenn es nicht auf Anhieb funktioniert, muss die Instanz deregistriert werden, bevor eine neue Klasse erstellt und eine Instanz davon registriert werden kann. Andernfalls könnte es zu Konflikten zwischen den beiden Versionen des Resolvers kommen. Dazu kommt noch Code für die Fehlerbehandlung, insgesamt also ein Vielfaches an Ballast im Vergleich zur eigentlichen Substanz.

Ebenso wie für das resolvers Modul gilt, dass alle Exceptions, die während der Ausführung eines Resolvers auftreten, von der ResolverRegistry abgefangen werden. Es wird allerdings nur eine kurze Meldung und kein Stacktrace ausgegeben, weil insbesondere globale Resolver sehr oft aufgerufen werden können. Somit würde ein Resolver, der einen Bug hat, durch die Ausgabe von Stacktraces für jeden Fehler das Client-Terminal überfluten. Daher sollten Resolver ihre eigenen Fehlerbehandlungsrou-

tinen enthalten. Dabei können zwar immer noch extrem viele Ausgaben erzeugt werden, aber speziell für Skripte sind diese hilfreicher als Java-Stacktraces.

Auch das in Abschnitt 54.1.1(1154) und Abschnitt 54.1.2(1156) Gesagte gilt für die `ResolverRegistry`, sofern es nicht spezifisch ist für Skripte oder das `resolvers` Modul.

Die Singleton Klasse `de.qfs.apps.qftest.extensions.ResolverRegistry` ist die zentrale Agentur für die Registrierung und das Entfernen von Resolvern.

Die API der `ResolverRegistry` enthält keine großen Überraschungen:

`static String getElementName(Object element)`
Diese statische Methode sollte an Stelle von `com.getName()` bzw. `widget.getData()` von Resolvern verwendet werden, die abhängig von bestehenden Namen von Komponenten operieren, mit Ausnahme des `NameResolver2`, der diesen Namen übergeben bekommt. Diese Methode behandelt triviale oder für AWT/Swing spezielle Namen gesondert.

Parameter

`element`	Das GUI Element dessen Name ermittelt werden soll.
Rückgabewert	Der Name der Komponente oder `null`, falls ein trivialer oder spezieller Name unterdrückt wird.

`static ResolverRegistry instance()`
Es gibt immer nur ein einziges `ResolverRegistry` Objekt und diese Methode ist der einzige Weg, Zugriff auf diese Singleton Instanz erlangen.

Rückgabewert	Die `ResolverRegistry` Singleton Instanz.

`static boolean isInstance(Object object, String className)`
Diese statische Methode sollte anstelle von `instanceof` (oder `isinstance()` in Jython) verwendet werden. Sie prüft, ob ein Objekt eine Instanz der gegebenen Klasse ist. Diese Prüfung wird nicht wie üblich über Reflection, sondern auf der Basis von Klassennamen durchgeführt. Dadurch werden Probleme mit verschiedenen ClassLoadern vermieden und die Klasse muss gar nicht erst importiert werden.

Parameter

`object`	Das zu prüfende Objekt.
`className`	Der Name der Klasse auf die getestet wird.
Rückgabewert	True falls das Objekt eine Instanz der Klasse ist.

`void registerExtraFeatureResolver(ExtraFeatureResolver resolver)`
Registriert einen globalen `ExtraFeatureResolver`.

Parameter

`resolver`	Der zu registrierende Resolver.

---

void registerExtraFeatureResolver(Object element, ExtraFeatureResolver resolver)

Registriert einen ExtraFeatureResolver für eine spezifische Komponente. Der Resolver beeinträchtigt nicht die Garbage-Collection und wird automatisch entfernt, wenn die Komponente nicht mehr erreichbar ist.

Parameter

element	Das GUI Element für das registriert wird.
resolver	Der zu registrierende Resolver.

void registerExtraFeatureResolver(String clazz, ExtraFeatureResolver resolver)

Registriert einen ExtraFeatureResolver für eine spezifische Komponenten Klasse.

Parameter

clazz	Der Name der Klasse für die registriert wird.
resolver	Der zu registrierende Resolver.

void registerFeatureResolver2(FeatureResolver2 resolver)

Registriert einen globalen FeatureResolver2.

Parameter

resolver	Der zu registrierende Resolver.

void registerFeatureResolver2(Object element, FeatureResolver2 resolver)

Registriert einen FeatureResolver2 für eine spezifische Komponente. Der Resolver beeinträchtigt nicht die Garbage-Collection und wird automatisch entfernt, wenn die Komponente nicht mehr erreichbar ist.

Parameter

element	Das GUI Element für das registriert wird.
resolver	Der zu registrierende Resolver.

void registerFeatureResolver2(String clazz, FeatureResolver2 resolver)

Registriert einen FeatureResolver2 für eine spezifische Komponenten Klasse.

Parameter

clazz	Der Name der Klasse für die registriert wird.
resolver	Der zu registrierende Resolver.

void registerIdResolver(IdResolver resolver)

Registriert einen globalen IdResolver.

Parameter

resolver	Der zu registrierende Resolver.

`void registerIdResolver(Object element, IdResolver resolver)`
Registriert einen `IdResolver` für eine spezifische Komponente. Der Resolver beeinträchtigt nicht die Garbage-Collection und wird automatisch entfernt, wenn die Komponente nicht mehr erreichbar ist.

Parameter

`element`	Das GUI Element für das registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerIdResolver(String clazz, IdResolver resolver)`
Registriert einen `IdResolver` für eine spezifische Komponenten Klasse.

Parameter

`clazz`	Der Name der Klasse für die registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerNameResolver2(NameResolver2 resolver)`
Registriert einen globalen `NameResolver2`.

Parameter

`resolver`	Der zu registrierende Resolver.

`void registerNameResolver2(Object element, NameResolver2 resolver)`
Registriert einen `NameResolver2` für eine spezifische Komponente. Der Resolver beeinträchtigt nicht die Garbage-Collection und wird automatisch entfernt, wenn die Komponente nicht mehr erreichbar ist.

Parameter

`element`	Das GUI Element für das registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerNameResolver2(String clazz, NameResolver2 resolver)`
Registriert einen `NameResolver2` für eine spezifische Komponenten Klasse.

Parameter

`clazz`	Der Name der Klasse für die registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerTreeTableResolver(TreeTableResolver resolver)`
Registriert einen globalen `TreeTableResolver`.

Parameter

`resolver`	Der zu registrierende Resolver.

`void registerTreeTableResolver(Object com, TreeTableResolver resolver)`

Registriert einen `TreeTableResolver` für eine spezifische Komponente. Der Resolver beeinträchtigt nicht die Garbage-Collection und wird automatisch entfernt, wenn die Komponente nicht mehr erreichbar ist.

Parameter

`com`	Das GUI Element für das registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerTreeTableResolver(String clazz, TreeTableResolver resolver)`

Registriert einen `TreeTableResolver` für eine spezifische Komponenten Klasse.

Parameter

`clazz`	Der Name der Klasse für die registriert wird.
`resolver`	Der zu registrierende Resolver.

`void unregisterExtraFeatureResolver(ExtraFeatureResolver resolver)`

Entfernt einen globalen `ExtraFeatureResolver`.

Parameter

`resolver`	Der zu entfernende Resolver.

`void unregisterExtraFeatureResolver(Object element, ExtraFeatureResolver resolver)`

Entfernt einen `ExtraFeatureResolver` für eine spezifische Komponente.

Parameter

`element`	Das GUI Element für das entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterExtraFeatureResolver(String clazz, ExtraFeatureResolver resolver)`

Entfernt einen `ExtraFeatureResolver` für eine spezifische Komponenten Klasse.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterFeatureResolver2(FeatureResolver2 resolver)`

Entfernt einen globalen `FeatureResolver2`.

Parameter

`resolver`	Der zu entfernende Resolver.

`void unregisterFeatureResolver2(Object element, FeatureResolver2 resolver)`

Entfernt einen `FeatureResolver2` für eine spezifische Komponente.

Parameter

`element`	Das GUI Element für das entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterFeatureResolver2(String clazz, FeatureResolver2 resolver)`

Entfernt einen `FeatureResolver2` für eine spezifische Komponenten Klasse.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterIdResolver(IdResolver resolver)`

Entfernt einen globalen `IdResolver`.

Parameter

`resolver`	Der zu entfernende Resolver.

`void unregisterIdResolver(Object element, IdResolver resolver)`

Entfernt einen `IdResolver` für eine spezifische Komponente.

Parameter

`element`	Das GUI Element für das entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterIdResolver(String clazz, IdResolver resolver)`

Entfernt einen `IdResolver` für eine spezifische Komponenten Klasse.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterNameResolver2(NameResolver2 resolver)`

Entfernt einen globalen `NameResolver2`.

Parameter

`resolver`	Der zu entfernende Resolver.

`void unregisterNameResolver2(Object element, NameResolver2 resolver)`

Entfernt einen `NameResolver2` für eine spezifische Komponente.

Parameter

`element`	Das GUI Element für das entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterNameResolver2(String clazz, NameResolver2 resolver)`

Entfernt einen `NameResolver2` für eine spezifische Komponenten Klasse.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterResolvers(Object element)`

Entfernt alle Resolver für eine spezifische Komponente.

Parameter

`element`	Das GUI Element für das entfernt wird.

`void unregisterResolvers(String clazz)`

Entfernt alle Resolver für eine Komponenten-Klasse.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.

`void unregisterTreeTableResolver(TreeTableResolver resolver)`

Entfernt einen globalen `TreeTableResolver`.

Parameter

`resolver`	Der zu entfernende Resolver.

`void unregisterTreeTableResolver(Object com, TreeTableResolver resolver)`

Entfernt einen `TreeTableResolver` für eine spezifische Komponente.

Parameter

`com`	Das GUI Element für das entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterTreeTableResolver(String clazz, TreeTableResolver resolver)`

Entfernt einen `TreeTableResolver` für eine spezifische Komponenten Klasse.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.
`resolver`	Der zu entfernende Resolver.

54.3 Implementierung eigener Unterelemente mit dem ItemResolver Interface

3.1+

Wie in Abschnitt 5.9(92) beschrieben, ist QF-Test in der Lage, jenseits der Struktur von GUI Elementen mit Unterelementen zu arbeiten, die selbst keine GUI Elemente sind, z.B. die Zellen einer Tabelle, Knoten in einem Baum oder Zeichnungen auf einem Canvas. Solche Unterelemente werden mit Hilfe des `ItemResolver` Mechanismus implementiert, über den Sie auch Ihre eigenen spezifischen Unterelemente erstellen können.

Das `ItemResolver` Interface ist deutlich komplexer als die einfachen `NameResolver2` oder `FeatureResolver2` Interfaces aus dem vorhergehenden Abschnitt und es kann nicht ohne gute Programmierkenntnisse und Verständnis der zu Grunde liegenden Konzepte implementiert werden. Außerdem sind `ItemResolver` und die im nächsten Abschnitt beschriebenen `Checker` eng miteinander verknüpft und sollten gemeinsam implementiert werden, wenn Sie Ihre Unterelemente auch überprüfen können wollen.

Lassen Sie sich davon aber nicht abschrecken, denn andererseits ist dieser Mechanismus sehr mächtig und wenn Ihre `ItemResolver` erst einmal implementiert und registriert sind, integrieren sie sich so nahtlos in QF-Test, dass kein Unterschied zwischen standard und nicht-standard Unterelementen zu erkennen ist.

Sie finden im Verzeichnis `qftest-9.0.0/Jython/Lib` unter QF-Tests Wurzelverzeichnis einige Beispielimplementierungen, wie `ktable.py` oder `gef.py`. Beide Resolver sind zwar für SWT spezifische Tabellen, das Konzept allerdings ist bei allen Engines das gleiche.

54.3.1 ItemResolver Konzepte

Bevor Sie mit der Umsetzung eines `ItemResolvers` beginnen können, müssen Sie sich über die Art der Unterelemente klar werden, die Ihr GUI Element enthalten kann. Es könnte mehr als eine Art geben, so dass diese Entscheidung willkürlich ist. So haben wir z.B. Unterelemente für Swing, SWT und JavaFX Tabellen so implementiert, dass sowohl einzelne Zellen, als auch ganze Spalten als Unterelemente unterstützt werden. Letzteres ist sehr hilfreich, weil damit Checks für komplette Spalten möglich sind.

Als nächstes müssen Sie entscheiden, wie Sie die Unterelemente intern darstellen wollen. Sie können jede Art von `Object` verwenden, da QF-Test niemals auf Ihre interne Repräsentation zugreift, sondern diese lediglich zwischen den Methoden Ihres `ItemResolvers` hin und her reicht. Was am besten funktioniert, hängt von der API des GUI Elements ab. Wenn dieses bereits ein Konzept für Unterelemente bietet, ist es vermutlich am besten, diese Klassen zu verwenden.

Die wichtigste Entscheidung ist, wie Sie ein Unterelement gegenüber QF-Test reprä-

sentieren. Wie in Abschnitt 5.9(92) beschrieben, kann der Anwender ein Unterelement über einen numerischen oder einen textuellen Index adressieren, wobei letzterer auch ein regulärer Ausdruck (vgl. Abschnitt 49.3(1023)) sein kann. Sie müssen also zwischen einem Unterelement und seinem Index (oder seinen Indizes) eine bidirektionale Abbildung ermöglichen, d.h. Sie müssen folgende Fragen beantworten können:

- Was ist der numerische und was der textuelle Index für ein gegebenes Unterelement?
- Welches Unterelement passt am besten zu einem gegebenen numerischen oder textuellen Index?

Die Kernpunkte bei der Vergabe von Namen für Unterelemente sind die gleichen wie bei normalen GUI Elementen. Arbeiten Sie hierzu bitte Abschnitt 5.4.2(66) und Abschnitt 5.4.2(68) gründlich durch bevor Sie fortfahren.

Ein einzelner numerischer oder textueller Index wird durch ein `SubItemIndex` Objekt repräsentiert. Das aktuelle Konzept unterstützt die Adressierung eines Unterelements mittels eines primären und eines sekundären Index. In der Zukunft hoffen wir, Indizes mit beliebiger Tiefe behandeln zu können, so dass ein Knoten in einem Baum statt über einen einzelnen Pfad auch über einen gemischten Index wie ”tree@Root&1%Name:.*” angesprochen werden könnte. Daher ist der gesamte Index ein Array von `SubItemIndex` Objekten, auch wenn dessen Länge aktuell auf ein oder zwei Objekte beschränkt ist.

Die meisten Unterelemente haben eine Geometrie, also eine Position und eine Größe. Die Koordinaten eines Unterelements beziehen sich immer auf die obere linke Ecke des GUI Elements, zu dem sie gehören, unabhängig davon, ob dieses gescrollt wird. Dadurch hängen die Koordinaten nicht von der aktuellen Scrollposition ab. Für Unterelemente, für die Geometrie keinen Sinn macht oder nicht ermittelt werden kann, können Koordinaten ignoriert werden und die Methoden `getItemLocation` und `getItemSize` sollten einfach [0,0] zurückliefern.

54.3.2 Das `ItemResolver` Interface

Die Methoden des Interface `de.qfs.apps.qftest.extensions.items.ItemResolver` teilen sich in drei Kategorien auf: Erkennen eines Unterelements, die Beziehung zwischen einem Unterelement und seinem Index und dem Auslesen von Informationen aus bzw. Ausführen von Aktionen auf einem Unterelement.

`Object getItem(Object element, int x, int y)`
Liefert ein Unterelement für ein GUI Element an einer bestimmten Position. Für Unterelemente ohne Geometrie können die Koordinaten ignoriert und das Unterelement aufgrund von anderen Eigenschaften bestimmt werden, z.B. anhand der Selektion.

Parameter

`element`	Das GUI Element, dessen Unterelement bestimmt werden soll.
`x`	Die X Koordinate relativ zum GUI Element.
`y`	Die Y Koordinate relativ zum GUI Element.
Rückgabewert	Ein beliebiges Objekt, welches das Unterelement repräsentiert, oder null falls es kein Unterelement an der angegebenen Stelle gibt.

`int getItemCount(Object element, Object item)`
Liefert die Anzahl der Unterelemente des GUI Elements auf der jeweils nächsten Ebene. Diese Methode wird aktuell zwar nicht verwendet, sollte aber wenn möglich korrekt implementiert werden. In der Zukunft könnte dies z.B. für einen 'Anzahl auslesen'-Knoten analog zu Index auslesen(843) oder Text auslesen(839) genutzt werden.

Parameter

`element`	Das GUI Element, für das die Zahl der Unterelemente ermittelt werden soll.
`item`	Null um die Anzahl der Unterelemente auf oberster Ebene zu erhalten, ein Unterelement um die Anzahl von dessen Unterelementen der nächsten Ebene zu ermitteln.
Rückgabewert	Die Anzahl von Unterelementen oder -1 falls es keine weitere Ebene gibt.

`Object getItemForIndex(Object element, SubItemIndex[] idx)`
Liefert ein Unterelement zu einem gegebenen Index.
Am Ende dieser Prozedur ist ggf. ein Aufruf von `setIndexesResolved` notwendig, falls mehr als ein Index aufgelöst wurde.

Parameter

`element`	Das GUI Element, dessen Unterelement bestimmt werden soll.
`idx`	Der Index oder die Indizes für das Unterelement.
Rückgabewert	Das Unterelement, das dem Index am besten entspricht.

Exceptions

`IndexNotFoundException`	Falls kein Unterelement zum angegebenen Index passt. Verwenden Sie für diesen Fall den Konstruktor `de.qfs.apps.qftest.shared.exceptions.IndexNotFoundException(SubItemIndex)`.

`SubItemIndex[] getItemIndex(Object element, Object item, int type)`

Liefert den/die `SubItemIndex(es)` für ein Unterelement eines GUI Elements.

Parameter

`element`	Das GUI Element, zu dem das Unterelement gehört.
`item`	Das Unterelement dessen Index ermittelt werden soll.
`type`	Die Art des zu bestimmenden Index. Mögliche Werte sind `INTELLIGENT`, `AS_STRING` und `AS_NUMBER`, alle in der Klasse `SubItemIndex` definiert. Sofern nicht nur eine Art von Index unterstützt wird, sollte ein textueller Index für `AS_STRING` und ein numerischer Index für `AS_NUMBER` geliefert werden. Beim Typ `INTELLIGENT` können Sie frei entscheiden, welcher Art von Index das Unterelement am besten repräsentiert. Auch ein gemischter Index ist möglich, z.B. ein Spaltentitel plus numerischer Zeilenindex für eine Zelle in einer Tabelle.
Rückgabewert	Ein Array von `SutItemIndex` Objekten. Aktuell sind nur Arrays mit einem oder zwei Elementen erlaubt.

`int[] getItemLocation(Object element, Object item)`

Liefert die Position eines Unterelements relativ zum GUI Element.

Parameter

`element`	Das GUI Element, zu dem das Unterelement gehört.
`item`	Das Unterelement dessen Position ermittelt werden soll.
Rückgabewert	Die Position des Unterelements als int Array der Form [X,Y]. Die Position muss immer relativ zur oberen linken Ecke des GUI Elements sein, auch wenn diese Ecke aktuell nicht sichtbar ist, z.B. wenn das GUI Element gescrollt wird. Für Unterelemente ohne Geometrie liefern Sie einfach [0,0].

`int[] getItemSize(Object element, Object item)`

Liefert die Größe eines Unterelements relativ zum GUI Element.

Parameter

`element`	Das GUI Element, zu dem das Unterelement gehört.
`item`	Das Unterelement dessen Größe ermittelt werden soll.
Rückgabewert	Die Größe des Unterelements als int Array der Form [Breite,Höhe]. Für Unterelemente ohne Geometrie liefern Sie einfach [0,0].

`String getItemValue(Object element, Object item)`

Liefert den Wert eines Unterelements für einen Text-Check.

Parameter

`element`	Das GUI Element, zu dem das Unterelement gehört.
`item`	Das Unterelement dessen Wert ermittelt werden soll.
Rückgabewert	Ein String entsprechend dem Wert des Unterelements, sein Inhalt, Label, etc.

`Boolean repositionMouseEvent(Object element, Object item, int[] pos)`

Ändert die Koordinaten eines Mausevents auf ein Unterelement eines GUI Elements zu einer standard Position, üblicherweise die Mitte der Komponente, sofern die Koordinaten bei der Wiedergabe keine Rolle spielen. Diese Methode wird nur aufgerufen, wenn die Option Mausevents ohne Koordinaten aufnehmen wo möglich (510) aktiviert ist. Ob es sicher ist, die Koordinaten zu überschreiben, kann von den ursprünglichen Koordinaten abhängen. So ändert QF-Test z.B. die Koordinaten für Events auf einen Baumknoten ab, sofern diese positiv sind und damit in den Knoten zeigen. Negative Koordinaten könnten einen Klick auf den Schalter zum Ein-/Ausklappen bedeuten und bleiben unverändert.

Parameter

`element`	Das GUI Element, zu dem das Unterelement gehört.
`item`	Das Ziel-Unterelement für den Event.
`pos`	Die Koordinaten des Events als int Array der Form [X,Y]. Die Werte können direkt im Array manipuliert werden. Sie können entweder gezielt Koordinaten für einen bestimmten Punkt angeben oder [Integer.MAX_VALUE,Integer.MAX_VALUE] um die Koordinaten zu ignorieren, so dass der Event später auf die Mitte abgespielt wird.
Rückgabewert	Boolean.TRUE falls die Position geändert wurde, Boolean.FALSE falls die Position nicht geändert wurde. Null um zu signalisieren, dass dieser Resolver das Element nicht behandelt.

Exceptions

`BadItemException`	Falls das Element und der Item Typ nicht zusammen passen (sollte niemals passieren).

`Boolean scrollItemVisible(Object element, Object item, int x, int y)`
Scrollt ein GUI Element so, dass ein Unterelement sichtbar wird. Wenn möglich sollte das Unterelement komplett sichtbar gemacht werden. Falls es nicht in den sichtbaren Bereich passt, sollte zumindest die angegebene Position dargestellt werden. In den meisten Fällen können Sie einfach null zurückgeben und QF-Test das Scrollen selbst überlassen. Manchmal kann Scrollen allerdings nicht generisch implementiert werden, z.B. bei SWT Tabellen mit sichtbarem Header. Für GUI Elemente, die Scrollen gar nicht unterstützen, liefern Sie einfach Boolean.FALSE. Wird Boolean.TRUE geliefert, ruft QF-Test die Methode nach kurzer Zeit ein weiteres mal auf, da SWT nicht immer die gewünschte Position übernimmt. Im Idealfall sollte die Position dann stimmen und FALSE geliefert werden. Nach drei Versuchen mit TRUE gibt QF-Test aus und melden einen Fehler.

Parameter

`element`	Das GUI Element, zu dem das Unterelement gehört.
`item`	Das Unterelement, das sichtbar gemacht werden soll.
`x`	Die X-Koordinate relativ zum Unterelement, die in jedem Fall sichtbar sein muss.
`y`	Die Y-Koordinate relativ zum Unterelement, die in jedem Fall sichtbar sein muss.
Rückgabewert	Boolean.TRUE falls die Scroll-Position des GUI Elements verändert wurde, Boolean.FALSE falls die Scroll-Position unverändert ist oder nicht gescrollt werden kann. Null um zu signalisieren, dass QF-Test das GUI Element selbst scrollen soll.

`void setIndexesResolved(int num)`
Informiert die Registry über die Anzahl der während der Indexbestimmung in `getItemForIndex` aufgelösten Indizes. Wird diese Methode am Ende von `getItemForIndex` nicht aufgerufen, geht die Registry von einem Index aus.

Parameter

`num`	Die Anzahl der aufgelösten Indizes.

54.3.3 Die Klasse `SubItemIndex`

Wie im vorhergehenden Abschnitt erklärt, repräsentiert ein `de.qfs.apps.qftest.shared.data.SubItemIndex` einen (partiellen) Index für ein Unterelement eines GUI Elements. Diese Klasse definiert einige Konstanten mit folgender Bedeutung:

STRING
Dies ist ein textueller Index

NUMBER
Dies ist ein numerischer Index

REGEXP
Dies ist ein regulärer Ausdruck (vgl. Abschnitt 49.3(1023)), passend zu einem textuellen Index

INTELLIGENT
Ermittle den Typ von Index, der am besten zum Unterelement passt

AS_STRING
Ermittle einen textuellen Index

AS_NUMBER
Ermittle einen numerischen Index

Sie stellt außerdem folgende Methoden bereit:

`SubItemIndex SubItemIndex(String index)`

Erstellt einen neuen `SubItemIndex` vom Typ `STRING`.

Parameter

`index`	Der textuelle Index.

`SubItemIndex SubItemIndex(int index)`

Erstellt einen neuen `SubItemIndex` vom Typ `NUMBER`.

Parameter

`index`	Der numerische Index.

`int asNumber()`

Liefert den Index als Zahl.

Rückgabewert	Der numerische Index.

Exceptions

`IndexFormatException`	Falls der Index nicht vom Typ `NUMBER` ist oder nicht als Integer geparst werden kann.

`String getIndex()`

Liefert den Index als String.

Rückgabewert	Der Index, konvertiert in einen String.

`String getType()`

Liefert den Typ des Index.

Rückgabewert	Der Typ des Index, `STRING`, `NUMBER` oder `REGEXP`.

`boolean matches(String name)`

Prüft, ob der Index zum Namen eines Unterelements passt.

Parameter

`name`	Der zu prüfende Name.
Rückgabewert	True falls der Index nicht numerisch ist und zum angegebenen Namen passt.

Exceptions

`IndexFormatException`	Falls der Index einen ungültigen regulären Ausdruck enthält.

54.3.4 Die `ItemRegistry`

Wenn Ihr `ItemResolver` implementiert und instanziiert ist, muss er bei der `ItemRegistry` registriert werden. Die Klasse `de.qfs.apps.qftest.extensions.items.ItemRegistry` bietet hierzu folgende Methoden:

`static ItemRegistry instance()`

Es gibt immer nur ein einziges `ItemRegistry` Objekt und diese Methode ist der einzige Weg, Zugriff auf diese Singleton Instanz erlangen.

Rückgabewert	Die `ItemRegistry` Singleton Instanz.

`void registerItemNameResolver2(Object element, ItemNameResolver2 resolver)`

Registriert einen `ItemNameResolver2` für ein spezifisches GUI Element. Der Resolver beeinträchtigt nicht die Garbage-Collection und wird automatisch entfernt, wenn die Komponente nicht mehr erreichbar ist.

Parameter

`element`	Das GUI Element für das registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerItemNameResolver2(String clazz, ItemNameResolver2 resolver)`

Registriert einen `ItemNameResolver2` für eine spezifische Klasse von GUI Elementen.

Parameter

`clazz`	Der Name der Klasse für die registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerItemResolver(Object element, ItemResolver resolver)`

Registriert einen `ItemResolver` für ein spezifisches GUI Element. Der Resolver beeinträchtigt nicht die Garbage-Collection und wird automatisch entfernt, wenn die Komponente nicht mehr erreichbar ist.

Parameter

`element`	Das GUI Element für das registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerItemResolver(String clazz, ItemResolver resolver)`

Registriert einen `ItemResolver` für eine spezifische Klasse von GUI Elementen.

Parameter

`clazz`	Der Name der Klasse für die registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerItemValueResolver2(Object element, ItemValueResolver2 resolver)`

Registriert einen `ItemValueResolver2` für ein spezifisches GUI Element. Der Resolver beeinträchtigt nicht die Garbage-Collection und wird automatisch entfernt, wenn die Komponente nicht mehr erreichbar ist.

Parameter

`element`	Das GUI Element für das registriert wird.
`resolver`	Der zu registrierende Resolver.

`void registerItemValueResolver2(String clazz, ItemValueResolver2 resolver)`

Registriert einen `ItemValueResolver2` für eine spezifische Klasse von GUI Elementen.

Parameter

`clazz`	Der Name der Klasse für die registriert wird.
`resolver`	Der zu registrierende Resolver.

`void unregisterItemNameResolver2(Object element, ItemNameResolver2 resolver)`

Entfernt einen `ItemNameResolver2` für ein spezifisches GUI Element.

Parameter

`element`	Das GUI Element für das entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterItemNameResolver2(String clazz, ItemNameResolver2 resolver)`

Entfernt einen `ItemNameResolver2` für eine spezifische Klasse von GUI Elementen.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterItemResolver(Object element, ItemResolver resolver)`

Entfernt einen `ItemResolver` für ein spezifisches GUI Element.

Parameter

`element`	Das GUI Element für das entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterItemResolver(String clazz, ItemResolver resolver)`

Entfernt einen `ItemResolver` für eine spezifische Klasse von GUI Elementen.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterItemValueResolver2(Object element, ItemValueResolver2 resolver)`

Entfernt einen `ItemValueResolver2` für ein spezifisches GUI Element.

Parameter

`element`	Das GUI Element für das entfernt wird.
`resolver`	Der zu entfernende Resolver.

`void unregisterItemValueResolver2(String clazz, ItemValueResolver2 resolver)`

Entfernt einen `ItemValueResolver2` für eine spezifische Klasse von GUI Elementen.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.
`resolver`	Der zu entfernende Resolver.

54.3.5 Standard Repräsentation von Unterelementen

Für die Implementierung der Interfaces `ItemNameResolver2`, `ItemValueResolver2` und `Checker` ist es wichtig, die Art der Objekte zu kennen, die ein Unterelement intern repräsentieren, denn dies sind die Objekte, die

an die Methoden `getItemName`, `getItemValue`, `getCheckData` und `getCheckDataAndItem` übergeben werden.

JavaFX

Die folgende Tabelle führt die komplexen Komponenten von JavaFX und die von QF-Test's standard `ItemResolvern` verwendete Repräsentation der jeweiligen Unterelemente auf.

GUI element class	Item type
`Accordion`	`Integer` Index
`ChoiceBox`	`Integer` Index
`ComboBox`	`Integer` Index
`ListView`	`Integer` Index
`TabPane`	`Integer` Index
`TableView`	`int` Array [column,row] wobei row < 0 eine ganze Spalte repräsentiert
`TableHeaderRow`	`Integer` Spaltenindex
`TextArea`	`Integer` Zeile
`TreeView`	`TreeItem` Objekt

Tabelle 54.1: Interne Repräsentation für Unterelement von JavaFX Komponenten

Swing

Die folgende Tabelle führt die komplexen Komponenten von Swing und die von QF-Test's standard `ItemResolvern` verwendete Repräsentation der jeweiligen Unterelemente auf.

GUI Element Klasse	Art des Unterelements
`JComboBox`	`Integer` Index
`JList`	`Integer` Index
`JTabbedPane`	`Integer` Index
`JTable`	`int` Array [column,row] wobei row < 0 eine ganze Spalte repräsentiert
`JTableHeader`	`Integer` Spaltenindex
`JTextArea`	`Integer` Zeile
`JTree`	`TreePath` Pfad

Tabelle 54.2: Interne Repräsentation für Unterelement von Swing Komponenten

SWT

Die folgende Tabelle führt die komplexen GUI Elemente von SWT und die von QF-Test's standard `ItemResolvern` verwendete Repräsentation der jeweiligen Unterelemente auf.

GUI Element Klasse	Art des Unterelements
CCombo	Integer Index
Combo	Integer Index
CTabFolder	Integer Index
List	Integer Index
StyledText	Integer Zeile
TabFolder	Integer Index
Table	int Array [Spalte,Zeile] oder nur Integer um eine Spalte zu repräsentieren
Text	Integer Zeile
Tree	Object Array [Integer Spalte,TreeItem Zeile] oder nur Integer um eine Spalte zu repräsentieren

Tabelle 54.3: Interne Repräsentation für Unterelement von SWT GUI Elementen

Web

Die folgende Tabelle führt die komplexen GUI Elemente für Web und die von QF-Test's standard ItemResolvern verwendete Repräsentation der jeweiligen Unterelemente auf.

GUI Element Klasse	Art des Unterelements
SELECT Knoten	OPTION Knoten
TEXTAREA Knoten	Integer Zeile

Tabelle 54.4: Interne Repräsentation für Unterelement von Web GUI Elementen

54.4 Implementierung eigener Checks mit dem Checker Interface

3.1+

Checks gehören zu QF-Test's nützlichsten Features. Ohne die Fähigkeit, die Ergebnisse von simulierten Aktionen zu verifizieren, wäre Testautomatisierung weitgehend nutzlos. Allerdings beschränkt sich das Repertoire von Checks in QF-Test natürlich auf die gängigen Attribute der standard Komponenten. Für besondere, selten genutzte Attribute oder für eigene Komponenten können Sie auf SUT-Skript(720) Knoten ausweichen, dort den benötigten Wert auslesen und über die Methode rc.checkEqual() mit dem erwarteten Wert vergleichen. So ein SUT-Skript Knoten funktioniert schnell und zuverlässig und lässt sich in einer Prozedur(672) modularisieren. Er hat aber zwei Nachteile: Er lässt sich nicht aufzeichnen und er ist abschreckend für Nicht-Entwickler.

Mit Hilfe der in diesem Abschnitt beschriebenen API ist es möglich, die Standardchecks

von QF-Test zu erweitern. QF-Test's eigene Checks basieren sogar selbst darauf. Durch Implementieren und Registrieren eines `Checkers` für eine Klasse von GUI Elementen können Sie Ihre eigenen Checks erstellen, die genauso aufgenommen und wiedergegeben werden können wie die Standardchecks.

Um dies so einfach wie möglich zu gestalten kümmert sich QF-Test selbst um alle Details, von der Darstellung im Check-Popupmenü, Abholen der Checkdaten, Aufnahme des entsprechenden Check Knotens um die Daten zu speichern, Schicken der Daten an das SUT zur Wiedergabe, Abholen der dann gültigen Checkdaten bis hin zum Vergleichen der beiden Werte und Darstellung der Ergebnisse im Protokoll. Alles was Sie dazu beitragen müssen, ist QF-Test mitzuteilen, welche Checks Ihr `Checker` implementiert und für diese auf Anfrage die Checkdaten zu ermitteln.

Illustrative Beispiele finden Sie am Ende des Kapitels sowie in der Testsuite `carconfigSwing_de.qft` im Verzeichnis `demo/carconfigSwing` Ihrer QF-Test Installation.

54.4.1 Das `Checker` Interface

Das Interface `de.qfs.apps.qftest.extensions.checks.Checker` muss implementiert werden, um eigene Checks für Ihre Anwendung bereitzustellen. Die damit verbundenen Hilfsklassen und Interfaces werden in den folgenden Abschnitten beschrieben.

`CheckData getCheckData(Object element, Object item, CheckType type)`
Liefert die Checkdaten für den aktuellen Zustand eines GUI Elements oder Unterelements.

Parameter

`element`	Das GUI Element, für das die Checkdaten ermittelt werden sollen.
`item`	Ein optionales Unterelement, das geprüft werden soll. Sein Typ hängt von der Art des GUI Elements und der dafür registrierten `ItemResolver` ab, wie in Abschnitt 54.3.5(1206) beschrieben.
`type`	Die Art des auszuführenden Checks.
Rückgabewert	Die Checkdaten für den aktuellen Zustand des GUI Elements selbst, falls `item` null ist, oder des Unterelements. Der gewünschte Checktyp wird über den Parameter `type` definiert, der normalerweise einer der vorher von `getSupportedCheckTypes` zurückgelieferten Checktypen sein sollte. Falls Sie den gewünschten Check für das angegebene Ziel nicht ausführen können, liefern Sie null.

`Pair getCheckDataAndItem(Object element, Object item, CheckType type)`
Liefert die Checkdaten für den aktuellen Zustand eines GUI Elements oder Unterelements und zusätzlich das Unterelement, auf das sich der Check bezieht. Diese Methode wird bei der Aufnahme aufgerufen, wo QF-Test nicht weiß, ob sich die aufgenommenen Daten nun auf das GUI Element selbst oder ein Unterelement beziehen. Um diese Methode ohne Duplizierung von Code zu implementieren, sollten Sie zum Ermitteln der Daten Ihre eigene `getCheckData` Methode aufrufen.

Parameter

`element`	Das GUI Element, für das die Checkdaten ermittelt werden sollen.
`item`	Ein optionales Unterelement, das geprüft werden soll. Sein Typ hängt von der Art des GUI Elements und der dafür registrierten `ItemResolver` ab, wie in Abschnitt 54.3.5[1206] beschrieben.
`type`	Die Art des auszuführenden Checks.
Rückgabewert	Ein `Pair` bestehend aus den Checkdaten für das GUI Element oder Unterelement sowie das Unterelement, auf das sich der Check bezieht. Letzteres kann null sein.

`CheckType[] getSupportedCheckTypes(Object element, Object item)`
Liefert die Arten von Checks, die für ein GUI Element und ein optionales Unterelement unterstützt werden.

Parameter

`element`	Das GUI Element, für das die möglichen Checks ermittelt werden.
`item`	Ein optionales Unterelement, das geprüft werden soll. Sein Typ hängt von der Art des GUI Elements und der dafür registrierten `ItemResolver` ab, wie in Abschnitt 54.3.5[1206] beschrieben.
Rückgabewert	Ein Array mit den `CheckType` Objekten, die Ihr `Checker` unterstützt. Das erste Element ist der Standard für die Check-Aufnahme mit einem Linksklick. Falls `item` null ist, liefern Sie nur die Checks zurück, die sich auf das GUI Element als ganzes beziehen. Ansonsten ist es am besten, alle verfügbaren Checks mit oder ohne Unterelement zu liefern, da der Anwender zwar vielleicht auf ein Unterelement geklickt hat, aber trotzdem einen Check für das ganze GUI Element aufnehmen möchte.

54.4.2 Die `Pair` Klasse

Die Klasse `de.qfs.lib.util.Pair` für den Rückgabewert von `getCheckDataAndItem` ist eine einfache Hilfsklasse, die praktisch für die Gruppierung von zwei Objekten ist. Sie müssen nur ihren Konstruktor kennen, können aber natürlich auch ihre Werte auslesen:

`Pair Pair(Object first, Object second)`

Erstellt ein neues `Pair`.

Parameter

`first`	Das erste Objekt. Kann null sein.
`second`	Das zweite Objekt Kann null sein.

`Object getFirst()`

Liefert das erste Objekt des `Pair`.

Rückgabewert	Das erste Objekt.

`Object getSecond()`

Liefert das zweite Objekt des `Pair`.

Rückgabewert	Das zweite Objekt.

54.4.3 Das `CheckType` Interface und seine Implementierung `DefaultCheckType`

Ein `de.qfs.apps.qftest.extensions.checks.CheckType` kapselt die Definition einer spezifischen Art von Check. Er kombiniert einen `CheckDataType` mit einer Bezeichnung und stellt eine anwenderfreundliche Repräsentation für das Checkmenü bereit. Sofern Sie nicht mehrsprachige Darstellungen im Checkmenü benötigen, sollten Sie dieses Interface nie selbst implementieren, sondern einfach einen `de.qfs.apps.qftest.extensions.checks.DefaultCheckType` instanziieren:

`DefaultCheckType(String identifier, CheckDataType dataType, String description)`

Erzeugt einen neuen `DefaultCheckType`.

Parameter

`identifier`	Der Bezeichner des Checks. Die Standardchecks von QF-Test benutzen hierfür ausschließlich Kleinbuchstaben. Um Konflikte zu vermeiden, beginnen Sie Ihre Bezeichner einfach groß.
`dataType`	Der `CheckDataType` für Ihre Checkdaten.
`description`	Die Beschreibung des Checks für das Checkmenü.

Der Vollständigkeit halber führen wir auch die Methoden des `CheckType` Interfaces auf:

`CheckDataType getDataType()`

Liefert den `CheckDataType` für den Check.

Rückgabewert	Der Datentyp des Checks.

`String getDescription()`

Liefert die lokalisierte Beschreibung für diesen Checktyp im Checkmenü.

Rückgabewert	Die Beschreibung für den Checktyp.

`String getIdentifier()`

Liefert den Bezeichner des Checktyps.

Rückgabewert	Der Bezeichner des Checktyps.

54.4.4 Die Klasse `CheckDataType`

Die Klasse `de.qfs.apps.qftest.extensions.checks.CheckDataType` ist vergleichbar zu `Enum`. Sie definiert einige Konstanten von `CheckDataType` Instanzen, die dazu dienen, die Art der Daten zu definieren, auf denen ein Check beruht. Jede Konstante entspricht dabei einem der verfügbaren 'Check'-Knoten von QF-Test.

Außer seiner Funktion als Identifikator hat ein `CheckDataType` keine public Attribute oder Methoden und Sie können keine neuen `CheckDataType` Objekte erstellen. Wenn Sie einen Check implementieren wollen, der nicht zu den verfügbaren Datentypen passt, müssen Sie Ihre Daten entsprechend konvertieren, z.B. in einen String. Folgende `CheckDataType` Konstanten sind definiert:

STRING
: Ein einzelner String. Entspricht dem Check Text(806) Knoten.

STRING_LIST
Eine Liste von Strings, wie die Zellen einer Tabellenspalte. Entspricht dem Check Elemente(818) Knoten.

SELECTABLE_STRING_LIST
Eine Liste von selektierbaren Strings, wie die Elemente einer Liste. Entspricht dem Check selektierbare Elemente(823) Knoten.

BOOLEAN
Ein Boolean Zustand, entweder true oder false. Entspricht dem Check Boolean(812) Knoten.

GEOMETRY
Ein Satz von 4 Integer Werten für X und Y-Koordinaten, Breite und Höhe. Nicht alle müssen definiert sein. Entspricht dem Check Geometrie(834) Knoten.

IMAGE
Ein Abbild einer ganzen Komponente, eines Unterelements oder einer Region darin. Entspricht dem Check Abbild(828) Knoten.

54.4.5 Die Klasse `CheckData` und ihre Unterklassen

Die Klasse `de.qfs.apps.qftest.shared.data.check.CheckData` und ihre Unterklassen, alle aus demselben Package, komplettieren die `Checker`-API. Ein `CheckData` Objekt kapselt die eigentlichen Daten für einen Check und muss von der Methode `Checker.getCheckData()` zurückgeliefert werden. Hiermit werden die Daten zwischen QF-Test und dem SUT ausgetauscht. Für jeden `CheckDataType` gibt es eine zugehörige Unterklasse von `CheckData`. Sie müssen nur deren Konstruktoren kennen, also führen wir auch nur diese auf:

`BooleanCheckData BooleanCheckData(String identifier, boolean value)`

Erzeugt ein neues `BooleanCheckData` Objekt.

Parameter

`identifier`	Der Bezeichner des Checktyps. Sollte normalerweise dem Bezeichner des `type` Arguments entsprechen, das an `Checker.getCheckData` übergeben wurde.
`value`	Der Wert des Checks, ein Boolean Zustand.

`GeometryCheckData GeometryCheckData(String identifier, int x, int y, int width, int height)`

Erzeugt ein neues `GeometryCheckData` Objekt.

Parameter

`identifier`	Der Bezeichner des Checktyps. Sollte normalerweise dem Bezeichner des `type` Arguments entsprechen, das an `Checker.getCheckData` übergeben wurde.
`x`	Die X-Koordinate für den Check.
`y`	Die Y-Koordinate für den Check.
`width`	Die Breite für den Check.
`height`	Die Höhe für den Check.

`ImageCheckData ImageCheckData(String identifier, ImageRep image, int xOffset, int yOffset, int subX, int subY, int subWidth, int subHeight)`

Erzeugt ein neues `ImageCheckData` Objekt.

Parameter

`identifier`	Der Bezeichner des Checktyps. Sollte normalerweise dem Bezeichner des `type` Arguments entsprechen, das an `Checker.getCheckData` übergeben wurde.
`image`	Das Abbild für den Check. Näheres hierzu unter Abschnitt 54.9(1232).
`xOffset`	Ein optionaler X-Offset.
`yOffset`	Ein optionaler Y-Offset.
`subX`	Die X-Koordinate einer optionalen Check-Region.
`subY`	Die Y-Koordinate einer optionalen Check-Region.
`subWidth`	Die Breite einer optionalen Check-Region.
`subHeight`	Die Höhe einer optionalen Check-Region.

`SelectableItemsCheckData SelectableItemsCheckData(String identifier, Object[][] values)`

Erzeugt ein neues `SelectableItemsCheckData` Objekt.

Parameter

`identifier`	Der Bezeichner des Checktyps. Sollte normalerweise dem Bezeichner des `type` Arguments entsprechen, das an `Checker.getCheckData` übergeben wurde.
`values`	Der Wert des Checks, ein Array von Arrays mit einem String und einem Boolean für ”regexp” und einem Boolean für ”selected”.

`StringCheckData StringCheckData(String identifier, String value)`

Erzeugt ein neues `StringCheckData` Objekt.

Parameter

`identifier`	Der Bezeichner des Checktyps. Sollte normalerweise dem Bezeichner des `type` Arguments entsprechen, das an `Checker.getCheckData` übergeben wurde.
`value`	Der Wert des Checks, ein String.

`StringItemsCheckData StringItemsCheckData(String identifier, String[] values)`

Erzeugt ein neues `StringItemsCheckData` Objekt.

Parameter

`identifier`	Der Bezeichner des Checktyps. Sollte normalerweise dem Bezeichner des `type` Arguments entsprechen, das an `Checker.getCheckData` übergeben wurde.
`values`	Der Wert des Checks, ein Array von Strings.

Darüber hinaus kann für ein ImageCheckData optional ein Algorithmus definiert werden.

`void setAlgorithm(String algorithm)`
Definiert einen Algorithmus. Eine genaue Beschreibung finden Sie in Details des Algorithmus zum Bildvergleich(1309).

54.4.6 Die `CheckerRegistry`

Wenn Ihr `Checker` implementiert und instanziiert ist, muss er bei der `CheckerRegistry` registriert werden. Die Klasse `de.qfs.apps.qftest.extensions.checks.CheckerRegistry` bietet hierzu folgende Methoden:

`static CheckerRegistry instance()`
Es gibt immer nur ein einziges `CheckerRegistry` Objekt und diese Methode ist der einzige Weg, Zugriff auf diese Singleton Instanz erlangen.

Rückgabewert Die `CheckerRegistry` Singleton Instanz.

`void registerChecker(Object element, Checker checker)`
Registriert einen `Checker` für ein spezifisches GUI Element. Der Checker beeinträchtigt nicht die Garbage-Collection und wird automatisch entfernt, wenn die Komponente nicht mehr erreichbar ist.

Parameter

`element`	Das GUI Element für das registriert wird.
`checker`	Der zu registrierende Checker.

`void registerChecker(String clazz, Checker checker)`
Registriert einen `Checker` für eine spezifische Klasse von GUI Elementen.

Parameter

`clazz`	Der Name der Klasse für die registriert wird.
`checker`	Der zu registrierende Checker.

`void unregisterChecker(Object element, Checker checker)`
Entfernt einen `Checker` für ein spezifisches GUI Element.

Parameter

`element`	Das GUI Element für das entfernt wird.
`checker`	Der zu entfernende Checker.

`void unregisterChecker(String clazz, Checker checker)`
Entfernt einen `Checker` für eine spezifische Klasse von GUI Elementen.

Parameter

`clazz`	Der Name der Klasse für die entfernt wird.
`checker`	Der zu entfernende Checker.

54.4.7 Beispiel für einen Checker

Das folgende Jython SUT-Skript zeigt, wie man alles zu einem eigenen Checker zusammenfügt. Nehmen wir an, Sie haben eine Java-Swing-Anwendung und möchten alle Labels in einem Panel gleichzeitig überprüfen. Dazu müssen Sie über alle Komponenten im Panel und deren Kind-Komponenten iterieren, dabei die Label-Komponenten identifizieren und eine Liste mit den enthaltenen Texten generieren. In QF-Test Notation heißt das, Sie brauchen einen `CheckDataType.STRING_LIST` Checktyp und müssen die Daten in einem `StringItemsCheckData` Objekt zurückliefern:

```
from de.qfs.apps.qftest.extensions import ResolverRegistry
from de.qfs.apps.qftest.extensions.checks import CheckerRegistry, \
    Checker, DefaultCheckType, CheckDataType
from de.qfs.apps.qftest.extensions.items import ItemRegistry
from de.qfs.apps.qftest.shared.data.check import StringItemsCheckData
from de.qfs.lib.util import Pair
from java.lang import String
import jarray
componentClass = "javax.swing.JPanel"
allLabelsCheckType = DefaultCheckType("AllLabels",
    CheckDataType.STRING_LIST,
    "Alle Labels im Panel")
class AllLabelsChecker(Checker):
    def __init__(self):
        pass
    def getSupportedCheckTypes(self, com, item):
        return jarray.array([allLabelsCheckType], DefaultCheckType)
    def getCheckData(self, com, item, checkType):
        if allLabelsCheckType.getIdentifier() == checkType.getIdentifier():
            labels = self._findLabels(com)
            labels = map(lambda l: l.getText(), labels)
            values = jarray.array(labels, String)
            return StringItemsCheckData(checkType.getIdentifier(), values)
        return None
    def getCheckDataAndItem(self, com, item, checkType):
        data = self.getCheckData(com, item, checkType)
        if data is None:
            return None
        return Pair(data, None)
    def _findLabels(self, com, labels=None):
        if labels is None:
            labels = []
        if ResolverRegistry.instance().isInstance(com, "javax.swing.JLabel"):
            labels.append(com)
        for c in com.getComponents():
            self._findLabels(c, labels)
        return labels
def unregister():
    try:
        CheckerRegistry.instance().unregisterChecker(
            componentClass, allLabelsChecker)
    except:
        pass
def register():
    unregister()
    global allLabelsChecker
    allLabelsChecker = AllLabelsChecker()
    CheckerRegistry.instance().registerChecker(
        componentClass, allLabelsChecker)
register()
```

Beispiel 54.30: Alle Labels in einem Panel checken

Nach Ausführung des Skripts findet man einen neuen Eintrag ”Alle Labels im Panel” im Checktyp Menü, sobald man im Checkaufnahmemodus mit der rechten Maustaste auf eine `JPanel` Komponente klickt (cf. Abschnitt 4.3(43)). Wenn Sie den `allLabelsChecker` überall in Ihrer Clientanwendung verwenden möchten, können Sie das obige SUT-Skript hinter den Warten auf Client Knoten in der Vorbereitung stellen. Ansonsten können Sie den Checker auch nach Bedarf registrieren und später in einem anderen SUT-Skript wieder entfernen:

```
from de.qfs.apps.qftest.extensions.checks import CheckerRegistry
global allLabelsChecker
def unregister():
    try:
        CheckerRegistry.instance().unregisterChecker(
            "javax.swing.JPanel", allLabelsChecker)
    except:
        pass
unregister()
```

Beispiel 54.31: Den Label Checker entfernen

54.5 Das Eclipse Graphical Editing Framework (GEF)

3.2+

Das Graphical Editing Framework (GEF) besteht aus mehreren Eclipse Plugins, mit deren Hilfe graphische Editoren erstellt werden können, um damit beliebige Datenmodelle zu visualisieren. Diese Bibliothek ist sehr populär, so dass QF-Test die Aufnahme und Wiedergabe von GEF Elementen bereits seit Version 2.2 unterstützt. Dies ist auch ein gutes Beispiel für die Stärke des `ItemResolver` Konzepts (siehe Abschnitt 54.3(1197)), denn das `gef` Jython-Modul enthält eine Implementierung genau dieser Schnittstelle.

Das `gef` Modul unterstützt GEF-Editoren in generischer Weise und kann sogar mit mehreren Editoren gleichzeitig umgehen. Zwar werden auch für GMF-Anwendungen brauchbare Namen für Unterelemente vergeben, doch nicht immer sind diese hinreichend gut. Je nachdem, wie die zugrunde liegenden Modell-Klassen aussehen, bleibt noch etwas Arbeit zu tun, nämlich die Implementierung eigener Resolver, die brauchbare Namen und Werte für die Unterelemente liefern.

54.5.1 Aufnahme von GEF Elementen

Die eigentliche GEF Komponente ist der `FigureCanvas`. Dieser stellt `Figures` dar, die `EditParts` repräsentieren. Nimmt man einen Mausklick auf solch ein Element auf, registriert QF-Test nicht bloß ein Mausevent für den Canvas mit entsprechenden (x,y)

Werten für die Position, sondern versucht, das Objekt unter dem Mauscursor zu erkennen. Die aufgenommene QF-Test ID der Komponente sieht zum Beispiel so aus:

```
 canvas@/Diagram/My ship/ShipLargeCargo (wine)
canvas@Connection-2
canvas@/Diagram/Rectangle 16329001
```

wobei ”canvas” die QF-Test ID der `FigureCanvas` Komponente ist, gefolgt vom Element-Index des erkannten `EditPart` Objekts (siehe Abschnitt 5.9(92)). `EditParts` sind in einer Baumstruktur organisiert, erkennbar am Pfadtrenner ’/’. Die Namen der einzelnen Elemente werden folgendermaßen abgeleitet:

- Der Elementname ist getModel().toString(), wenn darin kein Hashwert (z. B. NodeImpl@2d862) enthalten ist.
- QF-Test versucht, im Modell einen Namen für das Element zu finden (”My ship” in den obigen Beispielen).
- Die Klasse zusammen mit einer Beschreibung wird aufgenommen, etwa ”ShipLargeCargo (wine)”.
- Wenn es keine Beschreibung aber mehrere Elemente einer Klasse gibt, wird an diese ein Index angehangen, z. B. ”Connection-2” für die dritte Connection.
- Der Wurzelknoten heißt stets ”Diagram”.

Man kann sich denken, dass die generierten Namen nicht immer sinnvoll sind. Elemente könnten gelöscht werden und aufgezeichnete Indizes damit ihre Gültigkeit verlieren. Oder der Elementname ist instabil wie bei ”Rectangle 16329001” aus dem GEF Shapes Beispiel: Die Zahl ist rein zufällig und wird bei einem Neustart der Anwendung neu ermittelt. Drei Möglichkeiten gibt es, um solche Probleme zu lösen:

- Anstatt mit einem textuellen Index zu arbeiten, kann man es mit einem numerischen versuchen. Dazu setzt man in den Aufnahmeoptionen das Format für die Unterelemente auf den Wert ’Zahl’ (siehe Abschnitt 41.2.4(523)). Allerdings ist diese Lösung nicht sehr befriedigend, denn ein numerischer Index wie `/0/1` sagt nichts über das Element aus.
- Man könnte sich an die Entwickler wenden und diese davon zu überzeugen versuchen, eine brauchbare Implementierung der Methode `toString()` für das Datenmodell eines Elements zu liefern. Das würde Ihnen das Leben leicht machen, aber eben nur, wenn die Entwickler kooperativ sind.
- Implementierung der `ItemNameResolver2` Schnittstelle. Das ist nicht ganz einfach, doch leider oft unumgänglich. Dieses Thema wird im nächsten Abschnitt behandelt.

54.5.2 Implementierung eines ItemNameResolver2 für GEF

Wie in Abschnitt 54.1[1154] ausgeführt, ist `ItemNameResolver2` die Schnittstelle, um Namen für Elemente zu ändern oder überhaupt erst zu definieren. Ein erster Ansatz zur Implementierung ist das folgende Jython SUT-Skript[720], das unter Extrasequenzen[629] eingefügt werden kann:

```
def getItemName(canvas, item, name):
    print "name: %s" %name
    print "item: %s" %(item.__class__)
    model = item.getModel()
    print "model: %s" %(model.__class__)
resolvers.addItemNameResolver2("myGefItemNames", getItemName,
    "org.eclipse.draw2d.FigureCanvas")
```

Beispiel 54.32: Ein erster GEF ItemNameResolver2

Um die Installation des Resolvers zu vereinfachen, wird das in Abschnitt 54.1[1154] beschriebene `resolvers` Modul verwendet. Der Resolver wird auf die Klasse `FigureCanvas` registriert, die `EditParts` als Unterlemente enthält. Der von QF-Test generierte Name wird der Funktion `getItemName()` beim Aufruf als drittes Argument übergeben. Wenn man das Skript nun ausführt und dann den Aufnahmeknopf drückt, werden beim Überfahren der Elemente mit der Maus - man sollte zuvor ein paar davon erzeugt haben - Informationen im Terminal von QF-Test ausgegeben, etwa wie folgt:

```
 name: Rectangle 16329001
item: org.eclipse.gef.examples.shapes.parts.ShapeEditPart
model: org.eclipse.gef.examples.shapes.model.RectangularShape
```

Abgesehen von diesen Ausgaben ist der Resolver ohne Funktion. Die Frage ist nun: Gibt es im Modell irgendeine Eigenschaft oder Methode, die einen vernüftigen Namen für das Element liefert? Für das GEF Shapes Beispiel lautet die Antwort: Nein. Hoffentlich sind Sie mit Ihrer Anwendung in einer besseren Lage. Um das herauszufinden, fügen Sie der Funktion `getItemName()` die Zeile

```
 print dir(model)
```

hinzu und führen das Skript erneut aus. Nun werden beim Bewegen der Maus über die Elemente (im Aufnahmenmodus) auch die Methoden des Modells angezeigt. Mit etwas Glück tauchen hier Methoden wie `getId()` oder `getLabel()` auf, so dass man einen Resolver wie den folgenden implementieren kann.

```
def getItemName(canvas, item, name):
    model = item.getModel()
    return model.getId()
resolvers.addItemNameResolver2("myGefItemNames", getItemName,
    "org.eclipse.draw2d.FigureCanvas")
```

Beispiel 54.33: Ein einfacher ItemNameResolver2

Zurück zum GEF Shapes Beispiel, wo es solche Methoden nicht gibt. Hier sind nur Informationen über die Geometrie verfügbar, doch das ist wenig hilfreich. Zumindest lassen sich aber Rechtecke und Ellipsen unterscheiden. Um die Elementnamen eindeutig zu machen, fügen wir einfach den Index der Figur (des Kind-Knotens) an, wie im folgenden Resolver gezeigt:

```
def getItemName(canvas, item, name):
    name = None
    shapes = "org.eclipse.gef.examples.shapes"
    diagrammEditPart = shapes + ".parts.DiagramEditPart"
    shapeEditPart = shapes + ".parts.ShapeEditPart"
    connectionEditPart = shapes + ".parts.ConnectionEditPart"
    ellipticalShape = shapes + ".model.EllipticalShape"
    rectangularShape = shapes + ".model.RectangularShape"
    if qf.isInstance(item, shapeEditPart):
        siblings = item.getParent().getChildren()
        for i in range(len(siblings)):
            if (item == siblings[i]):
                if qf.isInstance(item.getModel(), ellipticalShape):
                    name = "Ellipse " + str(i)
                elif qf.isInstance(item.getModel(),
                                                rectangularShape):
                    name = "Rectangle " + str(i)
    elif qf.isInstance(item, connectionEditPart):
        source = item.getSource()
        target = item.getTarget()
        sourceName = getItemName(canvas, source, str(source.getModel()))
        targetName = getItemName(canvas, target, str(target.getModel()))
        name = "Connection " + sourceName + " " + targetName
    elif qf.isInstance(item, diagrammEditPart):
        name = "Diagram"
    return name
resolvers.addItemNameResolver2("shapesItemNames", getItemName,
    "org.eclipse.draw2d.FigureCanvas")
```

Beispiel 54.34: Ein ItemNameResolver2 für GEF Shapes

Mit diesem Resolver wird der Element-Index zu sowas wie

```
/Diagram/Rectangle 1
```

wobei die abschließende Zahl der Index des Kind-Knotens ist. Diese Implementierung liefert auch Namen für die Verbindungen, indem `getItemName()` rekursiv für das Quell- und Zielelement aufgerufen wird. Die Typüberprüfung erfolgt mit der Methode `qf.isInstance` (siehe Abschnitt 50.6(1059)), wodurch einem das Importieren der GEF-Klassen (was nicht ganz einfach ist) erspart bleibt.

Sobald der Resolver funktionstüchtig ist, sollte man das Skript in die Vorbereitung(638) Sequenz packen, direkt hinter den Warten auf Client(758) Knoten. Auf diese Weise wird der Resolver automatisch registriert, wenn man das SUT startet.

54.5.3 Implementierung eines ItemValueResolver2 für GEF

Üblicherweise besteht ein GEF Editor aus zwei Teilen. Bislang hatten wir uns auf den Canvas konzentriert, in den die Figuren gezeichnet werden. Nun werfen wir einen Blick auf die Palette, in der die Art der zu zeichnenden Figur ausgewählt wird (z. B. Rechteck, Ellipse oder Verbindung). Die Einträge sehen zwar aus wie Buttons, doch tatsächlich ist die Palette ebenfalls ein `FigureCanvas`. Erfreulicherweise funktioniert hier alles, ohne dass besondere Vorkehrungen getroffen werden müssten, d. h. ohne einen `ItemNameResolver2` zu implementieren. Wenn man etwa auf den 'Rectangle' Button klickt, erkennt QF-Test ein

```
/Palette Root/Palette Container (Shapes)/Palette Entry (Rectangle)
```

Element. Was wird wohl passieren, wenn man einen 'Object value' Check für diesen Button aufnimmt? Man könnte erwarten, den Text 'Rectangle' zu erhalten, doch tatsächlich ist der Wert des Elements

```
Palette Entry (Rectangle)
```

Der Grund dafür ist, dass Name und Wert eines Elements normalerweise gleich sind. Um dieses Verhalten zu ändern und selbstdefinierte Werte zu erhalten, muss ein `ItemValueResolver2` implementiert werden. Diese Schnittstelle ist der `ItemNameResolver2` Schnittstelle ganz ähnlich. Für die Palette können wir etwa das Folgende codieren:

```
def getItemValue(canvas, item, value):
    value = None
    paletteEditPart = \
        "org.eclipse.gef.ui.palette.editparts.PaletteEditPart"
    if qf.isInstance(item, paletteEditPart):
        value = item.getModel().getLabel()
    return value
resolvers.addItemValueResolver2("shapesItemValues", getItemValue,
    "org.eclipse.draw2d.FigureCanvas")
```

Beispiel 54.35: Ein ItemValueResolver2 für die GEF Shapes Palette

Die Methode `getLabel()` liefert den Text des Elements, so wie er in der Palette angezeigt wird.

54.6 Listener für den aktuellen Testlauf

3.1+

Einmal beim aktuellen Runcontext via `rc.addTestRunListener` registriert, wird eine Implementierung des `TestRunListener` Interfaces benachrichtigt, wenn ein Knoten betreten oder verlassen wird oder wenn Probleme auftreten. Ein illustratives Beispiel finden Sie in der Testsuite `TestRunListener.qft` im Verzeichnis `demo/runlistener` Ihrer QF-Test Installation. Am besten deaktivieren Sie den Debugger, bevor Sie diese Testsuite komplett ausführen.

Hinweis Eine Variante des `TestRunListener` Interfaces namens `DaemonTestRunListener` kann verwendet werden, um einen Testlauf über das Daemon API zu beobachten. Es wird in Abschnitt 55.2.5(1294) näher beschrieben.

Die API besteht aus den folgenden Klassen:

54.6.1 Das `TestRunListener` Interface

Das Interface `de.qfs.apps.qftest.extensions.qftest.TestRunListener` muss implementiert und via `rc.addTestRunListener()` beim aktuellen Runcontext registriert werden.

Hinweis Zur Implementierung des Interfaces können Sie von der Klasse `de.qfs.apps.qftest.extensions.qftest.AbstractTestRunListener` ableiten, die für alle Methoden leere Implementierungen bereitstellt, so dass Sie nur die Methoden implementieren müssen, die Sie interessieren.

void nodeEntered(TestRunEvent event)

Benachrichtigt den Listener, dass ein Knoten betreten wurde.

Parameter

event	Der Event mit den Details.

void nodeExited(TestRunEvent event)

Benachrichtigt den Listener, dass ein Knoten verlassen wurde.

Parameter

event	Der Event mit den Details.

void problemOccurred(TestRunEvent event)

Benachrichtigt den Listener, dass ein Problem aufgetreten ist.

Parameter

event	Der Event mit den Details.

void runStarted(TestRunEvent event)

Benachrichtigt den Listener, dass ein Testlauf gestartet wurde.

Parameter

event	Der Event mit den Details, in diesem Fall irrelevant.

void runStopped(TestRunEvent event)

Benachrichtigt den Listener, dass ein Testlauf beendet wurde.

Parameter

event	Der Event mit den Details, in diesem Fall irrelevant.

54.6.2 Die Klasse `TestRunEvent`

Die Klasse `de.qfs.apps.qftest.extensions.qftest.TestRunEvent` beinhaltet Informationen über die aktuell ausgeführten Knoten und den aktuellen Fehlerzustand. Sie definiert die folgenden Konstanten für Ausführungs- und Fehlerzustände: `STATE_NOT_IMPLEMENTED`, `STATE_SKIPPED`, `STATE_OK`, `STATE_WARNING`, `STATE_ERROR` und `STATE_EXCEPTION`. Die beiden ersten Zustände kommen nur bei Testfallsatz(606) und Testfall(599) Knoten vor.

In der `runStopped` Methode können Sie auch abfragen, ob der Testlauf normal beendet oder unterbrochen wurde. Hierfür können Sie die Konstanten `STATE_RUN_TERMINATED` und `STATE_RUN_INTERRUPTED` verwenden.

JsonObject asJsonValue()
Serialisiert das Event-Objekt als `JsonObject`. Dieses kann verwendet werden, um den Datenaustausch mit json-basierten Tools wie Web-Services oder Datenbanken zu vereinfachen. Die API der in QF-Test eingebetteten JSON-Bibliothek ist in `doc/javadoc/json.zip` in Ihrer QF-Test Installation dokumentiert.

Rückgabewert Das Event als `JsonObject`.

int getErrors()

Liefert die Zahl der Fehler für den Knoten.

Rückgabewert Die Gesamtzahl der Fehler für den gerade verlassenen Knoten. Verfügbar nur für `nodeExited`.

int getExceptions()

Liefert die Zahl der Exceptions für den Knoten.

Rückgabewert Die Gesamtzahl der Exceptions für den gerade verlassenen Knoten. Verfügbar nur für `nodeExited`.

int getLocalState()

Liefert den lokalen Status für den aktuellen Knoten.

Rückgabewert Der lokale Status für den aktuellen Knoten. Dies ist der höchste Fehlerzustand dieses Knotens und all seiner Kinder, unabhängig davon ob dieser weiter nach oben propagiert. Verfügbar nur für `nodeExited` und `problemOccurred`. Für Testfallsatz und Testfall Knoten kann der lokale Staus auch `STATE_SKIPPED` oder `STATE_NOT_IMPLEMENTED` sein.

String getMessage()

Liefert die aktuelle Fehlermeldung.

Rückgabewert Die Meldung für die aktuelle Warnung, Fehler oder Exception. Verfügbar nur für `problemOccurred`.

TestSuiteNode getNode()

Liefert den aktuellen Knoten

Rückgabewert Der aktuelle Knoten oder null für `runStarted` und `runStopped`.

TestSuiteNode[] getPath()

Liefert den gesamten Pfad für den aktuellen Knoten.

Rückgabewert Der Pfad für den aktuellen Knoten aus Sicht des Protokolls, vergleichbar mit dem Ausführungs-Stapel. Null für `runStarted` und `runStopped`. Der letzte Knoten im Array ist der aktuelle Knoten.

`int getState()`

Liefert den Status für den aktuellen Knoten.

Rückgabewert Der propagierende Status für den aktuellen Knoten. Dies ist der höchste Fehlerzustand dieses Knotens und all seiner Kinder, möglicherweise beschränkt durch ein Attribut Maximaler Fehler(619). Verfügbar nur für `nodeExited` und `problemOccurred`.

`int getWarnings()`

Liefert die Zahl der Warnungen für den Knoten.

Rückgabewert Die Gesamtzahl der Warnungen für den gerade verlassenen Knoten. Verfügbar nur für `nodeExited`.

54.6.3 Die Klasse `TestSuiteNode`

Die Klasse `de.qfs.apps.qftest.extensions.qftest.TestSuiteNode` repräsentiert einen aktuell ausgeführten QF-Test Knoten. Sie beinhaltet Informationen über die Art des Knotens, seinen Namen, Kommentar, etc.

`JsonObject asJsonValue()`

Serialisiert das Knoten-Objekt als `JsonObject`.

Rückgabewert Der Knoten als `JsonObject`.

`String getComment()`

Liefert die Bemerkung des Knotens.

Rückgabewert Die expandierte Bemerkung des Knotens.

`String getComponentId()`

Liefert die QF-Test ID der Komponente des Knotens, falls verfügbar.

Rückgabewert Die expandierte QF-Test ID der Komponente des Knotens.

`String getExpandedTreeName()`

Liefert den expandierten ”Baumnamen” des Knotens.

Rückgabewert Der Name des Knotens, wie er im Baum der Testsuite dargestellt wird, wobei Variablen durch deren Inhalt ersetzt sind.

`String getId()`

Liefert die QF-Test ID des Knotens.

Rückgabewert Die QF-Test ID des Knotens.

`String getName()`

Liefert den Namen des Knotens.

Rückgabewert Der Name des Knotens oder null, falls dieser kein Attribut 'Name' besitzt.

`String getReportName()`

Liefert den Reportnamen des Knotens.

Rückgabewert Der expandierte Reportname für Testfallsatz[606] und Testfall[599] Knoten. Der normale Name für andere Knoten oder null falls nicht definiert.

`String getSuite()`

Liefert die Testsuite, zu der der Knoten gehört.

Rückgabewert Der komplette Pfad der Testsuite, zu der der Knoten gehört.

`String getTreeName()`

Liefert den "Baumnamen" des Knotens.

Rückgabewert Der Name des Knotens wie er im Baum der Testsuite dargestellt wird.

`String getType()`

Liefert die Art des Knotens.

Rückgabewert Die Art des Knotens. Dies ist der letzte Teil des Namens der Klasse, die diesen Knoten implementiert.

`String getVerboseReportName()`

Liefert den Reportnamen bzw. den Namen des Knotens.

Rückgabewert Der expandierte Reportname für Testfallsatz[606] und Testfall[599] Knoten. Falls nicht vorhanden wird der expandierte Knotennamen zurückgegeben.

54.7 ResetListener

4.0.3+

Während der Testentwicklung ist es manchmal notwendig alle QF-Test Variablen zu löschen, sowie alle gestarteten Clients zu stoppen. Hierdurch wird sicher gestellt, dass eine spätere / darauffolgende Testausführung nicht wegen eines Seiteneffekts einer vorherigen fehlschlägt. Dieses Verhalten kann z.B. wegen fehlerhaft gesetzter globaler Variablen bzw. wegen eines sich im falschen Zustand befindenden Clients passieren.

Für das Zurücksetzen aller QF-Test Variablen und Clients bietet sich Menüaktion Wiedergabe→Alles zurücksetzen an. Mithilfe eines TestRunListeners ist es möglich diese Aktion an die eigenen Bedürfnisse anzupassen. Dadurch ist es z.B. möglich

sicherzustellen, dass

- (bestimmte) clients bestehen bleiben
- bestimmte globale QF-Test Variablen wiederhergestellt werden
- bestimmte zusätzliche Aktionen ausgeführt werden, wie z.B. das Löschen globaler Jython variablen

Ein ResetListener stellt jede bei QF-Test registrierte Klasse dar, die das ResetListener Interface `de.qfs.apps.qftest.extensions.qftest.ResetListener` realisiert. Zum Verwalten der bei QF-Test registrierten Klassen, stellt der Runcontext die Methoden(1030) `addResetListener`, `isResetListenerRegistered` und `removeResetListener` bereit.

Das ResetListener Interface besteht hierbei aus den folgenden Methoden:

`void afterReset()`
Diese Methode wird aufgerufen, nachdem alle Variablen und Clients zurückgesetzt wurden.

`Set<String> beforeReset()`
Diese Methode wird aufgerufen, bevor alle Variablen und Clients zurückgesetzt werden. Die Implementierung dieser Methode kann weiterhin verhindern, dass bestimmte von QF-Test gestartete SUTs durch das Ausführen von Wiedergabe→Alles zurücksetzen beendet werden.

Rückgabewert Ein `java.util.Set<String>` Objekt, dass die Namen aller Clients beinhaltet, die nicht beendet/zurückgesetzt werden sollen.

Das folgende Beispiel beschreibt die Implementierung und Registrierung eines Reset-Listners, der verhindert, dass die QF-Test Variable ”client” durch den Reset gelöscht wird, als auch dass das `"client"` SUT beendet wird.

```
from java.util import HashSet
from de.qfs.apps.qftest.extensions.qftest import ResetListener
from de.qfs.apps.qftest.shared.exceptions import UnboundVariableException
class L(ResetListener):
    def beforeReset(self):
        try:
            self.client = rc.getStr("client")
            h = HashSet()
            h.add(self.client)
            return h
        except UnboundVariableException:
            self.client = None
    def afterReset(self):
        if self.client != None:
            rc.setGlobal("client", self.client)
global resetListener
try:
    resetListener
except:
    resetListener = L()
    rc.addResetListener(resetListener)
```

Beispiel 54.36: Beispielhafte Implementierung eines ResetListeners

54.8 DOM Prozessoren

Beim Erstellen eines Reports aus einem Protokoll, oder der Generierung von Package Dokumentation aus einer Testsuite, arbeitet QF-Test mit einem zweistufigen Prozess. Im ersten Schritt wird ein XML-Dokument erstellt, welches im zweiten Schritt zu einem HTML-Dokument weiterverarbeitet wird. Beide Transformationen werden mittels XSLT Stylesheets durchgeführt.

Hinweis

Der Begriff DOM (für *Document Object Model*) findet auch bei XML-Dokumenten Anwendung, nicht nur bei HTML-Webseiten. Dieser Abschnitt befasst sich ausschließlich mit XML und XSLT und hat nichts mit dem DOM eines Web SUT zu tun.

Leider sind XSLT Stylesheets nicht sonderlich brauchbar, wenn es darum geht, reinen Text zu bearbeiten. Die Bemerkung Felder in Testfallsatz, Testfall, Testschritt, Package oder Prozedur Knoten enthalten aber oft zusätzliche Informationen in einer internen Struktur, die von Anwender zu Anwender verschieden ist, je nachdem, welche Konventionen verwendet werden. So werden zum Beispiel häufig JavaDoc Tags verwendet, um die Parameter einer Prozedur zu beschreiben. Nach der ersten Stufe der Transformation sieht zum Beispiel die Bemerkung der Prozedur `qfs.swing.menu.select` aus unserer Standardbibliothek wie folgt aus:

```
<comment>Select an item from a menu.
For example: for the File -> Open action, the QF-Test component ID
 "File" is the menu, and the QF-Test component ID "Open" is the item.
@param  client  The name of the SUT client.
@param  menu    The QF-Test ID of the menu.
@param  item    The QF-Test ID of menu item.</comment>
```

Beispiel 54.37: Beispiel Bemerkung nach der ersten Transformation

Es ist überaus schwierig, nur mittels XSLT die `@param` Tags auszuwerten. Hier kommen nun die DOM Prozessoren ist Spiel. Zwischen der ersten und zweiten Transformation führt QF-Test optional eine zusätzliche Transformation direkt auf dem DOM des XML-Dokuments aus. Bei dieser Transformation durchläuft QF-Test den gesamten DOM Baum und ruft für jedes Element die dafür registrierten DOM Prozessoren auf, die dadurch die Möglichkeit haben, das DOM zu verändern.

Hinweis Für JDK 1.4 ist das XML Document Object Model (DOM) Teil der Standard-API. Für ältere JDK-Versionen wird es vom XML-Parser xerces (aus dem Apache Projekt) bereitgestellt, der Teil von QF-Test ist. Die API-Spezifikation des DOM finden Sie unter anderem unter http://download.oracle.com/javase/1.5.0/docs/api/org/w3c/dom/package-summary.html.

54.8.1 Das `DOMProcessor` Interface

Das zu implementierende Interface ist `de.qfs.apps.qftest.extensions.DOMProcessor`. Es ist im Grunde trivial und besteht nur aus einer einzigen Methode:

`Element process(Element node)`

Verarbeitet einen Element Knoten.

Rückgabewert Ein Element oder null. Wird null zurückgegeben, werden die Kindknoten des Elements ganz normal verarbeitet, andernfalls werden die Kindknoten nicht weiter betrachtet. Wird ein anderes Element als das Original zurückgeliefert, wird das Original im DOM durch den Rückgabewert ersetzt.

In der `process` Methode kann der Prozessor tun und lassen, was er will, solange er sich dabei auf den Knoten und seine (direkten oder indirekten) Unterknoten beschränkt. Das Element kann durch Rückgabe eines anderen Elements komplett ersetzt werden.

Hinweis Um ein Element aus dem DOM zu löschen, muss der `DOMProcessor` für einen höher

gelegenen Knoten, z.B. den Vaterknoten, registriert werden. Der aktuelle Knoten darf in der `process` Methode nicht aus dem DOM entfernt werden.

Mit QF-Test werden zwei Beispiel Implementationen von DOM Prozessoren ausgeliefert. Der `ParagraphProcessor` ist in Java geschrieben und wird zur Illustration im Verzeichnis `misc` zur Verfügung gestellt. Er wird intern verwendet, um Bemerkungen an Leerzeilen in einzelne Absätze aufzubrechen.

Ebenfalls im `misc` Verzeichnis befindet sich der `DocTagProcessor`, der zur Transformation von JavaDoc Tags dient, wie sie im obigen Beispiel beschrieben wurden. Nach seinem Einsatz sähe dieses Beispiel wie folgt aus:

```
<comment>Select an item from a menu.
For example: for the File -> Open action, the QF-Test component ID
 "File" is the menu, and the QF-Test component ID "Open" is the item.</comment>
<param name="client">The name of the SUT client.</param>
<param name="menu">The QF-Test ID of the menu.</param>
<param name="item">The QF-Test ID of menu item.</param>
```

Beispiel 54.38: Beispiel Bemerkung nach Einsatz eines DOM Prozessors

Daraus kann im zweiten Schritt ohne großen Aufwand sinnvolles HTML generiert werden.

54.8.2 Die `DOMProcessorRegistry`

Damit ein DOM Prozessor zum Einsatz kommen kann, muss er zunächst für die entsprechenden Elemente registriert werden. Dies geschieht mit Hilfe der `DOMProcessorRegistry`.

Für jede Art von Transformation gibt es eine `DOMProcessorRegistry` Instanz, die durch einen Namen identifiziert wird. Für die Reportgenerierung lautet dieser `"report"`, für die Testfall Dokumentation `"testdoc"` und für die Package Dokumentation `"pkgdoc"`. Dazu kommt je eine Variante für die Generierung der Zusammenfassungen mit den Namen `"report-summary"`, `"testdoc-summary"` und `"pkgdoc-summary"`. Eine Instanz der Registry erhalten Sie mit Hilfe der Methode `instance`:

`DOMProcessorRegistry instance(String identifier)`

Liefert eine Registry Instanz .

Parameter

`identifier`	Der Bezeichner der entsprechenden Transformation.

Bei den restlichen Methoden handelt es sich um den üblichen Satz von register/unregister Varianten:

`void registerDOMProcessor(DOMProcessor processor)`
Registriert einen generischen DOM Prozessor, der für alle Arten von von Elementen aufgerufen wird.
Parameter

`processor`	Der zu registrierende Prozessor.

`void registerDOMProcessor(String node, DOMProcessor processor)`
Registriert einen DOM Prozessor für ein spezifisches Element.
Parameter

`name`	Der Name des Elements.
`processor`	Der zu registrierende Prozessor.

`void unregisterDOMProcessor(DOMProcessor processor)`
Entfernt einen generischen DOM Prozessor, der für alle Arten von von Elementen aufgerufen wird.
Parameter

`processor`	Der zu entfernende Prozessor.

`void unregisterDOMProcessor(String node, DOMProcessor processor)`
Entfernt einen DOM Prozessor für ein spezifisches Element.
Parameter

`name`	Der Name des Elements.
`processor`	Der zu entfernende Prozessor.

`void unregisterDOMProcessors()`
Entfernt alle DOM Prozessoren.

54.8.3 Fehlerbehandlung

Exceptions, die in der `process` Methode geworfen werden, fängt QF-Test ab und meldet sie. Die Transformation wird in diesem Fall gestoppt.

54.9 Image API Erweiterungen

3.0+

Die Image API von QF-Test benutzt für die Bildspeicherung die Klasse `ImageRep`. Diese Klasse speichert Bilder technologie- und plattformunabhängig. Des weiteren wird ein Interface bereitgestellt, um eigene Bildvergleiche durchzuführen.

54.9.1 Die `ImageRep` Klasse

Die Klasse `de.qfs.apps.qftest.shared.extensions.image.ImageRep` ist eine Wrapper Klasse für technologie- und plattformunabhängige Bildspeicherungen.

Die Klasse speichert das Bild entweder als ARGB-Informationen ab (ein Integer Feld) oder als RGB Information ab (ein byte Feld). Außerdem kann noch ein Name sowie die Höhe und Breite des Bildes gespeichert werden.

Die `ImageRep` Klasse bietet auch eine `equals` Methode, um Bildvergleiche durchzuführen. Falls Sie Ihren eigenen Bildschirmvergleichsalgorithmus verwenden möchten, müssen Sie das Interface `ImageComparator` implementieren. Diese eigene Erweiterung muss dann beim `ImageRep` Objekt noch registriert werden (siehe Abschnitt 54.9.2(1236) für mehr Informationen).

`ImageRep ImageRep()`

Konstruktor der `ImageRep` Klasse.

`ImageRep ImageRep(String name, byte[] rgb, boolean png, int width, int height)`

Konstruktor der `ImageRep` Klasse.

Parameter

`name`	Der Name des `ImageRep` Objektes.
`rgb`	Ein byte-Feld, das die RGB-Information des Bildes enthält.
`png`	Gibt an, ob das Bild bereits PNG formatiert ist.
`width`	Die Breite des Bildes.
`height`	Die Höhe des Bildes.

`ImageRep ImageRep(String name, int[] argb, boolean png, int width, int height)`

Konstruktor der `ImageRep` Klasse.

Parameter

`name`	Der Name des `ImageRep` Objektes.
`argb`	Ein int-Feld, das die ARGB-Information des Bildes enthält.
`png`	Gibt an, ob das Bild bereits PNG formatiert ist.
`width`	Die Breite des Bildes.
`height`	Die Höhe des Bildes.

`void crop(int x, int y, int width, int height)`

Speichert nur einen speziellen Ausschnitt des Bildes.

Parameter

`x`	Die X Koordinate der linken oberen Ecke des Bereiches.
`y`	Die Y Koordinate der linken oberen Ecke des Bereiches.
`width`	Die Breite des Bereiches.
`height`	Die Höhe des Bereiches.

`ImageRepDrawer draw()`

Erzeuge ein ImageRepDrawer Objekt für dieses Bild. Dieses Objekt erlaubt es, Linien, Rechtecke und weitere Figuren auf dem Bild zu zeichnen.

`ImageRepDrawer draw(Object obj)`

Erzeuge ein ImageRepDrawer Objekt für dieses Bild.

Parameter

`obj`	Ein Lamdba Objekt. Das Lambda Objekt bekommt ein java.awt.Graphics2D Objekt als Eingabe, welches benutzt werden kann, um auf das Bild zu zeichnen.

`boolean equals(ImageRep compare)`

Liefert zurück, ob das aktuelle Bild mit einem anderen Bild gleich ist. Dazu wird die `equals` Methode der aktuellen `ImageComparator` Implementierung verwendet.

Parameter

`compare`	Das zu vergleichende `ImageRep` Objekt.
Rückgabewert	True, wenn die Bilder gleich sind, sonst false.

`int[] getARGB()`

Liefert die ARGB Information. Wenn keine ARGB Information vorhanden ist, jedoch die RGB Information, dann wird die aktuelle RGB Information umgewandelt.

Rückgabewert	Die aktuelle ARGB Information.

`ImageComparator getComparator()`

Liefert die aktuelle `ImageComparator` Implementierung.

Rückgabewert	Die aktuelle `ImageComparator` Implementierung.

`int getHeight()`

Liefert die Höhe.

Rückgabewert	Die aktuelle Höhe.

`String getName()`

Liefert den Namen.

Rückgabewert	Der aktuelle Name.

`int getPixel(int x, int y)`

Liefert den ARGB-Wert für ein Pixel.

Parameter

`x`	Die X Koordinate des Pixels.
`y`	Die Y Koordinate des Pixels.
Rückgabewert	Der Pixelwert.

`byte[] getPng()`

Liefert die RGB Information. Wenn keine RGB Information vorhanden ist, jedoch die ARGB Information, dann wird die aktuelle ARGB Information umgewandelt.

Rückgabewert	Die aktuelle RGB Information.

`int getWidth()`

Liefert die Breite.

Rückgabewert	Die aktuelle Breite.

`void setARGB(int[] argb)`

Setzt die ARGB Information.

Parameter

`argb`	Die neue ARGB Information.

`void setComparator(ImageComparator comparator)`

Setzt eine `ImageComparator` Implementierung, welche für Bildvergleiche verwendet wird.

Parameter

`comparator`	Die neue `ImageComparator` Implementierung.

`void setHeight(int height)`

Setzt die Höhe.

Parameter

`height`	Die neue Höhe.

`void setName(String name)`

Setzt den Namen.

Parameter

`name`	Der neue Name.

`void setPng(byte[] png)`

Setzt die RGB Information.

Parameter

`png`	Die neue RGB Information.

`void setWidth(int width)`

Setzt die Breite.

Parameter

`width`	Die neue Breite.

54.9.2 Das `ImageComparator` Interface

Das Interface `de.qfs.apps.qftest.shared.extensions.image.ImageComparator` kann implementiert werden, wenn Sie einen eigenen Bildvergleichsalgorithmus verwenden wollen.

Die Implementierung muss dann beim verwendeten `ImageRep` Objekt mittels der `setComparator` Methode registriert werden.

`boolean equals(ImageRep actual, ImageRep expected)`

Liefert zurück, ob das aktuelle Bild mit einem anderen Bild gleich ist. Dazu wird die `equals` Methode der aktuellen `ImageComparator` Implementierung verwendet.

Parameter

`actual`	Das aktuelle `ImageRep` Objekt.
`expected`	Das erwartete `ImageRep` Objekt.
Rückgabewert	True, wenn die Bilder gleich sind, sonst false.

54.9.3 Die `ImageRepDrawer` Klasse

Die Klasse `de.qfs.apps.qftest.shared.extensions.image.ImageRepDrawer` bietet Methoden, um auf einem `ImageRep` Objekt zu zeichnen.

`ImageRepDrawer arrow(int x1, int y1, int x2, int y2)`

Zeichnet einen Pfeil.

Parameter

`x1`	Die x-Anfangskoordinate des Pfeils.
`y1`	Die y-Anfangskoordinate des Pfeils.
`x2`	Die x-Endkoordinate des Pfeils.
`y2`	Die y-Endkoordinate des Pfeils.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer arrow(int x1, int y1, int x2, int y2, int arrowStretch)`

Zeichnet einen Pfeil.

Parameter

`x1`	Die x-Anfangskoordinate des Pfeils.
`y1`	Die y-Anfangskoordinate des Pfeils.
`x2`	Die x-Endkoordinate des Pfeils.
`y2`	Die y-Endkoordinate des Pfeils.
`arrowStretch`	Die Größe des Pfeils.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer arrow(int x1, int y1, int x2, int y2, int arrowStretch, int strokeSize)`

Zeichnet einen Pfeil.

Parameter

`x1`	Die x-Anfangskoordinate des Pfeils.
`y1`	Die y-Anfangskoordinate des Pfeils.
`x2`	Die x-Endkoordinate des Pfeils.
`y2`	Die y-Endkoordinate des Pfeils.
`arrowStretch`	Die Größe des Pfeils.
`strokeSize`	Die Strichdicke des Pfeils.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer arrow(int x1, int y1, int x2, int y2, int arrowStretch, int strokeSize, Color strokeColor)`

Zeichnet einen Pfeil.

Parameter

`x1`	Die x-Anfangskoordinate des Pfeils.
`y1`	Die y-Anfangskoordinate des Pfeils.
`x2`	Die x-Endkoordinate des Pfeils.
`y2`	Die y-Endkoordinate des Pfeils.
`arrowStretch`	Die Größe des Pfeils.
`strokeSize`	Die Strichdicke des Pfeils.
`strokeColor`	Die Strichfarbe des Pfeils.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer arrow(int x1, int y1, int x2, int y2, int arrowStretch, int strokeSize, int r, int g, int b)`

Zeichnet einen Pfeil.

Parameter

`x1`	Die x-Anfangskoordinate des Pfeils.
`y1`	Die y-Anfangskoordinate des Pfeils.
`x2`	Die x-Endkoordinate des Pfeils.
`y2`	Die y-Endkoordinate des Pfeils.
`arrowStretch`	Die Größe des Pfeils.
`strokeSize`	Die Strichdicke des Pfeils.
`r`	Der Rotwert der Strichfarbe des Pfeils.
`g`	Der Grünwert der Strichfarbe des Pfeils.
`b`	Der Blauwert der Strichfarbe des Pfeils.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer arrow(int x1, int y1, int x2, int y2, int arrowStretch, int strokeSize, int r, int g, int b, int a)`

Zeichnet einen Pfeil.

Parameter

`x1`	Die x-Anfangskoordinate des Pfeils.
`y1`	Die y-Anfangskoordinate des Pfeils.
`x2`	Die x-Endkoordinate des Pfeils.
`y2`	Die y-Endkoordinate des Pfeils.
`arrowStretch`	Die Größe des Pfeils.
`strokeSize`	Die Strichdicke des Pfeils.
`r`	Der Rotwert der Strichfarbe des Pfeils.
`g`	Der Grünwert der Strichfarbe des Pfeils.
`b`	Der Blauwert der Strichfarbe des Pfeils.
`a`	Der Alphawert der Strichfarbe des Pfeils.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`BufferedImage asBufferedImage()`

Konvertiere dieses Bild in ein BufferedImage.

Rückgabewert	Dieses Bild als BufferedImage.

`ImageRepDrawer circle(int x, int y, int r)`

Zeichnet ein Kreis.

Parameter

`x`	Die x-Position des Mittelpunkt des Kreises.
`y`	Die y-Position des Mittelpunkt des Kreises.
`r`	Der Radius des Kreises.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer circle(int x, int y, int r, Color color)`

Zeichnet ein Kreis.

Parameter

`x`	Die x-Position des Mittelpunkt des Kreises.
`y`	Die y-Position des Mittelpunkt des Kreises.
`r`	Der Radius des Kreises.
`color`	Die Farbe des Kreises.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer cross(int x, int y)`

Zeichnet ein Kreuz.

Parameter

`x`	Die x-Position des Kreuzes.
`y`	Die y-Position des Kreuzes.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer cross(int x, int y, int size)`

Zeichnet ein Kreuz.

Parameter

`x`	Die x-Position des Kreuzes.
`y`	Die y-Position des Kreuzes.
`size`	Die Größe des Kreuzes.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer draw(Object drawFunction)`

Zeichne auf dem ImageRep Objekt.

Parameter

`drawFunction`	Ein Lamdba Objekt. Das Lambda Objekt kriegt ein java.awt.Graphics2D Objekt als Eingabe welches benutzt werden kann um auf dem Bild zu zeichnen.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer erase(int x, int y, int w, int h)`

”Löscht” einen Bereich des Bildes.

Parameter

`x`	Die x-Position des Bereiches der gelöscht werden soll.
`y`	Die y-Position des Bereiches der gelöscht werden soll.
`w`	Die Breite des Bereiches der gelöscht werden soll.
`h`	Die Höhe des Bereiches der gelöscht werden soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer fillRectangle(int x, int y, int w, int h)`

Zeichnet ein gefülltes Rechteck.

Parameter

`x`	Die x-Position des Rechtecks.
`y`	Die y-Position des Rechtecks.
`w`	Die Breite des Rechtecks.
`h`	Die Höhe des Rechtecks.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer fillRectangle(int x, int y, int w, int h, Color color)`

Zeichnet ein gefülltes Rechteck.

Parameter

`x`	Die x-Position des Rechtecks.
`y`	Die y-Position des Rechtecks.
`w`	Die Breite des Rechtecks.
`h`	Die Höhe des Rechtecks.
`color`	Die Füllfarbe des Rechtecks.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer image(ImageRep imgToDraw, int x, int y)`

Zeichnet ein Bild auf das bereits bestehende Bild.

Parameter

`imgToDraw`	Das zu zeichnende Bild.
`x`	Die x-Position an der das Bild gezeichnet werden soll.
`y`	Die y-Position an der das Bild gezeichnet werden soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer image(Image img, int x, int y)`

Zeichnet ein Bild auf das bereits bestehende Bild.

Parameter

`img`	Das zu zeichnende Bild.
`x`	Die x-Position an der das Bild gezeichnet werden soll.
`y`	Die y-Position an der das Bild gezeichnet werden soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer line(int x1, int y1, int x2, int y2)`

Zeichnet eine Linie.

Parameter

`x1`	Die x-Position der ersten Koordinate der Linie.
`y1`	Die y-Position der ersten Koordinate der Linie.
`x2`	Die x-Position der zweiten Koordinate der Linie.
`y2`	Die y-Position der zweiten Koordinate der Linie.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer line(int x1, int y1, int x2, int y2, Color strokeColor)`

Zeichnet eine Linie.

Parameter

`x1`	Die x-Position der ersten Koordinate der Linie.
`y1`	Die y-Position der ersten Koordinate der Linie.
`x2`	Die x-Position der zweiten Koordinate der Linie.
`y2`	Die y-Position der zweiten Koordinate der Linie.
`strokeColor`	Die Farbe der zu zeichnenden Linie.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer line(int x1, int y1, int x2, int y2, int strokeSize)`

Zeichnet eine Linie.

Parameter

`x1`	Die x-Position der ersten Koordinate der Linie.
`y1`	Die y-Position der ersten Koordinate der Linie.
`x2`	Die x-Position der zweiten Koordinate der Linie.
`y2`	Die y-Position der zweiten Koordinate der Linie.
`strokeSize`	Die Dicke der zu zeichnenden Linie.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer line(int x1, int y1, int x2, int y2, int strokeSize, Color strokeColor)`

Zeichnet eine Linie.

Parameter

`x1`	Die x-Position der ersten Koordinate der Linie.
`y1`	Die y-Position der ersten Koordinate der Linie.
`x2`	Die x-Position der zweiten Koordinate der Linie.
`y2`	Die y-Position der zweiten Koordinate der Linie.
`strokeSize`	Die Dicke der zu zeichnenden Linie.
`strokeColor`	Die Farbe der zu zeichnenden Linie.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer line(int x1, int y1, int x2, int y2, int r, int g, int b)`

Zeichnet eine Linie.

Parameter

`x1`	Die x-Position der ersten Koordinate der Linie.
`y1`	Die y-Position der ersten Koordinate der Linie.
`x2`	Die x-Position der zweiten Koordinate der Linie.
`y2`	Die y-Position der zweiten Koordinate der Linie.
`r`	Der Rotwert der Strichfarbe.
`g`	Der Grünwert der Strichfarbe.
`b`	Der Blauwert der Strichfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer line(int x1, int y1, int x2, int y2, int r, int g, int b, int a)`

Zeichnet eine Linie.

Parameter

`x1`	Die x-Position der ersten Koordinate der Linie.
`y1`	Die y-Position der ersten Koordinate der Linie.
`x2`	Die x-Position der zweiten Koordinate der Linie.
`y2`	Die y-Position der zweiten Koordinate der Linie.
`r`	Der Rotwert der Strichfarbe.
`g`	Der Grünwert der Strichfarbe.
`b`	Der Blauwert der Strichfarbe.
`a`	Der Alphawert der Strichfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer line(int x1, int y1, int x2, int y2, int strokeSize, int r, int g, int b, int a)`

Zeichnet eine Linie.

Parameter

`x1`	Die x-Position der ersten Koordinate der Linie.
`y1`	Die y-Position der ersten Koordinate der Linie.
`x2`	Die x-Position der zweiten Koordinate der Linie.
`y2`	Die y-Position der zweiten Koordinate der Linie.
`strokeSize`	Die Dicke der zu zeichnenden Linie.
`r`	Der Rotwert der Strichfarbe.
`g`	Der Grünwert der Strichfarbe.
`b`	Der Blauwert der Strichfarbe.
`a`	Der Alphawert der Strichfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer pixel(int x, int y)`

Färbt ein bestimmtes Pixel des Bildes mit der früher mithilfe von setStrokeColor gesetzten Farbe. Sollte keine Farbe gesetzt worden sein, wird die Farbe Schwarz genommen.

Parameter

`x`	Die x-Position des zu färbenden Pixel des Bildes.
`y`	Die y-Position des zu färbenden Pixel des Bildes.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer pixel(int x, int y, java.awt.Color color)`

Färbt ein bestimmtes Pixel des Bildes.

Parameter

`x`	Die x-Position des zu färbenden Pixel des Bildes.
`y`	Die y-Position des zu färbenden Pixel des Bildes.
`c`	Die zu setzende Farbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer pixel(int x, int y, int r, int g, int b)`

Färbt ein bestimmtes Pixel des Bildes.

Parameter

`x`	Die x-Position des zu färbenden Pixel des Bildes.
`y`	Die y-Position des zu färbenden Pixel des Bildes.
`r`	Der Rotwert den der zu färbende Pixel haben soll.
`g`	Der Grünwert den der zu färbende Pixel haben soll.
`b`	Der Blauwert den der zu färbende Pixel haben soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer pixel(int x, int y, int r, int g, int b, int a)`

Färbt ein bestimmtes Pixel des Bildes.

Parameter

`x`	Die x-Position des zu färbenden Pixel des Bildes.
`y`	Die y-Position des zu färbenden Pixel des Bildes.
`r`	Der Rotwert den der zu färbende Pixel haben soll.
`g`	Der Grünwert den der zu färbende Pixel haben soll.
`b`	Der Blauwert den der zu färbende Pixel haben soll.
`a`	Der Alphawert den der zu färbende Pixel haben soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer rectangle(int x, int y, int w, int h)`

Zeichne ein Rechteck.

Parameter

`x`	Die x-Position an der ein Rechteck gezeichnet werden soll.
`y`	Die y-Position an der ein Rechteck gezeichnet werden soll.
`w`	Die Breite des zu zeichnenden Rechtecks.
`h`	Die Höhe des zu zeichnenden Rechtecks.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer rectangle(int x, int y, int w, int h, int b)`

Zeichne ein Rechteck.

Parameter

`x`	Die x-Position an der ein Rechteck gezeichnet werden soll.
`y`	Die y-Position an der ein Rechteck gezeichnet werden soll.
`w`	Die Breite des zu zeichnenden Rechtecks.
`h`	Die Höhe des zu zeichnenden Rechtecks.
`b`	Die Strichdicke des zu zeichnenden Rechtecks.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer rectangle(int x, int y, int w, int h, int b, Color strokeColor)`

Zeichne ein Rechteck.

Parameter

`x`	Die x-Position an der ein Rechteck gezeichnet werden soll.
`y`	Die y-Position an der ein Rechteck gezeichnet werden soll.
`w`	Die Breite des zu zeichnenden Rechtecks.
`h`	Die Höhe des zu zeichnenden Rechtecks.
`b`	Die Strichdicke des zu zeichnenden Rechtecks.
`strokeColor`	Die zu verwendende Strichfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer rectangle(int x, int y, int w, int h, int b, Color strokeColor, Color fillColor)`

Zeichne ein Rechteck.

Parameter

`x`	Die x-Position an der ein Rechteck gezeichnet werden soll.
`y`	Die y-Position an der ein Rechteck gezeichnet werden soll.
`w`	Die Breite des zu zeichnenden Rechtecks.
`h`	Die Höhe des zu zeichnenden Rechtecks.
`b`	Die Strichdicke des zu zeichnenden Rechtecks.
`strokeColor`	Die zu verwendende Strichfarbe.
`fillColor`	Die zu verwendende Füllfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer rectangle(int x, int y, int w, int h, int b, int strokeCap, Color strokeColor)`

Zeichne ein Rechteck.

Parameter

`x`	Die x-Position an der ein Rechteck gezeichnet werden soll.
`y`	Die y-Position an der ein Rechteck gezeichnet werden soll.
`w`	Die Breite des zu zeichnenden Rechtecks.
`h`	Die Höhe des zu zeichnenden Rechtecks.
`b`	Die Strichdicke des zu zeichnenden Rechtecks.
`strokeCap`	Die zu verwendenden Strichenden.
`strokeColor`	Die zu verwendende Strichfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer rectangle(int x, int y, int w, int h, int b, int strokeCap, Color strokeColor, Color fillColor)`

Zeichne ein Rechteck.

Parameter

`x`	Die x-Position an der ein Rechteck gezeichnet werden soll.
`y`	Die y-Position an der ein Rechteck gezeichnet werden soll.
`w`	Die Breite des zu zeichnenden Rechtecks.
`h`	Die Höhe des zu zeichnenden Rechtecks.
`b`	Die Strichdicke des zu zeichnenden Rechtecks.
`strokeCap`	Die zu verwendenden Strichenden.
`strokeColor`	Die zu verwendende Strichfarbe.
`fillColor`	Die zu verwendende Füllfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer rectangle(int x, int y, int w, int h, int b, int strokeCap, int rstroke, int gstroke, int bstroke, int rfill, int gfill, int bfill)`

Zeichne ein Rechteck.

Parameter

`x`	Die x-Position an der ein Rechteck gezeichnet werden soll.
`y`	Die y-Position an der ein Rechteck gezeichnet werden soll.
`w`	Die Breite des zu zeichnenden Rechtecks.
`h`	Die Höhe des zu zeichnenden Rechtecks.
`b`	Die Strichdicke des zu zeichnenden Rechtecks.
`strokeCap`	Die zu verwendenden Strichenden.
`rstroke`	Der Rotwert der zu verwendenden Strichfarbe.
`gstroke`	Der Grünwert der zu verwendenden Strichfarbe.
`bstroke`	Der Blauwert der zu verwendenden Strichfarbe.
`rfill`	Der Rotwert der zu verwendenden Füllfarbe.
`gfill`	Der Grünwert der zu verwendenden Füllfarbe.
`bfill`	Der Blauwert der zu verwendenden Füllfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer rectangle(int x, int y, int w, int h, int b, int strokeCap, int rstroke, int gstroke, int bstroke, int astroke, int rfill, int gfill, int bfill, int afill)`

Zeichne ein Rechteck.

Parameter

`x`	Die x-Position an der ein Rechteck gezeichnet werden soll.
`y`	Die y-Position an der ein Rechteck gezeichnet werden soll.
`w`	Die Breite des zu zeichnenden Rechtecks.
`h`	Die Höhe des zu zeichnenden Rechtecks.
`b`	Die Strichdicke des zu zeichnenden Rechtecks.
`strokeCap`	Die zu verwendenden Strichenden.
`rstroke`	Der Rotwert der zu verwendenden Strichfarbe.
`gstroke`	Der Grünwert der zu verwendenden Strichfarbe.
`bstroke`	Der Blauwert der zu verwendenden Strichfarbe.
`astroke`	Der Alphawert der zu verwendenden Strichfarbe.
`rfill`	Der Rotwert der zu verwendenden Füllfarbe.
`gfill`	Der Grünwert der zu verwendenden Füllfarbe.
`bfill`	Der Blauwert der zu verwendenden Füllfarbe.
`afill`	Der Alphawert der zu verwendenden Füllfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setFillColor(java.awt.Color color)`

Setzen der Füllfarbe.

Parameter

`color`	Die neue Farbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setFillColor(int r, int g, int b)`

Setzen der Füllfarbe.

Parameter

`r`	Der Rotwert der Füllfarbe.
`g`	Der Grünwert der Füllfarbe.
`b`	Der Blauwert der Füllfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setFillColor(int r, int g, int b, int a)`

Setzen der Füllfarbe.

Parameter

`r`	Der Rotwert der Füllfarbe.
`g`	Der Grünwert der Füllfarbe.
`b`	Der Blauwert der Füllfarbe.
`a`	Der Alphawert der Füllfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setFont(java.awt.Font font)`

Setzen der Schriftart.

Parameter

`font`	Die zu setzende Schriftart.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setFont(String fontName)`

Setzen der Schriftart.

Parameter

`fontName`	Der Name der zu benutzenden Schrift.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setFont(String fontName, int size)`

Setzen der Schriftart.

Parameter

`fontName`	Der Name der zu benutzenden Schrift.
`size`	Der Größe der zu benutzenden Schrift.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setFont(String fontName, int size, int style)`

Setzen der Schriftart.

Parameter

`fontName`	Der Name der zu benutzenden Schrift.
`size`	Der Größe der zu benutzenden Schrift.
`style`	Die Schriftart der zu benutzenden Schrift.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setStrokeCap(int cap)`

Setzen der Strichenden.

Parameter

`cap`	Kann ImageRepDrawer.CAPS_SQUARED für Striche mit eckigen Enden oder ImageRepDrawer.CAPS_ROUND für Striche mit runden Ecken sein.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setStrokeColor(java.awt.Color color)`

Setzen der Strichfarbe.

Parameter

`color`	Die neue Farbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setStrokeColor(int r, int g, int b)`

Setzen der Strichfarbe.

Parameter

`r`	Der Rotwert der Strichfarbe.
`g`	Der Grünwert der Strichfarbe.
`b`	Der Blauwert der Strichfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setStrokeColor(int r, int g, int b, int a)`

Setzen der Strichfarbe.

Parameter

`r`	Der Rotwert der Strichfarbe.
`g`	Der Grünwert der Strichfarbe.
`b`	Der Blauwert der Strichfarbe.
`a`	Der Alphawert der Strichfarbe.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer setStrokeSize(int size)`

Setzen der Strichdicke.

Parameter

`size`	Die neue Strichdicke.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer text(int x, int y, String text)`

Zeichne einen bestimmten Text.

Parameter

`x`	Die x-Position an der der Text gezeichnet werden soll.
`y`	Die y-Position an der der Text gezeichnet werden soll.
`text`	Der Text der gezeichnet werden soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer text(int x, int y, String text, Color textColor)`

Zeichne einen bestimmten Text.

Parameter

`x`	Die x-Position an der der Text gezeichnet werden soll.
`y`	Die y-Position an der der Text gezeichnet werden soll.
`text`	Der Text der gezeichnet werden soll.
`textColor`	Die Farbe mithilfe der Text gezeichnet werden soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer text(int x, int y, String text, String fontName)`

Zeichne einen bestimmten Text.

Parameter

`x`	Die x-Position an der der Text gezeichnet werden soll.
`y`	Die y-Position an der der Text gezeichnet werden soll.
`text`	Der Text der gezeichnet werden soll.
`fontName`	Der Name der zu benutzenden Schrift.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer text(int x, int y, String text, Color textColor, Font font)`

Zeichne einen bestimmten Text.

Parameter

`x`	Die x-Position an der der Text gezeichnet werden soll.
`y`	Die y-Position an der der Text gezeichnet werden soll.
`text`	Der Text der gezeichnet werden soll.
`textColor`	Die Farbe mithilfe der der Text gezeichnet werden soll.
`font`	Der Font mithilfe der Text gezeichnet werden soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer text(int x, int y, String text, String fontName, int fontSize)`

Zeichne einen bestimmten Text.

Parameter

`x`	Die x-Position an der der Text gezeichnet werden soll.
`y`	Die y-Position an der der Text gezeichnet werden soll.
`text`	Der Text der gezeichnet werden soll.
`fontName`	Der Name der zu benutzenden Schrift.
`fontSize`	Der Größe der zu benutzenden Schrift.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer text(int x, int y, String text, int r, int g, int b)`

Zeichne einen bestimmten Text.

Parameter

`x`	Die x-Position an der der Text gezeichnet werden soll.
`y`	Die y-Position an der der Text gezeichnet werden soll.
`text`	Der Text der gezeichnet werden soll.
`r`	Der Rotwert der Farbe mithilfe der der Text gezeichnet werden soll.
`g`	Der Grünwert der Farbe mithilfe der der Text gezeichnet werden soll.
`b`	Der Blauwert der Farbe mithilfe der der Text gezeichnet werden soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer text(int x, int y, String text, String fontName, int fontSize, int fontStyle)`

Zeichne einen bestimmten Text.

Parameter

`x`	Die x-Position an der der Text gezeichnet werden soll.
`y`	Die y-Position an der der Text gezeichnet werden soll.
`text`	Der Text der gezeichnet werden soll.
`fontName`	Der Name der zu benutzenden Schrift.
`fontSize`	Der Größe der zu benutzenden Schrift.
`fontStyle`	Die Schriftart der zu benutzenden Schrift.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

`ImageRepDrawer text(int x, int y, String text, int r, int g, int b, int a)`

Zeichne einen bestimmten Text.

Parameter	
`x`	Die x-Position an der der Text gezeichnet werden soll.
`y`	Die y-Position an der der Text gezeichnet werden soll.
`text`	Der Text der gezeichnet werden soll.
`r`	Der Rotwert der Farbe mithilfe der der Text gezeichnet werden soll.
`g`	Der Grünwert der Farbe mithilfe der der Text gezeichnet werden soll.
`b`	Der Blauwert der Farbe mithilfe der der Text gezeichnet werden soll.
`a`	Der Alphawert der Farbe mithilfe der der Text gezeichnet werden soll.
Rückgabewert	Das ImageRepDrawer Objekt für Methodenkontaktion.

Im folgenden werden einige Beispiele für das Zeichnen auf einem ImageRep Objekt mithilfe der ImageRepDrawer Methode gezeigt:

```
from imagewrapper import ImageWrapper
from java.awt import Color
iw = ImageWrapper(rc)
img = iw.loadPng(r"C:/temp/foobar.png")
img.draw().setStrokeSize(12).setStrokeColor(Color.RED).line(20, 20, 2000, 3500) \
    .setStrokeColor(Color.GREEN).setStrokeSize(1).rectangle(40, 45, 250, 300)
rc.logImage(img)
```

Beispiel 54.39: Benutzung des ImageRepDrawer Objekts um eine rote Linie und ein grünes Rechteck auf ein Bild zu malen.

54.10 Pseudo DOM API für Web-Anwendungen

Web

QF-Test bietet SUT-Skript[720] Knoten Zugriff auf einen Teil der DOM-API einer Web-Anwendung. Diese API entspricht nicht ganz der auf JavaScript-Ebene, auf welcher mittels der in diesem Kapitel beschriebenen Methoden `toJS`, `callJS` und `evalJS` mit der Web-Anwendung interagiert werden kann. Mit der DOM-API ist es möglich, das DOM zu traversieren und Attribute von DOM-Knoten auszulesen und zu setzen, aber nicht die Struktur des DOM zu verändern. Diese API ist damit hilfreich für die Implementierung von `Name-` oder `FeatureResolvern` (vgl. Abschnitt 54.1[1154]).

Bei Swing, FX und SWT arbeitet QF-Test mit den konkreten Java-GUI-Klassen, während bei Web-Anwendungen folgende pseudo Klassenhierarchie verwendet wird:

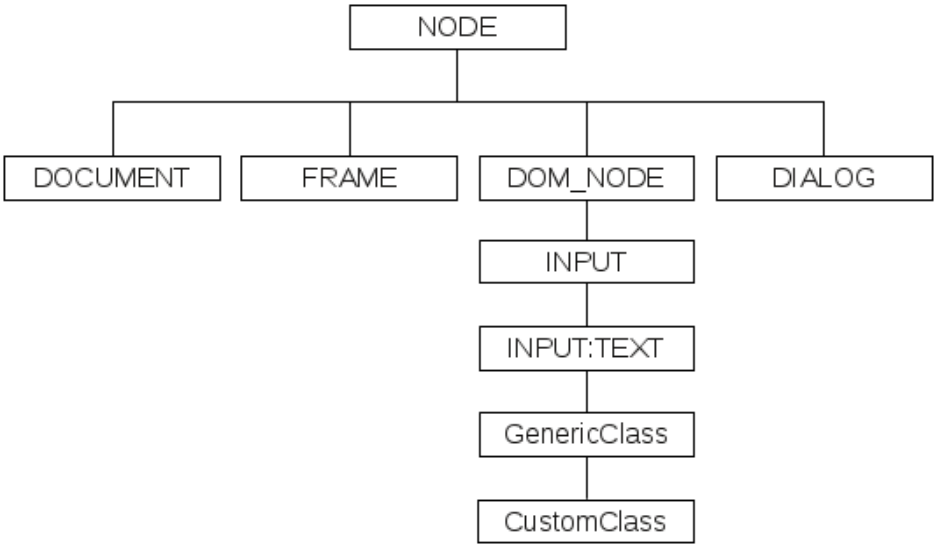


Abbildung 54.1: Pseudo Klassenhierarchie für Elemente von Web-Anwendungen

Wie zu sehen, ist ”NODE” die Wurzel der pseudo Klassenhierarchie. Diese Klasse passt auf jedes Element des DOM. Von ”NODE” abgeleitet sind ”DOCUMENT”, ”FRAME”, ”DOM_NODE” und ”DIALOG”, die Knotentypen, die die Pseudo-DOM-API implementieren, welches in Abschnitt 54.10(1253) beschrieben ist. ”DOM_NODE” wird an Hand des Tag-Namens des Knotens weiter spezialisiert, z.B: ”H1”, ”A” oder ”INPUT”, wobei es für manche Tags zusätzliche Unterklassen wie ”INPUT:TEXT” gibt.

Hinweis

Das DOM kann für verschiedene Browser unterschiedlich ausfallen. Daher sollten Sie versuchen, sich in Ihren Skripten und Resolvern nicht auf Kindindizes zu verlassen, falls Sie beabsichtigen mit mehreren Browsern zu testen.

Die DOM-API von QF-Test besteht aus den folgenden fünf Klassen:

54.10.1 Die abstrakte Klasse `Node`

Alle Klassen von QF-Test’s pseudo DOM API sind von dieser Klasse abgeleitet und implementieren damit das folgende Interface. Die Klasse befindet sich im Package `de.qfs.apps.qftest.client.web.dom`.

`Node findCommonAncestor(Node node1, Node node2, Node topmost)`

Liefert einen gemeinsamen Vorgänger von zwei Knoten.

Parameter

`node1`	Der erste Knoten.
`node2`	Der zweite Knoten.
`topmost`	Der oberste Knoten, bei dem die Suche stoppen soll.
Rückgabewert	Der gemeinsame Vorgänger oder null.

`Node getAncestorOfClass(String clazz)`

Liefert die nächsten Vorgänger dieser Komponente von einer angegebenen Klasse.

Parameter

`clazz`	Das Klassenname des Parents.
Rückgabewert	Der passende Vorgänger oder null.

`Node getAncestorOfClass(String clazz, int maxDepth)`

Liefert die nächsten Vorgänger dieser Komponente von einer angegebenen Klasse unter Angabe einer maximalen Suchhöhe.

Parameter

`clazz`	Das Klassenname des Parents.
`maxDepth`	Das maximale Suchlevel.
Rückgabewert	Der passende Vorgänger oder null.

`String getAttribute(String name)`

Liefert den Wert eines Attributs des Knotens. Zur bequemeren Nutzung ist diese Methode bereits auf dieser Ebene für `Node` definiert. Falls es sich bei einem konkreten Knoten nicht um ein Objekt der Klasse `DomNode` handelt, ist das Ergebnis immer null. In einigen Fällen liefert diese Methode Werte für Attribute zurück, die nicht explizit angegeben wurden, z.B. die Attribute ”width” und ”height” für einen IMG Knoten. Welche Attribute implizit zur Verfügung stehen, hängt vom jeweiligen Browser ab.

Parameter

`name`	Der Name des Attributs.
Rückgabewert	Der Wert des Attributs oder null, falls es für diesen Knoten kein solches Attribut gibt.

`String getAttributeIfSpecified(String name)`
Liefert den Wert eines Attributs, sofern dieses explizit im HTML-Code spezifiziert wurde. Zur bequemeren Nutzung ist diese Methode bereits auf dieser Ebene für `Node` definiert. Falls es sich bei einem konkreten Knoten nicht um ein Objekt der Klasse `DomNode` handelt, ist das Ergebnis immer null.

Parameter

`name`	Der Name des Attributs.
Rückgabewert	Der Wert des Attributs oder null, falls für diesen Knoten kein solches Attribut explizit spezifiziert wurde.

`Node getChild(int index)`
Liefert den Kind-Knoten an einem bestimmten Index. Hinweis für Opera vor Version 86, Google Chrome und Microsoft Edge vor Version 100: Text-Knoten, die nur zur Formatierung des HTML-Codes dienen und keinen Einfluss auf die Darstellung der Webseite haben, werden im CDP-Verbindungsmodus nicht vom Browser an QF-Test übermittelt. Daher kann in diesem Verbindungsmodus mit diesen Browsern eine abweichende Zählung der Kind-Knoten auftreten. Prinzipiell ist hier ein Zugriff über `getChildrenByTagName` zu bevorzugen.

Parameter

`index`	Der Index des Kind-Knotens, beginnend mit 0.
Rückgabewert	Der Kind-Knoten an dem angegebenen Index.

Exceptions

`IllegalArgumentExceptio`	Falls der Index negativ oder größer als die Zahl der Kind-Knoten ist.

`int getChildCount()`
Liefert die Anzahl an Kind-Knoten. Hinweis für Opera vor Version 86, Google Chrome und Microsoft Edge vor Version 100: Text-Knoten, die nur zur Formatierung des HTML-Codes dienen und keinen Einfluss auf die Darstellung der Webseite haben, werden im CDP-Verbindungsmodus nicht vom Browser an QF-Test übermittelt. Daher kann in diesem Verbindungsmodus mit diesen Browsern eine abweichende Anzahl der Kind-Knoten auftreten.

Rückgabewert	Die Anzahl an Kind-Knoten.

`Node[] getChildren()`
Liefert alle Kinder dieses Knotens als Array. Hinweis für Opera vor Version 86, Google Chrome und Microsoft Edge vor Version 100: Text-Knoten, die nur zur Formatierung des HTML-Codes dienen und keinen Einfluss auf die Darstellung der Webseite haben, werden im CDP-Verbindungsmodus nicht vom Browser an QF-Test übermittelt. Daher kann in diesem Verbindungsmodus mit diesen Browsern eine abweichende Anzahl Kind-Knoten auftreten.

Rückgabewert	Die Kind-Knoten.

`String getClassName()`

Liefert den aktuellsten Klassennamen des Knotens.

Rückgabewert Die Name der aktuellsten Klasse des Knotens.

`String[] getClassNames()`

Liefert alle Klassen des Knotens.

Rückgabewert Ein Array mit allen Klassen des Knotens.

`DocumentNode getDocument()`

Liefert das Dokument, zu dem dieser Knoten gehört.

Rückgabewert Das Dokument, zu dem dieser Knoten gehört. Ein `DocumentNode` liefert sich selbst, ein `DialogNode` liefert null.

`Node getElementById(String id)`

Liefert einen direkten oder indirekten Kind-Knoten mit einer bestimmten ID. Verglichen wird hierbei nicht mit dem Original-Attribut 'ID', sondern der möglicherweise durch `IdResolver` modifizierten ID (vgl. Abschnitt 54.1.17(1180)).

Parameter

`id` Die gesuchte ID.

Rückgabewert Der Kind-Knoten mit der vorgegebenen ID, ein beliebiger im Fall mehrerer Treffer oder null falls keiner gefunden wird. Ein `DialogNode` liefert immer null, ein `FrameNode` Knoten reicht den Aufruf an seinen `DocumentNode` Knoten weiter und ein `DocumentNode` Knoten an seinen HTML Wurzelknoten.

`Node[] getElementsByClassName(String className)`

Liefert alle direkten oder indirekten Kind-Knoten mit einer bestimmten Klasse.

Parameter

`className` Der Klassenname.

Rückgabewert Ein Array von Kind-Knoten mit der vorgegebenen Klasse. Werden keine gefunden, wird ein leeres Array zurückgeliefert.

`Node[] getElementsById(String id)`
Liefert alle direkten oder indirekten Kind-Knoten mit einer bestimmten ID. Verglichen wird hierbei nicht mit dem Original-Attribut 'ID' , sondern der möglicherweise durch `IdResolver` modifizierten ID (vgl. Abschnitt 54.1.17[1180]).

Parameter

`id`	Die gesuchte ID.
Rückgabewert	Ein Array von Kind-Knoten mit der vorgegebenen ID. Werden keine gefunden, wird ein leeres Array zurückgeliefert. Ein `DialogNode` liefert immer ein leeres Array, ein `FrameNode` Knoten reicht den Aufruf an seinen `DocumentNode` Knoten weiter und ein `DocumentNode` Knoten an seinen HTML Wurzelknoten.

`Node[] getElementsByIdAndTagname(String id, String tagName)`
Liefert alle direkten oder indirekten Kind-Knoten mit einem bestimmten TagNamen und einer bestimmten ID. Verglichen wird hierbei nicht mit dem Original-Attribut 'ID', sondern der möglicherweise durch `IdResolver` modifizierten ID (vgl. Abschnitt 54.1.17[1180]).

Parameter

`id`	Die gesuchte ID.
`tagName`	Der gesuchte Tagname.
Rückgabewert	Ein Array von Kind-Knoten mit der vorgegebenen ID und dem vorgegebenen Tagnamen. Werden keine gefunden, wird ein leeres Array zurückgeliefert. Ein `DialogNode` liefert immer ein leeres Array, ein `FrameNode` Knoten reicht den Aufruf an seinen `DocumentNode` Knoten weiter und ein `DocumentNode` Knoten an seinen HTML Wurzelknoten.

`Node[] getElementsByTagName(String tagName)`
Liefert alle direkten oder indirekten Kind-Knoten mit einem bestimmten Tagnamen.

Parameter

`tagName`	Der gesuchte Tagname.
Rückgabewert	Ein Array von Kind-Knoten mit dem vorgegebenen Tagnamen. Werden keine gefunden, wird ein leeres Array zurückgeliefert. Ein `DialogNode` liefert immer ein leeres Array, ein `FrameNode` Knoten reicht den Aufruf an seinen `DocumentNode` Knoten weiter und ein `DocumentNode` Knoten an seinen HTML Wurzelknoten.

`Node getFirstChild()`
Liefert den ersten Kind-Knoten: ein Element, ein Text-Knoten oder ein Kommentar.

Rückgabewert	Der erste Kind-Knoten.

`Node getFirstElementChild()`

Liefert das erste Kind-Element (Text-Knoten und Kommentare werden übersprungen).

Rückgabewert Das erste Kind-Element.

`String getFlatText()`

Liefert den ”flachen” Textinhalt eines Knotens. Für einen einfachen Text-Knoten ist dies sein Wert. Andernfalls werden die Werte aller direkt in diesem Knoten enthaltenen Text-Knoten gesammelt. Knoten in darunter liegenden Ebenen werden nicht berücksichtigt.

Rückgabewert Der gesamte direkte Textinhalt eines Knotens.

`String getGenericClassName()`

Liefert den generischen Klassenname des Knotens.

Rückgabewert Der Name der generischen Klasse.

`int getIndexOfChild(Node child)`

Liefert den Index eines bestimmten Kind-Knotens.

Parameter

`child` Der Kind-Knoten dessen Index ermittelt werden soll.

Rückgabewert Der Index des Kind-Knoten, beginnend bei 0, oder -1 falls es sich dabei nicht um ein Kind dieses Knotens handelt.

`Node getInterestingParent()`

Liefert den interessanten Parent eines Knoten. Das ist der Parent, der auch als Parent aufgezeichnet wird. Ein `DocumentNode` auf oberster Ebene oder ein `DialogNode` liefern null. Alle anderen Knoten sollten einen Parent haben, sofern sie nicht via JavaScript aus dem DOM entfernt wurden.

Rückgabewert Der interessante Parent des Knoten.

`Node getInterestingParent(int n)`

Liefert den interessanten Parent eines Knoten. Das ist der Parent, der auch als Parent aufgezeichnet wird. Ein `DocumentNode` auf oberster Ebene oder ein `DialogNode` liefern null. Alle anderen Knoten sollten einen Parent haben, sofern sie nicht via JavaScript aus dem DOM entfernt wurden.

Parameter

`n` Das Level des interessanten Parents.

Rückgabewert Der interessante Parent des Knoten.

`String getName()`

Liefert den *Tagnamen* eines Knoten. Dieser Name entspricht dem Type des Knotens in Großbuchstaben, z.B: ”HTML” für einen HTML Knoten. Einfache Text-Knoten werden als `DomNode` mit dem Tagnamen ”#text” repräsentiert. Pseudo-Tagnamen sind für `DocumentNodes` (”DOCUMENT”) und `DialogNodes` (”DIALOG”) definiert.

Rückgabewert Der Tagname eines Knotens.

`Node getNextElementSibling()`

Liefert das nachfolgende Element auf der gleichen Ebene des DOM-Baums.

Rückgabewert Das nachfolgende Element.

`Node getNextSibling()`

Liefert den nachfolgenden Knoten: ein Element, ein Text-Knoten oder ein Kommentar.

Rückgabewert Der nachfolgende Knoten.

`String getNodeType()`

Liefert einen Bezeichner für die Art des Knotens. Diese Methode ist zwar nicht ganz im Sinne einer puristischen OO Doktrin, beim Traversieren des DOM aber sehr praktisch, um ohne Aufwand die Art des jeweiligen Knotens zu bestimmen.

Rückgabewert Ein String der die Art des Knoten angibt. Die zugehörigen Konstanten sind in den konkreten Unterklassen definiert:`DocumentNode.DOCUMENT_NODE`, `FrameNode.FRAME_NODE`, `DomNode.DOM_NODE` und `DialogNode.DIALOG_NODE`.

`Node getNthParent(int n)`

Liefert den Parent vom n-tn Level.

Parameter

`n` Das Level des Parents.

Rückgabewert Der Parent oder null.

`Node getParent()`

Liefert den Parent eines Knoten. Ein `DocumentNode` auf oberster Ebene oder ein `DialogNode` liefern null. Alle anderen Knoten sollten einen Parent haben, sofern sie nicht via JavaScript aus dem DOM entfernt wurden.

Rückgabewert Der Parent des Knoten.

`Node getPreviousElementSibling()`

Liefert das vorangehende Element auf der gleichen Ebene des DOM-Baums.

Rückgabewert Das vorangehende Element.

`Node getPreviousSibling()`

Liefert den vorangehenden Knoten: ein Element, ein Text-Knoten oder ein Kommentar.

Rückgabewert Der vorangehende Knoten.

`Object getProperty(String name)`

Liefert eine benutzerdefinierte Eigenschaft zurück.

Parameter

`name` Der Name der Eigenschaft.

Rückgabewert Der Wert der Eigenschaft oder null.

`String getSimpleText()`
Liefert den ”einfachen” Textinhalt eines Knotens. Für einen einfachen Text-Knoten ist dies sein Wert. Andernfalls wird das DOM ab diesem Knoten traversiert und die Werte aller Text-Knoten gesammelt, die direkt in diesem Knoten, oder in ”einfachen” Knoten wie B enthalten sind. Strukturell komplexere Knoten wie TABLE werden für die Suche nicht betreten.

Rückgabewert	Der einfache Textinhalt eines Knotens.

`String getText()`
Liefert den Textinhalt eines Knotens. Für einen einfachen Text-Knoten ist dies sein Wert. Andernfalls wird das DOM ab diesem Knoten traversiert und die Werte aller verschachtelten Text-Knoten gesammelt. Dabei wird darauf geachtet, Leerraum möglichst so wiederzugeben, wie er im Browser dargestellt wird.

Rückgabewert	Der gesamte direkte und indirekte Textinhalt eines Knotens.

`String getVisibleFlatText()`
Liefert den ”flachen” Textinhalt eines Knotens. Für einen einfachen Text-Knoten ist dies sein Wert. Andernfalls werden die Werte aller direkt in diesem Knoten enthaltenen Text-Knoten gesammelt. Unsichtbare Knoten und Knoten in darunter liegenden Ebenen werden nicht berücksichtigt.

Rückgabewert	Der gesamte direkte sichtbare Textinhalt eines Knotens.

`String getVisibleSimpleText()`
Liefert den ”einfachen” Textinhalt eines Knotens. Für einen einfachen Text-Knoten ist dies sein Wert. Andernfalls wird das DOM ab diesem Knoten traversiert und die Werte aller sichtbaren Text-Knoten gesammelt, die direkt in diesem Knoten, oder in ”einfachen” Knoten wie B enthalten sind. Strukturell komplexere Knoten wie TABLE werden für die Suche nicht betreten.

Rückgabewert	Der sichtbare einfache Textinhalt eines Knotens.

`String getVisibleText()`
Liefert den Textinhalt eines Knotens. Für einen einfachen Text-Knoten ist dies sein Wert. Andernfalls wird das DOM ab diesem Knoten traversiert und die Werte aller verschachtelten Text-Knoten gesammelt, wobei unsichtbare Knoten ignoriert werden. Dabei wird darauf geachtet, Leerraum möglichst so wiederzugeben, wie er im Browser dargestellt wird.

Rückgabewert	Der gesamte direkte und indirekte sichtbare Textinhalt eines Knotens.

`boolean isAncestor(Node node)`

Prüft, ob ein angegebener Knoten ein Vorgänger des aktuellen Knotens ist.

Parameter

`node`	Der vermeintliche Vorgängerknoten.
Rückgabewert	True, falls der Knoten ein Vorgänger ist, sonst false.

`boolean isAttributeSpecified(String name)`
Prüft, ob ein Attribut eines Knotens explizit im HTML-Code spezifiziert wurde. Zur bequemeren Nutzung ist diese Methode bereits auf dieser Ebene für `Node` definiert. Falls es sich bei einem konkreten Knoten nicht um ein Objekt der Klasse `DomNode` handelt, ist das Ergebnis immer false.

Parameter

`name`	Der Name des Attributs.
Rückgabewert	True falls das Attribut für diesen Knoten explizit spezifiziert wurde.

`boolean isBrowserChrome()`
Prüft, ob der Browser, zu dem dieser Knoten gehört, eine Chrome Variante ist.

Rückgabewert	True für Chrome Varianten, ansonsten false.

`boolean isBrowserHeadless()`
Prüft, ob der Browser, zu dem dieser Knoten gehört, eine headless Browser ist.

Rückgabewert	True für Headless Browser, ansonsten false.

`boolean isBrowserMozilla()`
Prüft, ob der Browser, zu dem dieser Knoten gehört, eine Mozilla Variante ist.

Rückgabewert	True für Mozilla Varianten, ansonsten false.

`boolean isBrowserSafari()`
Prüft, ob der Browser, zu dem dieser Knoten gehört, ein Safari Browser ist.

Rückgabewert	True für Safari Browser, ansonsten false.

`boolean isMatchingClass(String className)`
Prüft, ob der Knoten von der angegebenen Klasse ist.

Parameter

`className`	Der zu prüfende Klassenname.
Rückgabewert	True, wenn der Knoten diese Klasse besitzt, sonst false.

`void setProperty(String name, Object value)`
Setzt eine benutzerdefinierte Eigenschaft.

Parameter

`name`	Der Name der Eigenschaft.
`value`	Der Wert der Eigenschaft oder null um eine Eigenschaft wieder zu entfernen.

54.10.2 Die Klasse `DocumentNode`

Web

Das Wurzel-`Document` einer Webseite wird nicht durch einen Fenster Knoten, sondern

den speziellen Knoten Webseite[925] repräsentiert. Verschachtelte `Document` Knoten in `Frames` entsprechen Komponenten Knoten.

Die Klasse `DocumentNode` ist von `Node` abgeleitet und befindet sich ebenfalls im Package `de.qfs.apps.qftest.client.web.dom`. Zusätzlich zu den in der `Node` Klasse definierten Methoden, die oben beschrieben wurden, bietet `DocumentNode` folgendes an:

`Object callJS(String code)`
Führt JavaScript Code im Context dieses Dokuments innerhalb einer Funktion aus. Dies funktioniert häufig auch dann, wenn ein Aufruf von `eval()` durch eine Sicherheitsrichtlinie für den Inhalt (CSP) unterbunden wird.

Parameter

`code`	Der auszuführende Code.
Rückgabewert	Was immer der Code explizit mit `return` zurückliefert, konvertiert in einen passenden Objekttyp. Sogar die Rückgabe von DOM-Knoten, Frames oder Dokumenten funktioniert.

`Object evalJS(String script)`
Führt JavaScript Code mit einem `window.eval()`-Aufruf im Kontext dieses Dokuments aus. In den meisten Fällen ist die Methode `callJS` zu bevorzugen, da der Aufruf von `eval()` je nach Dokument durch eine Sicherheitsrichtlinie für den Inhalt (CSP) unterbunden sein kann.

Parameter

`script`	Das auszuführende Skript.
Rückgabewert	Was immer das Skript zurückliefert, konvertiert in einen passenden Objekttyp. Sogar die Rückgabe von DOM-Knoten, Frames oder Dokumenten funktioniert.

`FrameNode[] getFrames()`

Liefert die Kind-Frames des Dokuments.

Rückgabewert	Die Kind-Frames des Dokuments, ein leeres Array falls das Dokument keine Frames hat.

`DomNode getRootElement()`

Liefert den HTML Wurzelknoten des Dokuments.

Rückgabewert	Der Wurzelknoten des Dokuments.

`String getSourcecode()`
Liefert den HTML-Quelltext des Dokuments im aktuellen Zustand, also nicht notwendigerweise das, was beim Öffnen des Dokuments geladen wurde, sondern inklusive aller Änderungen an Attributen oder der Struktur des DOM, die seither vorgenommen wurden, z.B. via JavaScript.

Rückgabewert	Der aktuelle HTML-Quelltext des Dokuments.

`String getTitle()`
Liefert den Titel des Dokuments aus dem TITLE Knoten im HEAD Knoten des Wurzelknotens.

Rückgabewert	Der Titel des Dokuments.

`String getUrl()`

Liefert die URL des Dokuments.

Rückgabewert	Die URL des Dokuments.

`boolean hasParent()`
Um zu prüfen, ob ein Dokument sich auf oberster Ebene befindet, sollte diese Methode verwendet werden, anstatt das Ergebnis von `getParent()` auf null zu testen. Der Grund dafür liegt darin, dass das Laden von Kind-Dokumenten vor dem Laden des Parent-Dokument abgeschlossen sein kann. In der Zwischenzeit ist bereits bekannt, dass das Kind-Dokument einen Parent haben wird, obwohl dieser noch nicht zur Verfügung steht.

Rückgabewert	True falls das Dokument einen Parent besitzt, false wenn es sich um ein Dokument auf oberster Ebene handelt.

Neben den obigen Instanz-Methoden bietet die `DocumentNode` Klasse einige statische Methoden, um URL einfacher manipulieren zu können.

`static String getUrlBase(String url)`
Liefert den Basis-Teil einer URL inklusive Host und Verzeichnis, aber ohne eventuelle Parameter, abgetrennt durch ein '?' und ohne eine eventuelle Tomcat Session ID.

Parameter

`url`	Die URL deren Basis-Teil ermittelt werden soll.
Rückgabewert	Der Basis-Teil der URL.

`static String getUrlHost(String url)`

Liefert den Host-Teil einer URL, also den Teil zwischen "http(s)://" und dem nächsten '/'.

Parameter

`url`	Die URL deren Host-Teil ermittelt werden soll.
Rückgabewert	Der Host-Teil der URL oder null, falls das Protokoll weder http noch https ist.

`static ExtraFeatureSet getUrlParameters(String url)`
Liefert die Parameter einer URL, sowohl die durch ein '?' abgetrennten, als auch eine mögliche Tomcat Session ID, in Form eines `ExtraFeatureSet` wie es für einen `ExtraFeatureResolver` benötigt wird (vgl. Abschnitt 54.1.11[1168]).

Parameter

`url`	Die URL deren Parameter ermittelt werden sollen.
Rückgabewert	Die Parameter der URL.

`static String normalizeUrl(String url)`
Normalisiert eine URL. Dabei werden '\' durch '/' ersetzt, "file:/(/(/))" durch "file:///", Windows Laufwerk-Buchstaben groß geschrieben und %.. Sequenzen dekodiert.

Parameter

`url`	Die zu normalisierende URL.
Rückgabewert	Die normalisierte URL.

54.10.3 Die Klasse `FrameNode`

Die Klasse `FrameNode` ist von `Node` abgeleitet und befindet sich ebenfalls im Package `de.qfs.apps.qftest.client.web.dom`. Zusätzlich zu den in der `Node` Klasse definierten Methoden, die oben beschrieben wurden, bietet `FrameNode` folgendes an:

`DocumentNode getChildDocument()`

Liefert das Kind-Dokument des Frames.

Rückgabewert	Das Kind-Dokument des Frames.

`DomNode getFrameElement()`
Liefert, falls bekannt, das `DomNode`-Objekt, welches das Frame enthält, also den IFRAME- oder FRAME-Knoten.

Rückgabewert	Der `DomNode` oder `null`.

`String getFrameName()`

Liefert den Namen des Frames, der durch sein Attribut "name" bestimmt wird.

Rückgabewert	Der Name des Frames.

`String getFrameUrl()`
Liefert die URL des Frames, die normalerweise die gleiche sein sollte, wie die URL seines Kind-Dokuments.

Rückgabewert	Die URL des Frames.

`int[] getGeometry()`
Liefert Position und Größe des Frames relativ zu seinem Parent-Frame oder der Anzeigefläche des Browser-Fensters.

Rückgabewert	Die Geometrie des Frames in der Form [x, y, width, height].

54.10.4 Die Klasse `DomNode`

Die Klasse `DomNode` ist von `Node` abgeleitet und befindet sich ebenfalls im Package `de.qfs.apps.qftest.client.web.dom`. Zusätzlich zu den in der `Node` Klasse definierten Methoden, die oben beschrieben wurden, bietet `DomNode` folgendes an:

`Object callJS(String code)`

Führt JavaScript Code im Context des Dokuments dieses Knotens aus. Dies funktioniert häufig auch dann, wenn ein Aufruf von `eval()` durch eine Sicherheitsrichtlinie für den Inhalt (CSP) unterbunden wird.

Parameter

`code`	Der Code, der in einer Funktion ausgeführt wird. `_qf_node` ist dabei das Objekt, worüber auf das HTML-Element zugegriffen werden kann.
Rückgabewert	Was immer der Code explizit mit `return` zurückliefert, konvertiert in einen passenden Objekttyp. Sogar die Rückgabe von DOM-Knoten, Frames oder Dokumenten funktioniert.

`Object evalJS(String script)`

Führt JavaScript Code mit einem `window.eval()`-Aufruf im Kontext des Dokuments dieses Knotens aus. In den meisten Fällen ist die Methode `callJS` zu bevorzugen, da der Aufruf von `eval()` je nach Dokument durch eine Sicherheitsrichtlinie für den Inhalt (CSP) unterbunden sein kann.

Parameter

`script`	Das auszuführende Skript. `_qf_node` ist dabei das Objekt, worüber auf das HTML-Element zugegriffen werden kann.
Rückgabewert	Was immer das Skript zurückliefert, konvertiert in einen passenden Objekttyp. Sogar die Rückgabe von DOM-Knoten, Frames oder Dokumenten funktioniert.

`DomNode[] getAllByCSS(String css)`

Liefert alle Knoten, die mittels CSS-Selector gefunden werden.

Parameter

`css`	Der CSS-Selector ab diesem Knoten.
Rückgabewert	Ein Array der gefundenen Knoten oder null.

`DomNode[] getAllByXPath(String xpath)`

Liefert alle Knoten, die mittels XPath gefunden werden.

Parameter

`xpath`	Der XPath ab diesem Knoten.
Rückgabewert	Ein Array der gefundenen Knoten oder null.

`DomNode getByCSS(String css)`

Liefert einen Knoten, der mittels CSS-Selector gefunden wird.

Parameter

`css`	Der CSS-Selector ab diesem Knoten.
Rückgabewert	Der gefundene Knoten oder null.

`DomNode getByXPath(String xpath)`

Liefert einen Knoten, der mittels XPath gefunden wird.

Parameter

`xpath`	Der XPath ab diesem Knoten.
Rückgabewert	Der gefundene Knoten oder null.

`Node[] getChildrenByTagName(String tagName)`

Liefert alle direkten Kinder des Knotens mit einem vorgegebenen Tagnamen.

Parameter

`tagName`	Der zu suchende Tagname.
Rückgabewert	Ein Array mit den Kind-Knoten mit dem vorgegebenen Tagnamen. Werden keine gefunden, wird ein leeres Array zurückgeliefert.

`String getId()`

Liefert die gecachte Version der ID des Knotens. Diese Version wurde eventuell durch einen `IdResolver` modifiziert (vgl. Abschnitt 54.1.17[1180]). Das originale, unverfälschte Attribut 'ID' kann mittels `getAttribute("id")` bestimmt werden.

Rückgabewert	Die gecachte ID des Knotens.

`int[] getLocationOnScreen()`

Liefert Position und Größe des Knotens relativ zum Desktop.

Rückgabewert	Die Geometrie des Knotens in der Form [x, y, width, height].

`boolean hasCSSClass(String cl)`

Prüft, ob der Knoten eine angegebene CSS Klasse besitzt.

Parameter

`cl`	Der zu prüfende CSS-Klassenname.
Rückgabewert	True, wenn der Knoten die Klasse besitzt, sonst false.

`boolean hasFocus()`

Prüft, ob der Knoten den Tastatur-Fokus besitzt.

Rückgabewert	True falls der Knoten den Tastatur-Fokus besitzt.

`boolean isShowing()`
Prüft, ob der Knoten aktuell sichtbar ist oder durch Scrollen sichtbar gemacht werden kann.

Rückgabewert	True falls der Knoten sichtbar ist, false falls er nicht sichtbar ist oder sichtbar gescrollt werden kann.

`void requestFocus()`
Fordert den Tastatur-Fokus für den Knoten an. Ob der Knoten den Tastatur-Fokus tatsächlich erhält, hängt allerdings von Browser und Betriebssystem ab.

`void scrollVisible()`
Versucht die Webseite oder einen Frame so zu scrollen, dass der Knoten komplett sichtbar ist. Ob und wie dies geschieht, hängt allerdings von Browser und Betriebssystem ab.

`void setAttribute(String name, String value)`

Setzt ein Attribut des Knotens.

Parameter

`name`	Der Name des Attributs.
`value`	Der zu setzende Wert.

`void toJS(String name)`
Setzt eine JavaScript Variable im Kontext des Dokuments dieses Knotens auf diesen Knoten.

Parameter

`name`	Der Name der zu setzenden Variable.

54.10.5 Die Klasse `DialogNode`

Die Klasse `DialogNode`, ebenfalls von `Node` abgeleitet, ist keine Standard-Klasse des DOM, sondern dient zum einfachen Zugriff auf Dialog in QF-Test. Sie befindet sich ebenfalls im Package `de.qfs.apps.qftest.client.web.dom`. Ein `DialogNode` repräsentiert einen Meldungs- oder Fehlerdialog, wie er in JavaScript erzeugt werden kann. Die Klasse besitzt die folgenden Methoden:

`long getStyle()`
Liefert die Art des Dialogs.

Rückgabewert	Die Art des Dialogs. Mögliche Werte sind die Konstanten `STYLE_ALERT`, `STYLE_CONFIRM`, `STYLE_AUTHENTICATE` oder `STYLE_DOWNLOAD`, die in der `DialogNode` Klasse definiert sind.

`String getText()`

Liefert den Meldungstext des Dialogs.

Rückgabewert	Der Meldungstext des Dialogs.

`String getTitle()`

Liefert den Titel des Dialogs.

Rückgabewert	Der Titel des Dialogs.

54.11 WebDriverConnection SUT API

4.1+

Die WebDriverConnection-SUT-API stellt Klassen und Interfaces bereit, um die Selenium WebDriver Java-API auch innerhalb von SUT-Skripte(720) benutzbar zu machen. Mit dieser Art Bridge ist es möglich, bestehende Selenium WebDriver Skripte innerhalb eines SUT-Skript(720) Knotens von QF-Test zu verwenden. Darüber hinaus kann sogar die Pseudo DOM-API (Abschnitt 54.10(1253)) mit Selenium WebDriver Skripten in Kombination verwendet werden.

Hinweis Diese API kann nur verwendet werden, wenn der Browser über den Verbindungsweg "WebDriver" angebunden ist. Aufrufe auf dem zurückgegebenen WebDriver-Objekt werden automatisch synchronisiert und mit einem Timeout versehen.

```
from webdriver import WebDriverConnection
from org.openqa.selenium import By
wdc = WebDriverConnection(rc)
driver = wdc.getDriver()
# driver ist nun ein Objekt vom Typ org.openqa.selenium.WebDriver
element = driver.findElement(By.cssSelector(".myClass"))
# element ist nun vom Typ org.openqa.selenium.WebElement
# Auf dem Element können direkt die WebDriver Methoden aufgerufen werden
element.click()
# Objekte vom Typ WebElement können auf Objekte
# der QF-Test Pseudo DOM-API abgebildet werden
node = wdc.getComponent(element)
# Und dann einer im Komponenten-Baum definierten Komponente zugewiesen werden
rc.overrideElement("Your-QF-Test-Id",node)
# Ebenfalls kann eine QF-Test Komponente auf ein WebElement abgebildet werden
node = rc.getComponent("QF-Test-Id-Of-Some-Textfield")
element = wdc.getElement(node)
# und dann damit interagiert:
element.clear()
```

Beispiel 54.40: WebDriver Verwendung in einem Jython SUT Script

Hinweis Das WebDriver-Objekt ist um Methoden zur Kontrolle des automatischen Timeouts er-

weitert.

```
import de.qfs.WebDriverConnection
def wdc = new WebDriverConnection(rc)
def driver = wdc.getDriver()
println sprintf("Current timeout: %d ms",driver.getCallTimeout())
driver.setCallTimeout(30000) # 30 sec
driver.get("http://www.slowpage.com") # Langsame WebDriver Aktion
driver.resetCallTimeout()
```

Beispiel 54.41: WebDriver Timeout Kontrolle (Groovy Script)

54.11.1 Die `WebDriverConnection` Klasse

Es folgt eine alphabetische Aufstellung aller Methoden der `WebDriverConnection` Klasse. Die verwendete Syntax ist ein Gemisch aus Java und Python. Python unterstützt zwar selbst keine statische Typisierung, die Parameter werden jedoch an Java weitergereicht, so dass falsche Typen Exceptions auslösen können. Folgt einem Parameter ein '='-Zeichen und ein Wert, ist dies der Defaultwert des Parameters und eine Angabe beim Aufruf ist optional.

`Object getComponent(WebElement element, String windowname=None)`
Gibt die QF-Test Komponente für das übergebene WebDriver WebElement zurück. Diese Komponente kann dann mit den anderen QF-Test APIs genutzt werden.(z.B.: die Pseudo-DOM-API aus Abschnitt 54.10(1253))

Parameter

`element`	Das WebDriver WebElement.
`windowname`	Der windowname des Browsers in dem das WebElement angefragt wird.
Rückgabewert	The zugehörige QF-Test Komponente.

`WebDriver getDriver(String windowname=None)`
Gibt die WebDriver Instanz zurück welche genutzt wird um mit dem Browser im WebDriver mode zu interagieren. Erfordert, dass zuvor eine Webseite mittels eines Web-Engine starten(737) Knotens geöffnet wurde.

Parameter

`windowname`	Der windowname des Browsers in dem die WebDriver Instanz angefragt wird.
Rückgabewert	Die WebDriver Instanz.

`WebElement getElement(Object componentOrId)`
Gibt das WebDriver WebElement für die übergebene QF-Test Komponente oder QF-Test ID der Komponente zurück.

Parameter

`componentOrId`	Die QF-Test Komponente oder die QF-Test ID der Komponente.
Rückgabewert	Das WebDriver WebElement Objekt der QF-Test Komponente.

`WebDriver getUnmanagedDriver(String browserType=None, DesiredCapabilities desiredCapabilities=None)`
Gibt eine WebDriver Instanz mit dem angegebenen Browser-Typ zurück. So lange noch keine Webseite mittels eines Web-Engine starten(737) Knotens geöffnet wurde wird die WebDriver-Instanz nicht von QF-Test überwacht, die damit geöffneten Webseiten und Komponenten werden also nicht direkt von QF-Test erkannt. Eine Interaktion kann dann nur mit der eingebetteten Selenium-API erfolgen, und Checks müssen via `rc.check` und `rc.checkEqual` erfolgen (siehe Abschnitt 11.1(187))

Parameter

`browserType`	Der Typ des Browsers, mit dem getestet werden soll (siehe Art des Browsers(738)). Ist der Typ leer oder nicht gesetzt wird zunächst die `browserName` Capability der *desiredCapabilities* ausgewertet, und zuletzt wird der Wert des Art des Browsers(738) Attributs im Web-Engine starten(737) Knoten, mit dem das SUT gestartet wurde.
`desiredCapabilities`	Die DesiredCapabilities, welche an den WebDriver weitergereicht werden sollen.
Rückgabewert	Die WebDriver Instanz.

54.12 Windows Control API

5.0+

Die Elemente von nativen Windows-Anwendungen werden im Test durch Java-Objekte der Klasse `WinControl` repräsentiert. Diese stellt zum Beispiel für die Entwicklung eigener Resolver verschiedene Methoden öffentlich zur Verfügung.

54.12.1 Die `WinControl` Klasse

Es folgt eine alphabetische Aufstellung aller Methoden der `WinControl` Klasse.

`WinControl getAncestorByUiaType(String typeName)`

Sucht ein Vorgänger-Control mit dem angegeben UIAutomation-Typ.

Parameter

`typeName`	Der Typ als Suchkriterium.
Rückgabewert	Der Vorgänger oder `null`.

`WinControl getChild(int index)`

Liefert das Kind des Controls mit dem angegebenen Index.

Parameter

`index`	Der Index des Kind-Controls.
Rückgabewert	Das Kind.

Exceptions

`IllegalArgumentException`	Wenn der Index negativ ist oder größer/gleich der Anzahl der Kind-Controls.

`int getChildCount()`

Liefert die Anzahl an Kind-Controls.

Rückgabewert	Die Anzahl der Kind-Controls.

`WinControl[] getChildren()`

Liefert alle Kind-Controls des `WinControls`.

Rückgabewert	Ein Array mit allen Kind-Controls.

`WinControl[] getChildrenByUiaClassName(String className)`

Liefert alle Kinder des Controls mit dem angegebenen UIAutomation-Klassennamen.

Parameter

`className`	Der Klassenname.
Rückgabewert	Ein Array mit den Kind-Controls.

`WinControl[] getChildrenByUiaType(String typeName)`

Liefert alle Kinder des Controls mit dem angegebenen UIAutomation-Typs.

Parameter

`typeName`	Der UIAutomation-Typ.
Rückgabewert	Ein Array mit den Kind-Controls.

`WinControl[] getElementsByClassName(String className)`

Liefert alle untergeordneten Controls zurück, die den angegeben Klassennamen haben.

Parameter

`className`	Der Klassenname als Suchkriterium.
Rückgabewert	Ein Array von `WinControl` Objekten, welche das aktuelle Control als Vorgänger und den angegebenen Klassennamen haben.

`WinControl[] getElementsByClassName(String[] classNames)`

Liefert alle untergeordneten Controls zurück, die einen der angegeben Klassennamen haben.

Parameter

`classNames`	Passende Klassennamen als Suchkriterium.
Rückgabewert	Ein Array von `WinControl` Objekten, welche das aktuelle Control als Vorgänger und einen der angegebenen Klassennamen haben.

`WinControl[] getElementsByClassName(String[] classNames, String[] stopClassNames)`

Liefert alle untergeordneten Controls zurück, die einen der angegeben Klassennamen haben, aber bricht die Suche bei jedem Control ab, welches einen Klassennamen aus `stopClassNames` hat.

Parameter

`classNames`	Passende Klassennamen als Suchkriterium.
`stopClassNames`	Klassennamen als Stop-Kriterium für die Tiefensuche.
Rückgabewert	Ein Array mit passenden Nachfolger-Controls.

`String[] getGenericClassNames()`

Liefert die generischen Klassennamen des Controls.

Rückgabewert	Ein String-Array mit den Klassennamen des Controls.

`int getHwnd()`

Liefert das native Windows-Handle des Controls.

Rückgabewert	Das native Windows-Handle.

`int[] getLocation()`

Liefert die (physikalische) Pixel-Position des Controls in seinem Eltern-Control.

Rückgabewert	Ein Array mit den X- und Y-Koordinaten.

`int[] getLocationOnScreen()`

Liefert die (physikalische) Pixel-Position des Controls auf dem Bildschirm.

Rückgabewert	Ein Array mit den X- und Y-Koordinaten, der Breite und der Höhe.

WinControl getNextSibling()

Liefert den nächsten Nachbarn des Controls zurück.

Rückgabewert Das nächste Nachbar-Control oder `null`, wenn es keine weiteren Nachbarn mehr gibt.

String getPatterns()

Liefert alle Pattern des Controls (mit Leerzeichen getrennt). Die Methode `hasPattern()` ist zu bevorzugen, damit man nicht mit dem genauen Format des Strings arbeiten muss.

Rückgabewert Eine Zeichenkette mit den Pattern.

int[] getSize()

Liefert die (physikalische) Größe des Elements.

Rückgabewert Ein Feld mit der Breite und Höhe.

String getTextOrValue()

Liefert den "Wert" eines Elements, der sich meist aus dem Value- oder Text-Pattern ergibt, falls verfügbar. Der Wert kann sich aus dem Text der Kind-Controls oder in einigen Fällen aus dem Automation Name ergeben.

Rückgabewert Ein Wert oder `null`.

WinControl getTopAncestor()

Liefert den obersten Vorgänger des Controls.

Rückgabewert Der obersten Vorgänger oder `null`.

String getUiaClassName()

Liefert den Klassennamen des `WinControl`s. Dieser ergibt sich aus dem UIAutomation-Klassennamen, dem ein PRefix vorangestellt ist, um Verwechslungen mit den generischen QF-Test Klassennamen vorzubeugen.

Rückgabewert Eine Zeichenkette mit dem Klassennamen des Controls.

AutomationBase getUiaControl()

Erstellt für das `WinControl` ein `AutomationBase`, welches dann mit der `uiauto`-Skriptbibliothek verwendet werden kann (Kapitel 52(1136)).

Rückgabewert Das `AutomationBase`-Objekt.

String getUiaDescription()

Liefert die UIAutomation-Beschreibung des Controls. Wenn es keine FullDescription gibt, wird als Fallback die Beschreibung aus der Accessibility Schnittstelle zurückgeliefert.

Rückgabewert Die Beschreibung oder ein leerer String.

String getUiaHelp()

Liefert den Hilfetext des Controls.

Rückgabewert Der Hilfetext oder ein leerer String.

`String getUiaId()`

Liefert die UIAutomation-ID des Controls.

Rückgabewert Die ID, wenn gesetzt, sonst `null`.

`String getUiaName()`

Liefert den UIAutomation-Name des Controls.

Rückgabewert Der Name, falls gesetzt, sonst `null`.

`String getUiaType()`

Liefert den Typ des `WinControl`s. Dieser ergibt sich aus dem Namen des UIAutomation-Typs, ergänzt um ein Präfix `Uia.`, um Verwechslungen mit den generischen QF-Test Klassennamen vorzubeugen.

Rückgabewert Eine Zeichenkette mit dem Typ des Controls.

`boolean hasPattern(String pattern)`

Überprüft, ob das zugrundeliegende Automation Element das angegebene Pattern unterstützt.

Parameter

`pattern` Das Pattern, z.B. ”Invoke”, ”ExpandCollapse” usw.

Rückgabewert `true` wenn das Element das Pattern unterstützt, sonst `false`.

`boolean isMatchingClass(String className)`

Prüft, ob dem Cantrol die angegebene Klasse zugeordnet ist.

Parameter

`className` Der zu prüfende Klassenname.

Rückgabewert `true` wenn dem Control die angegebene Klasse zugeordnet ist, sonst `false`.

`boolean isMatchingClass(String[] classNames)`

Prüft, ob das Cantrol eine der angegebenen Klassen zugeordnet ist.

Parameter

`classNames` Die zu prüfende Klassennamen.

Rückgabewert `true` wenn dem Control eine der angegebenen Klassen zugeordnet ist, sonst `false`.

`boolean isShowing()`

Liefert die Sichtbarkeit des Elements.

Rückgabewert `true`, wenn das Control als ”Sichtbar auf dem Bildschirm” angesehen wird, sonst `false`.

Kapitel 55

Daemon-Modus

!!! Warnung !!!

Jeder, der Zugriff auf den QF-Test Daemon hat, kann auf dessen Rechner Programme mit den Rechten des Benutzerkontos starten, unter dem der Daemon läuft. Daher sollte Zugriff nur berechtigten Nutzern gewährt werden.

Wenn Sie den Daemon nicht in einer sicheren Umgebung betreiben, in der jeder Nutzer als berechtigt gilt, oder wenn Sie eine eigene Bibliothek zum Zugriff auf den Daemon entwickeln, sollten Sie unbedingt **Abschnitt 55.3**(1294) **lesen**.

55.1 Daemon Konzepte

Im Daemon-Modus gestartet (vergleichbar aber nicht äquivalent zu einem ”Service” unter Windows) lauscht QF-Test auf RMI Verbindungen und stellt darüber ein Interface für die verteilte Ausführung von Tests zur Verfügung. Dies kann für die Testdurchführung in einem verteilten Lasttest Szenario ebenso hilfreich sein wie für die Integration mit vorhandenen Test-Management oder Test-Durchführungs Werkzeugen.

Um den Daemon-Modus zu aktivieren gibt es zwei spezielle Kommandozeilenargumente:

- `-daemon`(980) - Startet QF-Test im Daemon-Modus
- `-daemonport <Port>`(980) - Legt den Netzwerkport für den Daemon fest, Standard ist 3543.
- `-daemonrmiport <Port>`(980) - Legt den Netzwerkport für den RMI-Port des Daemon’s fest. Die Angabe dieses Parameters macht nur Sinn sollte der Daemon hinter einer Firewall betrieben werden. Standard ist ein beliebiger freier Port.

Im kombinierten Batch- und Daemon-Modus belegt QF-Test zunächst keine Lizenz sondern erst, wie unten beschrieben, dynamisch bei der Durchführung von Tests. Im interaktiven Daemon Modus ist QF-Test normal funktionsfähig und benötigt daher eine eigene Lizenz. Wie im Batchmodus nimmt es dabei zusätzlich Verbindungen von außen an und belegt dabei weitere Lizenzen. Dieses Szenario ist insbesondere während der Entwicklung von verteilten Tests hilfreich.

In einem laufenden Daemon können Tests auf folgende zwei Arten gestartet werden:

- Durch Verwendung des Kommandozeilenarguments `-calldaemon`[979] von QF-Test.
- Durch direkte Implementierung gegen die Daemon API, welche in Abschnitt 55.2[1277] beschrieben ist.

Beide Optionen werden - auch an Hand von Beispielen - in Abschnitt 25.2[346] näher erläutert.

55.2 Daemon API

Um QF-Test im Daemon-Modus von einer weiteren QF-Test Instanz oder einem anderen Werkzeug zur Durchführung von QF-Test basierten Tests einzusetzen müssen folgende relativ einfache Schritte implementiert werden:

Hinweis Wenn Sie eine auf dem Daemon API basierende Anwendung schreiben, müssen Sie sich über Sicherheitsthemen Gedanken machen und Sicherheit entweder deaktivieren, oder einige RMI-spezifische Properties setzen. Näheres hierzu finden Sie in Abschnitt 55.3[1294].

- Nutzen Sie den `DaemonLocator`, um damit Zugriff auf einen `Daemon` zu erhalten.
- Holen Sie sich entweder den gemeinsamen `TestRunDaemon` oder weisen Sie den `Daemon` an, einen `TestRunDaemon` zu erzeugen. Über diesen `TestRunDaemon` können Sie globale Variablen und das Wurzelverzeichnis der Testsuiten für die kommenden Testläufe definieren.
- Holen Sie sich entweder den gemeinsamen `DaemonRunContext` des `TestRunDaemon` oder lassen Sie diesen eine oder mehrere Instanzen von `DaemonRunContext` erzeugen. Ein `DaemonRunContext` repräsentiert den Ausführungs-Kontext für einen funktionalen Test oder Lasttest. Jeder `DaemonRunContext` ist eigenständig, mit Ausnahme von Gruppen die mittels `TestRunDaemon.createContexts(int threads)` erstellt wurden. Diese bilden einen gemeinsamen Kontext für Lasttests, analog zum Start von QF-Test

mit dem Kommandozeilenargument -threads <Anzahl>[994]. Jeder DaemonRunContext belegt eine QF-Test Entwickler- oder Runtime Lizenz für die Dauer seines Bestehens.

- Nun können Sie den DaemonRunContext anweisen, Tests in Ihrem Namen durchzuführen, wahlweise ganze Testsuiten oder spezifische Testfallsatz[606] oder Testfall[599] Knoten. Mit richtig implementierten Abhängigkeiten[630] können Sie im Daemon-Modus einzelne Testfälle in beliebiger Reihenfolge ausführen, ohne sich um die nötigen Vorbereitungs- und Aufräumarbeiten kümmern zu müssen.
- Der DaemonRunContext bietet auch Methoden zur Abfrage des aktuellen Status eines Tests oder zum Warten auf sein Ende an.
- Schließlich können Sie noch das Protokoll des Testlauf vom Daemon abholen. Im Moment können Sie dieses lediglich in eine Datei speichern. Später planen wir die API weiter zu öffnen, um zum Beispiel Protokolle von verschiedenen Testläufen aus potenziell verschiedenen Daemons in einem Protokoll zusammenzufassen.

Beim Übergang von einem Testfall zum nächsten kann die korrekte Abarbeitung der Abhängigkeiten - inklusive das Auflösen der nicht mehr benötigten Abhängigkeiten auf dem Stapel - nur dann funktionieren, wenn immer der selbe DaemonRunContext verwendet wird. Dies erreichen Sie am einfachsten durch Verwendung der gemeinsamen TestRunDaemon und DaemonRunContext Objekte. Das Erzeugen von eigenen Objekten macht nur in Spezialfällen Sinn.

Die folgenden Abschnitte enthalten eine komplette Referenz der gesamten Daemon API. Erläuterungen und Beispiele zu deren Verwendung finden Sie in Abschnitt 25.2[346].

55.2.1 Der DaemonLocator

Über die Singleton Klasse de.qfs.apps.qftest.daemon.DaemonLocator kann auf Daemon Instanzen zugegriffen werden.

---

static DaemonLocator instance()

Es gibt immer nur ein einziges DaemonLocator Objekt und diese Methode ist der einzige Weg, Zugriff auf diese Singleton Instanz zu erlangen.

Rückgabewert Die DaemonLocator Singleton Instanz.

Daemon locateDaemon(String host, int port)

Liefert einen Daemon für einen spezifischen Host und Port.

Parameter

host	Der Ziel-Host, Name oder IP-Adresse als String.
port	Der Ziel-Port.
Rückgabewert	Der Daemon oder null falls kein Daemon gefunden wurde.

`Daemon[] locateDaemons(long timeout)`

Alle bekannten Daemons lokalisieren.

Parameter

`timeout`	Die Zeit in Millisekunden, um auf Reaktionen von Daemons zu warten.
Rückgabewert	Die bekannten Daemons.

`void setKeystore(String keystoreFile)`

Legt den Keystore fest, welcher für die Sicherung der Daemon-Kommunikation verwendet werden soll.

Parameter

`keystoreFile`	Der Pfad zur Datei, welche den Keystore zur Verschlüsselung der Daemon-Kommunikation enthält.

`void setKeystorePassword(String password)`

Legt das Passwort für den Keystore fest, der mit `setKeystore` gesetzt wurde.

Parameter

`password`	Das Passwort.

`void setTruststore(String truststoreFile)`

Legt den Truststore fest, welcher für die Sicherung der Daemon-Kommunikation verwendet werden soll. Wenn dieser nicht gesetzt ist dann wird der Keystore verwendet, der mit `setKeystore` gesetzt wurde.

Parameter

`truststoreFile`	Der Pfad zur Datei, welche den Truststore zur Verschlüsselung der Daemon-Kommunikation enthält.

`void setTruststorePassword(String password)`

Legt das Passwort für den Truststore fest, der mit `setTruststore` gesetzt wurde.

Parameter

`password`	Das Passwort.

55.2.2 Der `Daemon`

Das `de.qfs.apps.qftest.daemon.Daemon` Interface bietet einen Rahmen für verschiedene Arten von QF-Test Daemons. Aktuell ist nur der `TestRunDaemon` verfügbar, weitere Klassen wie ein `SUTClientStarterDaemon` sind bereits in Planung und weitere werden eventuell folgen.

`void cleanup()`

Räumt alle TestRunDaemons dieses Daemons und beendet dann alle Clients. Auf ein eventuelles Auflösen von Abhängigkeiten wird maximal 30 Sekunden gewartet.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`void cleanup(long timeout)`

Räumt alle TestRunDaemons dieses Daemons und beendet dann alle Clients.

Parameter

`timeout`	Die maximale Wartezeit in Millisekunden für ein eventuelles Aufräumen von Abhängigkeiten.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`TestRunDaemon createTestRunDaemon()`

Erzeugt einen TestRunDaemon.

Rückgabewert	Ein TestRunDaemon.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`String getHost()`

Liefert den Host des Daemons.

Rückgabewert	Der Host des Daemons.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`String getIp()`

Liefert die IP-Adresse des Daemons.

Rückgabewert	Die IP-Adresse des Daemons.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`int getPort()`

Liefert den Port des Daemons.

Rückgabewert	Der Port des Daemons.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`TestRunDaemon getSharedTestRunDaemon()`

Liefert den gemeinsamen TestRunDaemon.

Rückgabewert	Der gemeinsame TestRunDaemon.
Exceptions	
`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`TestRunDaemon[] getTestRunDaemons()`

Liefert alle von diesem Daemon erzeugten TestRunDaemons, die noch aktiv sind.

Rückgabewert	Die aktiven TestRunDaemons, die von diesem Daemon erzeugt wurden, exklusive dem gemeinsamen TestRunDaemon.
Exceptions	
`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`void killClients()`

Beendet alle Clients, die zur VM des Daemons gehören.

Exceptions	
`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`void ping()`

Prüft, ob der Daemon noch lebt.

Exceptions	
`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`void terminate(int exitCode)`

Beendet den Daemon-Prozess durch den Aufruf von System.exit.

Parameter	
`exitCode`	Der Exit-Code für den Daemon.
Exceptions	
`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

55.2.3 Der `TestRunDaemon`

Der `de.qfs.apps.qftest.daemon.TestRunDaemon` ist der Einstiegspunkt für die Ausführung von Tests. Er stellt die Umgebung für Testläufe bereit und erstellt `DaemonRunContext` Objekte welche die eigentliche Durchführung übernehmen.

Verschiedenes

`Daemon getDaemon()`

Liefert den Daemon zu dem der TestRunDaemon gehört.

Rückgabewert	Der Daemon des TestRunDaemons.
Exceptions	
`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

Bearbeiten globaler Variablen

Der TestRunDaemon hält einen eigenen Satz globaler Variablen vor, die zur Initialisierung der globalen Variablen beim Erzeugen eines neuen DaemonRunContext dienen. Die folgenden Methoden haben keinen Einfluss auf bereits laufende DaemonRunContext Instanzen.

`void clearGlobals()`

Löscht alle globalen Variablen.

Exceptions	
`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`String getGlobal(String name)`

Liefert den Wert einer globalen Variable.

Parameter	
`name`	Der Name der globalen Variable.
Rückgabewert	Der Wert der globalen Variable oder null wenn undefiniert.
Exceptions	
`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`Object getGlobalObject(String name)`

Liefert den Wert einer globalen Variable.

Bei der Verarbeitung des gelieferten Objekts ist zu beachten, dass die Eigenschaften und Methoden des Objekts auch davon abhängig sind, mit welchem Skriptinterpreter das Objekt erstellt wurden.

Parameter	
`name`	Der Name der globalen Variable.
Rückgabewert	Der Wert der globalen Variable oder null wenn undefiniert.
Exceptions	
`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`Map getGlobalObjects()`
Liefert die Werte aller globalen Variablen.
Bei der Verarbeitung der gelieferten Objekte ist zu beachten, dass die Eigenschaften und Methoden der Objekte auch davon abhängig sind, mit welchem Skriptinterpreter die Objekte erstellt wurden.

Rückgabewert Die Werte aller globalen Variablen.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

`Properties getGlobals()`
Liefert die Werte aller globalen Variablen als Strings.

Rückgabewert Die Werte aller globalen Variablen als Strings.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

`void setGlobal(String name, String value)`
Setzt den Wert einer globalen Variable.

Parameter

`name` Der Name der globalen Variable.

`value` Der Wert der globalen Variable.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

Testausführung

`void cleanup()`
Räumt alle DaemonRunContexte dieses TestRunDaemons auf und gibts sie frei. Auf ein eventuelles Auflösen von Abhängigkeiten wird maximal 30 Sekunden gewartet.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

`void cleanup(long timeout)`
Räumt alle DaemonRunContexte dieses TestRunDaemons auf und gibts sie frei.

Parameter

`timeout` Die maximale Wartezeit in Millisekunden für ein eventuelles Aufräumen von Abhängigkeiten.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

`DaemonRunContext createContext()`
Erzeugt einen einzelnen Daemon Runcontext. Hierfür wird eine Lizenz benötigt und belegt.

Rückgabewert Der Runcontext oder null wenn keine Lizenz belegt werden konnte.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

`DaemonRunContext[] createContexts(int threads)`
Erzeugt Daemon-Runcontext-Objekte für mehrere Threads. Es wird eine Lizenz je Thread benötigt und belegt.

Parameter

`threads` Die Zahl der Threads für den Runcontext.

Rückgabewert Die Runcontext-Objekte oder null wenn nicht genug Lizenzen belegt werden konnten.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

`DaemonRunContext[] getContexts()`
Liefert alle von diesem TestRunDaemon erzeugten DaemonRunContexte, die noch aktiv sind und nicht freigegeben wurden.

Rückgabewert Die aktiven DaemonRunContexte, die von diesem TestRunDaemon erzeugt wurden, exklusive dem gemeinsamen DaemonRunContext.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

`DaemonRunContext getSharedContext()`
Liefert den gemeinsamen DaemonRunContext. Falls dieser neu erstellt werden muss, wird eine neue Lizenz benötigt und belegt.

Rückgabewert Der gemeinsame RunContext oder null wenn keine Lizenz belegt werden konnte.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

`void setRootDirectory(String directory)`

Setzt das Testsuite-Wurzelverzeichnis für neu erzeugte Daemon Runcontexte.

Parameter

`directory` Das neue Wurzelverzeichnis.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

Identifikation

`String getIdentifier()`
Liefert den Namen zur Identifikation des TestRunDaemons. Wurde kein Name explizit via setIdentifier gesetzt, wird ein Name aus dem Namen des Daemons, zu dem der TestRunDaemon gehört, sowie einem Zähler gebildet.

Rückgabewert Der Name des TestRunDaemons.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

`void setIdentifier(String identifier)`
Setzt einen Namen zur Identifikation des TestRunDaemons. Dies kann nützlich sein um einen TestRunDaemon zu identifizieren, der via Daemon.getTestRunDaemons() ermittelt wurde.

Parameter

`identifier` Der zu setzende Name.

Exceptions

`RemoteException` Wenn etwas RMI-Spezifisches schief geht.

55.2.4 Der `DaemonRunContext`

Das `de.qfs.apps.qftest.daemon.DaemonRunContext` Interface dient der eigentlichen Testdurchführung.

Folgende Laufzustände sind definiert:

Status	Wert	Beschreibung
`STATE_INVALID`	`-1`	Ungültig nach der Freigabe - kann nicht reaktiviert werden.
`STATE_IDLE`	`0`	Kein Lauf geplant.
`STATE_SCHEDULED`	`1`	Lauf geplant aber nicht gestartet.
`STATE_RUNNING`	`2`	Laufend.
`STATE_PAUSED`	`3`	Laufend aber pausiert.
`STATE_FINISHED`	`4`	Lauf beendet, Resultat und Protokoll verfügbar.

Tabelle 55.1: Der Laufzustand

6.0+ Die folgenden Ergebnis-Werte für die Methode `getResult()` sind die gleichen wie überall in QF-Test:

Ergebnis	Wert	Beschreibung
RESULT_OK	0	Lauf OK, keine Warnungen, Fehler oder Exceptions.
RESULT_WARNING	1	Lauf weitgehend OK, einige Warnungen aber keine Fehler oder Exceptions.
RESULT_ERROR	2	Lauf mit Fehlern aber ohne Exceptions.
RESULT_EXCEPTION	3	Lauf mit Exception fehlgeschlagen.

Tabelle 55.2: Die Ergebnis-Werte

`void addTestRunListener(DaemonTestRunListener listener, boolean synchronous, long timeout)`

Registriert einen DaemonTestRunListener beim DaemonRunContext.

Parameter

`listener`	Der zu registrierende Listener.
`synchronous`	Legt fest, ob der Listener synchron benachrichtigt werden soll. In diesem Fall wird der Testlauf blockiert, bis der Listener den Event verarbeitet hat.
`timeout`	Wartezeit in Millisekunden für Aufrufe des Listeners. Falls der Listener nicht innerhalb dieser Zeitspanne antwortet, wird er automatisch deregistriert, um weitere Probleme zu vermeiden. Im Fall eines synchronen Listeners läuft dann auch der Test weiter. Ein Wert von 0 bedeutet kein Timeout, was nicht ungefährlich ist, aber nützlich sein kann.

---

boolean callProcedure(String procedure, Properties bindings=None)

Führt eine Prozedur in diesem Runcontext aus.

Parameter

procedure	Die Prozedur, die ausgeführt werden soll, in der Form `Suite#Procedure`, wobei `Procedure` der vollständige Name einer Prozedur sein muss.
bindings	Ein optionaler Satz von Variablendefinitionen. Diese binden stärker als die globalen Variablen und als alle Definitionen auf dem sekundären Stapel.
Rückgabewert	True falls der Prozeduraufruf gestartet wurde, false wenn die Suite oder die Prozedur nicht gefunden werden konnten.

Exceptions

RemoteException	Wenn etwas RMI-Spezifisches schief geht.
IllegalStateException	Falls im gegenwärtigen Status kein Test gestartet werden kann, der Status also weder STATE_IDLE noch STATE_FINISHED ist.

void clearGlobals()

Löscht die globalen Variablen des DaemonRunContext.

Exceptions

RemoteException	If something RMI specific goes wrong.

void clearTestRunListeners()

Entfernt alle DaemonTestRunListener vom DaemonRunContext.

String getGlobal(String name)

Liefert den Wert einer globalen Variable als String im DaemonRunContext.

Parameter

name	Der Name der Variable.
Rückgabewert	Der Wert der Variable als String oder null falls nicht definiert.

Exceptions

RemoteException	Wenn etwas RMI-Spezifisches schief geht.

`Object getGlobalObject(String name)`
Liefert den Wert einer globalen Variable im DaemonRunContext.
Bei der Verarbeitung des gelieferten Objekts ist zu beachten, dass die Eigenschaften und Methoden des Objekts auch davon abhängig sind, mit welchem Skriptinterpreter das Objekt erstellt wurden.

Parameter

`name`	Der Name der Variable.
Rückgabewert	Der Wert der Variable oder null falls nicht definiert.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`Map getGlobalObjects()`
Liefert alle globalen Variablen im DaemonRunContext.
Bei der Verarbeitung der gelieferten Objekte ist zu beachten, dass die Eigenschaften und Methoden der Objekte auch davon abhängig sind, mit welchem Skriptinterpreter die Objekte erstellt wurden.

Rückgabewert	Die globalen Variablen.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`Properties getGlobals()`
Liefert alle globalen Variablen als Strings im DaemonRunContext.

Rückgabewert	Die globalen Variablen als Strings.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`Object getGroupObject(String group, String name)`
Liefert den Wert eines Gruppen-Objekts (Property oder Ressource) im DaemonRun-Context.
Bei der Verarbeitung des gelieferten Objekts ist zu beachten, dass die Eigenschaften und Methoden des Objekts auch davon abhängig sind, mit welchem Skriptinterpreter das Objekt erstellt wurden.

Parameter

`name`	Der Name der Property- oder Ressource-Gruppe.
`name`	Der Name der Property.
Rückgabewert	Der Wert der Property oder null falls nicht definiert.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`Map getGroupObjects(String group)`
Liefert alle Objekte einer Property- oder Ressource-Gruppe im DaemonRunContext. Bei der Verarbeitung der gelieferten Objekte ist zu beachten, dass die Eigenschaften und Methoden der Objekte auch davon abhängig sind, mit welchem Skriptinterpreter die Objekte erstellt wurden.

Parameter

`name`	Der Name der Property- oder Ressource-Gruppe.
Rückgabewert	Die Properties oder null falls die Gruppe nicht existiert.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`String getIdentifier()`
Liefert den Namen zur Identifikation des DaemonRunContexts. Wurde kein Name explizit via setIdentifier gesetzt, wird ein Name aus dem Namen des TestRunDaemons, zu dem der DaemonRunContext gehört, sowie einem Zähler gebildet.

Rückgabewert	Der Name des DaemonRunContexts.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`String getLastTest()`
Liefert den Namen der aktuell oder zuletzt von diesem DaemonRunContext ausgeführt wird oder wurde.

Rückgabewert	Der Name des aktuellen bzw. zuletzt ausgeführten Tests.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`int getNumThreads()`

Liefert die Zahl der Threads in der Gruppe zu welcher der DaemonRunContext gehört.

Rückgabewert	Die Zahl der Threads der Gruppe des DaemonRunContext.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`Properties getProperties(String group)`
Liefert alle Properties einer Property- oder Ressource-Gruppe als Strings im DaemonRunContext.

Parameter

`name`	Der Name der Property- oder Ressource-Gruppe.
Rückgabewert	Die Properties als Strings oder null falls die Gruppe nicht existiert.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`String getProperty(String group, String name)`

Liefert den Wert einer Property oder Ressource als String im DaemonRunContext.

Parameter

`name`	Der Name der Property- oder Ressource-Gruppe.
`name`	Der Name der Property.
Rückgabewert	Der Wert der Property als String oder null falls nicht definiert.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`int getResult()`

Liefert das Resultat des Testlaufs.

Rückgabewert	Das Ergebnis des Testlaufs, einer der Werte RESULT_OK, RESULT_WARNING, RESULT_ERROR oder RESULT_EXCEPTION.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.
`IllegalStateException`	Wenn der Laufzustand ungleich STATE_FINISHED ist.

`byte[] getRunLog()`

Liefert das Protokoll des Testlaufs.

Rückgabewert	Das Protokoll in Form eines byte array.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.
`IllegalStateException`	Wenn der Laufzustand ungleich STATE_FINISHED ist.

`int getRunState()`

Liefert den aktuellen Laufzustand des Runcontext.

Rückgabewert	Der aktuelle Laufzustand des Runcontext, einer von STATE_IDLE, STATE_SCHEDULED, STATE_RUNNING, STATE_PAUSED oder STATE_FINISHED.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`TestRunDaemon getTestRunDaemon()`

Liefert den TestRunDaemon zu dem der DaemonRunContext gehört.

Rückgabewert	Der TestRunDaemon des DaemonRunContext.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`int getThreadNum()`

Liefert den Thread-Index des DaemonRunContext.

Parameter

Rückgabewert	Der Thread-Index des DaemonRunContext.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`void release()`

Gibt den DaemonRunContext frei und auch die Lizenz, die dieser belegt hatte. Wenn ein Test läuft, wird dieser angehalten.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.
`IllegalStateException`	Wenn kein DaemonRunContext angelegt war.

`void removeTestRunListener(DaemonTestRunListener listener)`

Entfernt einen DaemonTestRunListener vom DaemonRunContext.

Parameter

`listener`	Der zu entfernende Listener.

`void rollbackDependencies()`

Löst die Abhängigkeiten dieses DaemonRunContext auf.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.
`IllegalStateException`	Falls im gegenwärtigen Status kein Test gestartet werden kann, der Status also weder STATE_IDLE noch STATE_FINISHED ist.

boolean runTest(String test, Properties bindings=None)

Startet einen Test in diesem Runcontext.

Parameter

test Der Test, der ausgeführt werden soll, in der Form `Suite#Test`, wobei `#Test` optional ist und `Test` der vollständige Name eines Testfallsatzes oder Testfalls oder ein "." sein kann. Letzteres ist äquivalent zur Angabe von `Suite` alleine und bewirkt das Ausführen der ganzen Testsuite.

Beispiele:

`MySuite`	Führt die ganze Testsuite `MySuite` aus.
`MySuite#.`	Führt die ganze Testsuite `MySuite` aus.
`MySuite#MyTestSet`	Führt den Testfallsatz `MyTestSet` in der Testsuite `MySuite` aus.
`MySuite#MyTestSet.MyTestCase`	Führt den Testfall `MyTestCase` aus, der sich im Testfallsatz `MyTestSet` in der Testsuite `MySuite` befindet.

bindings Ein optionaler Satz von Variablendefinitionen. Diese binden stärker als die globalen Variablen und als alle Definitionen auf dem sekundären Stapel.

Rückgabewert True falls der Test gestartet wurde, false wenn die Suite oder der Test nicht gefunden werden konnten.

Exceptions

RemoteException Wenn etwas RMI-Spezifisches schief geht.

IllegalStateException Falls im gegenwärtigen Status kein Test gestartet werden kann, der Status also weder STATE_IDLE noch STATE_FINISHED ist.

`void setIdentifier(String identifier)`
Setzt einen Namen zur Identifikation des DaemonRunContexts. Dies kann nützlich sein, um einen DaemonRunContext zu identifizieren, der via TestRunDaemon.getContexts() ermittelt wurde.

Parameter

`identifier`	Der zu setzende Name.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`void setGlobals(Properties globals)`
Setzt die globalen Variablen des DaemonRunContext.

Parameter

`globals`	Die zu setzende globalen Variablen.

Exceptions

`RemoteException`	If something RMI specific goes wrong.

`void setRootDirectory(String directory)`
Setzt das Testsuite-Wurzelverzeichnis für den nächsten Testlauf.

Parameter

`directory`	Das neue Wurzelverzeichnis.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

`void stopRun()`
Hält den Testlauf an.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.
`IllegalStateException`	Wenn kein Testlauf angestoßen war.

`boolean waitForRunState(int state, long timeout)`
Wartet bis der Runcontext einen vorgegebenen Zustand erreicht.

Parameter

`state`	Der Zustand auf den gewartet werden soll.
`timeout`	Maximale Wartezeit im Millisekunden.
Rückgabewert	True falls der Zustand erreicht wurde, false wenn die Wartezeit verstrichen ist und der Zustand bis dahin nicht erreicht wurde.

Exceptions

`RemoteException`	Wenn etwas RMI-Spezifisches schief geht.

55.2.5 Der `DaemonTestRunListener`

3.1+

Das Interface `de.qfs.apps.qftest.daemon.DaemonTestRunListener` ist identisch zum Interface `de.qfs.apps.qftest.extensions.qftest.TestRunListener`, welches in Abschnitt 54.6(1223) beschrieben wird, außer dass seine Methoden eine `RemoteException` bei RMI Problemen werfen können. Wenn Sie dieses Interface implementieren, müssen Sie von der Klasse `java.rmi.server.UnicastRemoteObject` ableiten.

Sie können den Listener beim `DaemonRunContext` mittels seiner Methode `addTestRunListener` registrieren, die im vorhergehenden Abschnitt beschrieben wurde.

55.3 Absicherung des QF-Test Daemon

3.5+

Jeder, der auf den QF-Test Daemon Zugriff hat, kann auf dessen Rechner und mit dessen Benutzerkonto Programme starten, daher sollte dieser Zugriff auf den dazu berechtigten Personenkreis beschränkt werden.

Selbstverständlich sollte der QF-Test Daemon grundsätzlich auf einem Rechner betrieben werden, der durch eine Firewall vor dem Zugriff von außerhalb des lokalen Netzwerks geschützt ist. Wenn alle verbleibenden Anwender, die diesen Rechner erreichen können, zum Zugriff auf den Daemon berechtigt sind, ist diese Maßnahme ausreichend. Wenn der Nutzerkreis weiter eingeschränkt werden soll, lesen Sie bitte weiter.

Standardmäßig verwendet der QF-Test Daemon SSL zur Absicherung seiner RMI Verbindung. Ohne zusätzliche Maßnahmen bedeutet dies jedoch nur, dass die Kommunikation zwischen dem Daemon und seinem Client verschlüsselt wird. Um den Zugriff auf bestimmte Anwender zu beschränken, ist ein weiterer Schritt nötig.

Das Aufsetzen einer SSL Kommunikation kann sehr komplex sein. Man muss sich normalerweise mit den Themen ”Schlüssel”, ”Zertifikate”, ”Zertifizierungsstelle”, ”ununterbrochene Sicherheitskette” etc. befassen. Zum Glück ist dies ein sehr spezieller Fall und die Tatsache, dass ein:e Anwender:in, der/die Zugriff auf den Daemon hat, damit auch Kontrolle über dessen Rechner erhält, macht eine Unterscheidung zwischen Daemon-Administrator:in und Daemon-Nutzer:in aus Sicherheits-Sicht überflüssig.

Ohne zu tief in die Details einzusteigen: QF-Test nutzt normalerweise einen einzelnen Keystore mit einem einzelnen selbst-signierten Zertifikat sowohl auf der Daemon als auch der Client-Seite. Komplexere Szenarien sind möglich aber nicht Gegenstand dieses Handbuchs. Die Standard-Keystore-Datei heißt `daemon.keystore` und liegt im Systemverzeichnis oder im versionsspezifischen Verzeichnis von QF-Test. Durch Erstellen einer eigenen Keystore-Datei wie unten beschrieben können Sie sicher stellen,

dass nur Anwender auf den Daemon zugreifen können, die diese Keystore-Datei verwenden.

55.3.1 Erstellen einer eigenen Keystore-Datei

Um eine Keystore-Datei zu erzeugen, benötigen Sie ein aktuelles JDK Version 1.5 oder höher - ein JRE ist nicht ausreichend. In einer Shell oder einem Konsolenfenster führen Sie folgendes Kommando aus (ggf. müssen Sie den kompletten Pfad für das `keytool` Programm angeben, welches sich im `bin` Verzeichnis des JDK befindet):

```
keytool -keystore daemon.keystore -genkey -alias "qftest daemon"
          -keyalg DSA -validity 999999
```

Beispiel 55.1: Erstellen einer Keystore-Datei zur Absicherung der Kommunikation mit dem Daemon

Nähere Informationen zu `keytool` finden Sie unter http://download.oracle.com/javase/6/docs/technotes/tools/solaris/keytool.htm.

Als Kennwort für die Keystore-Datei geben Sie `123456` ein. Wenn Sie nach Name oder Organisation gefragt werden, können Sie beliebige Einträge machen, für QF-Test sind diese nicht relevant. Sie können natürlich auch ein sicheres Kennwort an Stelle von `123456` angeben, aber das macht nur den Start des Daemon und seiner Clients komplizierter, ohne wirklich viel zur Sicherheit beizutragen. Sie könnten auch einen kürzeren Gültigkeitszeitraum angeben, aber falls die Keystore-Datei einmal in die falschen Hände gerät, müssen Sie lediglich den Daemon und seine Anwender mit einer neuen Keystore-Datei versorgen, die alte ist dann wertlos.

55.3.2 Festlegen der Keystore-Datei

Sie können QF-Test auf verschiedene Arten mitteilen, welche Keystore-Datei für den Daemon verwendet werden soll:

- Speichern Sie die Datei als `daemon.keystore` in QF-Test's Systemverzeichnis.
- Speichern Sie die Datei als `daemon.keystore` im benutzerspezifischen Konfigurationsverzeichnis[12].
- Speichern Sie die Datei wo Sie möchten und geben Sie diese über das Kommandozeilenargument `-keystore <Keystore-Datei>`[983] an QF-Test.

Falls Sie ihr eigenes Kennwort für die Keystore-Datei gewählt haben, müssen Sie dieses über das Kommandozeilenargument `-keypass <Kennwort>`(983) an QF-Test übergeben.

Falls Sie den Daemon ohne SSL Unterstützung starten möchten, z.B. um mit einer QF-Test Version älter als 3.5 zu interagieren, entfernen Sie entweder die Datei `daemon.keystore` aus QF-Test's versionsspezifischem Verzeichnis oder verwenden Sie das Kommandozeilenargument in der Form `-keystore=` um festzulegen, dass kein Keystore verwendet werden soll..

55.3.3 Festlegen der Keystore-Datei auf der Client-Seite

Wenn Sie mittels `qftest -batch -calldaemon` oder über Skript-Knoten von QF-Test aus auf den Daemon zugreifen, gelten die selben Anweisungen wir für den Daemon selbst.

Um den Daemon über SSL von Ihrem eigenen Code aus über die Daemon API anzusteuern, müssen Sie in Ihrer Anwendung die System Properties `javax.net.ssl.keyStore` und `javax.net.ssl.trustStore` auf die Keystore-Datei und die Property `javax.net.ssl.keyStorePassword` auf das Kennwort für die Keystore-Datei setzen. In Abschnitt 55.2(1277) finden Sie Details zur Daemon API und in Abschnitt 25.2(346) Beispiele dazu.

Kapitel 56

Die Procedure Builder Definitionsdatei

Für allgemeine Informationen über den Procedure Builder, siehe Kapitel Kapitel 27(368).

56.1 Platzhalter

Sie können bestimmte Platzhalter in der Definition der Prozeduren und Packages verwenden. In der folgenden Liste finden Sie deren Beschreibung:

Platzhalter	Beschreibung
<COMPID>	Die QF-Test ID der Komponente
<COMPNAME>	Der Name der Komponente
<COMPFEATURE>	Das Merkmal der Komponente
<COMPCLASS>	Die aufgezeichnete Klasse der Komponente
<COMPTKCLASS>	Die spezifische Klasse der Komponente
<COMPSYSCLASS>	Die Systemklasse der Komponente
<COMPGENCLASS>	Die generische Klasse der Komponente
<COMPEF-name-des-weiteren-merkmals>	Der Wert des angegebenen weiteren Merkmals der Komponente
<CURRENTVALUE>	Der aktuelle Wert der Komponente, z.B. der Text eines Textfeldes oder der aktuelle Wert einer Combobox etc.
<CURRENTENABLEDSTATE>	Der aktuelle Status, ob die Komponente enabled ist.
<CURRENTSELECTEDSTATE>	Der aktuelle Status, ob die Komponente selektiert ist.
<CURRENTEDITABLESTATE>	Der aktuelle Status, ob die Komponente editierbar ist.
<PCOMPID>	Die QF-Test ID der Elternkomponente
<PCOMPNAME>	Der Name der Elternkomponente
<PCOMPFEATURE>	Das Merkmal der Elternkomponente
<PCOMPCLASS>	Die Klasse der Elternkomponente
<PCOMPEF-name-des-weiteren-merkmals>	Der Wert des angegebenen weiteren Merkmals der Elternkomponente
<GPCOMPID>	Die QF-Test ID der Großelternkomponente
<GPCOMPNAME>	Der Name der Großelternkomponente
<GPCOMPFEATURE>	Das Merkmal der Großelternkomponente
<GPCOMPCLASS>	Die Klasse der Großelternkomponente
<GPCOMPEF-name-des-weiteren-merkmals>	Der Wert des angegebenen weiteren Merkmals der Großelternkomponente
<ENGINE>	Der Name der Engine, entweder 'awt', 'swt', 'web' oder 'fx'.
<ENGINE2>	Der alternative Name der Engine, entweder 'swing', 'swt', 'web' oder 'fx'.

Tabelle 56.1: Platzhalter für Komponentenprozeduren

Für Prozeduren einer Container oder Composite Komponente können Sie zusätzlich noch Platzhalter für deren Kindkomponenten definieren. Diese sind in der folgenden Liste erklärt:

Platzhalter	Beschreibung
<CCOMPID>	Die QF-Test ID der Kindkomponente
<CCOMPNAME>	Der Name der Kindkomponente
<CCOMPFEATURE>	Das Merkmal der Kindkomponente
<CCOMPCLASS>	Die aufgezeichnete Klasse der Kindkomponente
<CCOMPTKCLASS>	Die spezifische Klasse der Kindkomponente
<CCOMPSYSCLASS>	Die Systemklasse der Kindkomponente
<CCOMPGENCLASS>	Die generische Klasse der Kindkomponente
<CCOMPEF-name-des-weiteren-merkmals>	Der Wert des angegebenen weiteren Merkmals der Kindkomponente
<CCURRENTVALUE>	Der aktuelle Wert der Kindkomponente, z.B. der Text eines Textfeldes oder der aktuelle Wert einer Combobox etc.
<CCURRENTENABLEDSTATE>	Der aktuelle Status, ob die Kindkomponente enabled ist.
<CCURRENTSELECTEDSTATE>	Der aktuelle Status, ob die Kindkomponente selektiert ist.
<CCURRENTEDITABLESTATE>	Der aktuelle Status, ob die Kindkomponente editierbar ist.

Tabelle 56.2: Zusätzliche Platzhalter für Containerprozeduren

56.1.1 Rückgriffswerte für Platzhalter

4.1.3+

In vielen Projekten werden die Prozedur- und Parameternamen aus den Platzhaltern `COMPNAME`, `COMPFEATURE` oder `COMPEF-qfs:labelBest` generiert, weil diese den Komponentenbezeichner[66] oder die Beschriftung beinhalten. Nun kann es allerdings vorkommen, dass nicht alle Komponenten diese Platzhalter vollständig füllen können, z.B. könnte bei neu hinzugefügten Buttons der Name fehlen oder es kann für ein Textfeld kein passendes Label ermittelt werden. Nun kann man für diese Ausnahmen zwar Resolver implementieren, um auch diese Komponenten mit etwas brauchbaren zu füllen, aber so ein Schritt muss wieder stabile Informationen liefern. Stattdessen ist es nun auch möglich, Rückgriffswerte, so genannte Fallbackwerte, für diese Platzhalter zu spezifizieren. Diese Platzhalter werden herangezogen, wenn der eigentliche Wert nicht verfügbar ist, es wird also auf diese Werte zurückgegriffen.

Diese Werte definieren Sie im Attribut 'Bemerkung' der konfigurierten 'Prozedur'- oder 'Packages'-Knoten. Hierzu schreiben Sie `@fallback_` und dann den Namen des Platzhalters. Danach definieren Sie die Platzhalter, die als Rückgriffswerte dienen sollen. Für den Platzhalter `COMPNAME`, sieht eine solche Definition also wie folgt aus: `@fallback_COMPNAME COMPFEATURE`. Hier wird der Wert des Platzhalters `COMPFEATURE` verwendet, falls kein Wert für `COMPNAME` ermittelt werden konnte. Es ist

übrigens auch möglich, mehrere Fallbackwerte für einen Platzhalter anzugeben, z.B. `@fallback_COMPEF-qfs:labelBest COMPFEATURE,COMPNAME`. Wie Sie sehen, werden diese Platzhalter also kommasepariert angegeben.

56.2 Spezielle Bedingungen für die Definition von Packages und Prozeduren

Sie können die Erstellungen von Packages und Prozeduren über das Setzen des Attributs 'Bemerkung' des jeweiligen 'Package'- bzw. 'Prozedur'-Knotens beeinflußen.

Beschreibungsattribut	Beschreibung
@ABSOLUTECALL	Verwendung in Prozeduren, die mit @FORCHILDREN Prozeduraufrufe erzeugen. Wenn dieser Wert für den Prozeduraufruf gesetzt ist, wird nicht `procbuilder` als erstes Package im Attribut 'Name der Prozedur' des Prozeduraufrufs erzeugt, sondern direkt der Prozeduraufruf, der im Anschluss an die Kindkomponentenklasse spezifiziert ist.
@CONDITION	Hier kann man eine Bedingung für die Erstellung eines Packages, einer Prozedur oder eines Aufrufs angeben. Bedingungen können entweder in Jython, Groovy, JavaScript oder als regulärer Ausdruck (vgl. Abschnitt 49.3[1023]) definiert werden. Näheres hierzu finden Sie weiter unten.
@EXCEPT	Hier kann man Klassen angeben, wofür das Package bzw. die Prozedur nicht erstellt werden soll. Dieser Wert ist dann nützlich, wenn Sie Prozeduren für eine abstrakte Klasse definiert haben und nicht alle Ableitungen die entsprechenden Prozeduren benötigen.
@FORCECREATION	In Prozeduren, in denen @FORCHILDREN gesetzt ist, bewirkt diese Anweisung, dass der Prozeduraufruf unabhängig von Kindkomponenten einmalig angelegt wird.
@FORCHILDREN	Wenn dieser Wert für Container/Composite Prozeduren gesetzt ist, dann wird QF-Test in dieser Prozedur versuchen, die einzelnen Prozeduren der Kindkomponenten aufzurufen.
@NOTINHERIT	Wenn dieser Wert gesetzt ist, dann wird dieses Package bzw. diese Prozedur nur für Komponenten von exakt dieser Klasse verwendet und nicht für deren Ableitungen.
@SUBITEM	Funktioniert zur Zeit nur für Menüs. Wenn dieser Wert gesetzt ist wird das Package bzw. die Prozedur nur für Menüelemente der zweiten Hierarchieebene erstellt.
@SWTSTYLE	Prüft eine bestimmte Eigenschaft eines SWT-Buttons. Dieses SWT-spezifische Attribut ist notwendig, weil SWT zwischen Checkboxen, Comboboxen, Radiobuttons und normale Buttons nur mittels einer bestimmten Eigenschaft unterscheidet. Für einen normalen Button können Sie `@SWTSTYLE=PUSH`, für einen Radiobutton `@SWTSTYLE=RADIO` definieren etc..

Tabelle 56.3: Bemerkungsattribute für die Prozedurenerstellung

56.3 Auswertung der Komponentenhierarchie

In einigen Fällen kann es nützlich sein die Komponentenhierarchie in der Packagestruktur abzubilden. Dieser Ansatz erleichtert das Auffinden der jeweiligen Prozedur. Wenn Sie die Komponentenhierarchie in die Erstellung miteinbeziehen wollen, können Sie folgende Platzhalter in den Packgenamen verwenden:

Platzhalter	Beschreibung
<HIERARCHY>	Erstellt Packages für die gesamte Komponentehierarchie. Für die Ersetzung wird die QF-Test ID der Komponente herangezogen.
<HIERARCHY_NAME>	Erstellt Packages für die gesamte Komponentehierarchie. Für die Ersetzung wird der Komponentenname herangezogen. Besitzt die Komponente keinen Namen, wird diese ignoriert.
<HIERARCHY_FEATURE>	Erstellt Packages für die gesamte Komponentehierarchie. Für die Ersetzung wird das Komponentenmerkmal herangezogen. Besitzt die Komponente kein Merkmal, wird diese ignoriert.
<IHIERARCHY>	Erstellt Packages nur für interessante Komponenten in der Hierarchie. Eine interessante Komponente hat ein gesetztes Attribut 'Merkmal'. Für die Ersetzung wird die QF-Test ID der Komponente herangezogen.
<IHIERARCHY_NAME>	Erstellt Packages nur für interessante Komponenten in der Hierarchie. Eine interessante Komponente hat ein gesetztes Attribut 'Merkmal'. Für die Ersetzung wird der Komponentenname herangezogen. Besitzt die Komponente keinen Namen, wird diese ignoriert.
<IHIERARCHY_FEATURE>	Erstellt Packages nur für interessante Komponenten in der Hierarchie. Eine interessante Komponente hat ein gesetztes Attribut 'Merkmal'. Für die Ersetzung wird das Komponentenmerkmal herangezogen. Besitzt die Komponente kein Merkmal, wird diese ignoriert.
<MHIERARCHY>	Erstellt Packages nur für Menükomponenten in der Hierarchie. Menükomponenten sind Menüs und Menüelemente. Für die Ersetzung wird die QF-Test ID der Komponente herangezogen.
<MHIERARCHY_NAME>	Erstellt Packages nur für Menükomponenten in der Hierarchie. Menükomponenten sind Menüs und Menüelemente. Für die Ersetzung wird der Komponentenname herangezogen. Besitzt die Komponente keinen Namen, wird diese ignoriert.
<MHIERARCHY_FEATURE>	Erstellt Packages nur für Menükomponenten in der Hierarchie. Menükomponenten sind Menüs und Menüelemente. Für die Ersetzung wird das Komponentenmerkmal herangezogen. Besitzt die Komponente kein Merkmal, wird diese ignoriert.

Tabelle 56.4: Platzhalter für die Hierarchie

56.4 Details zu @CONDITION

3.4+

`@CONDITION` erlaubt es Ihnen, eine Bedingung für die Erstellung eines Knotens zu konfigurieren. Diese wird zum Erstellzeitpunkt ausgewertet.

Diese Bedingungen können verwendet werden, um auf einen bestimmten Namen zu prüfen oder auf das Vorhandensein eines bestimmten Buchstabens zu reagieren. Falls die Bedingung nicht erfüllt wird, wird der entsprechende Knoten nicht erzeugt. Sie können selbstverständlich alle bekannten Platzhalter wie <COMPID> oder <CCOMPNAME> verwenden.

Bedingung	Bedeutung
@CONDITION jython ”<COMPFEATURE>”.startswith(”abc”)	Hier wird eine Jythonbedingung definiert, die den Knoten nur dann erzeugt, wenn das Merkmal des Knotens mit ’abc’ beginnt. Es ist hier möglich sämtliche Vergleichsmethoden von Jython zu verwenden.
@CONDITION groovy ”<COMPFEATURE>”.startsWith(”abc”)	Hier wird eine Groovybedingung definiert, die den Knoten nur dann erzeugt, wenn das Merkmal des Knotens mit ’abc’ beginnt. Es ist hier möglich sämtliche Vergleichsmethoden von Groovy zu verwenden.
@CONDITION javascript ”<COMPFEATURE>”.startsWith(”abc”)	Hier wird eine JavaScriptbedingung definiert, die den Knoten nur dann erzeugt, wenn das Merkmal des Knotens mit ’abc’ beginnt. Es ist hier möglich sämtliche Vergleichsmethoden von JavaScript zu verwenden.
@CONDITION regexp ”<COMPFEATURE>” =∼ ”abc.*”	Hier wird eine Bedingung mittels regulären Ausdrucks (vgl. Abschnitt 49.3[1023]) definiert, die den Knoten nur dann erzeugt, wenn das Merkmal des Knotens den Ausdruck ’abc.*’ entspricht. Es ist hier möglich sämtliche Möglichkeiten von regulären Ausdrücken von Java zu verwenden.
@CONDITION regexp ”<COMPFEATURE>” !∼ ”abc.*”	Hier wird eine Bedingung mittels regulären Ausdrucks (vgl. Abschnitt 49.3[1023]) definiert, die den Knoten nur dann erzeugt, wenn das Merkmal des Knotens nicht den Ausdruck ’abc.*’ entspricht. Es ist hier möglich sämtliche Möglichkeiten von regulären Ausdrücken von Java zu verwenden.

Tabelle 56.5: Beispiele für @CONDITION

Falls Sie mehrzeilige Bedingungen definieren wollen, können Sie dies mit einem ’\’ am Ende der ersten Zeile tun.

Kapitel 57

Der ManualStepDialog

Der `ManualStepDialog` ist eine Java-Klasse, die mit QF-Test ausgeliefert wird. Sie können diesen Dialog auch für Ihren eigenen Bedarf verwenden. Hierzu ein Beispiel:

```
from de.qfs.apps.qftest import ManualStepDialog
#create the dialog and show it immediately
manualDialog = ManualStepDialog(None, "New Test Case Title", \
"Step Description", "Expected Test Result")
#did the test fail or succeed?
failOrSuccess = manualDialog.getResult()
#get the content of the received result
receivedResult = manualDialog.getReceivedResult()
#get the execution information, whether skipped or cancelled
execInfo = manualDialog.getExecInfo()
```

Beispiel 57.1

57.1 Die ManualStepDialog API

`ManualStepDialog ManualStepDialog(Component parent, String title, String stepText, String expResult)`

Konstruktor Methode der `ManualStepDialog` Klasse

Parameter

`parent`	Die Parent-Komponente des Dialoges
`title`	Der Titel des Dialoges
`stepText`	Der Text für das ”Beschreibung” Textfeld
`expResult`	Der Text für das ”Erwartete Ergebnis” Textfeld

`String getExecInfo()`

Gibt Informationen über die Ausführung zurück.

Rückgabewert Die Testausführungsinformationen

`String getReceivedResult()`

Gibt das erhaltene Resultat zurück.

Rückgabewert Das erhaltene Resultat

`String getResult()`

Gibt das Resultat des Testschrittes zurück. Für eine Liste der möglichen Resultate siehe Abschnitt 34.5(454).

Rückgabewert Das Resultat des Testschrittes

`boolean isStatusCanceled()`

Gibt zurück, ob der Status CANCELED ist.

Rückgabewert Wahr, wenn der Status CANCELED ist, sonst false

`boolean isStatusFailed()`

Gibt zurück, ob der Status FAILED ist.

Rückgabewert Wahr, wenn der Status FAILED ist, sonst false

`boolean isStatusPassed()`

Gibt zurück, ob der Status PASSED ist.

Rückgabewert Wahr, wenn der Status PASSED ist, sonst false

`boolean isStatusSkipped()`

Gibt zurück, ob der Status SKIPPED ist.

Rückgabewert Wahr, wenn der Status SKIPPED ist, sonst false

`void setExecInfo(String newExecInfo)`

Setzt Ausführungsinformationen des Testschrittes.

Parameter

`newExecInfo` Die Ausführungsinformationen des Testschrittes

`void setReceivedResult(String newRecResult)`

Setzt das erhaltene Resultat des Testschrittes.

Parameter

`newReceived Result` Das erhaltene Resultat

`void setResult(String newResult)`

Setzt das Resultat des Testschrittes. Für eine Liste der möglichen Resultate siehe Abschnitt 34.5(454).

Parameter

`newResult` Das Resultat

Kapitel 58

Details zu Knotenkonvertierungen

3.1+

58.1 Einführung

Der Konvertierungsmechanismus erlaubt es Knotentypen zu ändern, z.B. kann eine Sequenz in eine Prozedur oder ein Test in einen Testfall umgewandelt werden. Solche Aktionen können für das Refactoring sehr effizient sein.

Sie können eine Konvertierung mittels Rechtsklick auf den jeweiligen Knoten und Auswahl von Knoten konvertieren und den gewünschten Typen durchführen.

Hinweis QF-Test zeigt nur erlaubte Konvertierungen an, die im aktuellen Kontext auch möglich sind, deshalb können Sie manchmal nicht alle Möglichkeiten sehen.

58.2 Konvertierungen mit Typwechseln

Die folgenden Konvertierungen ändern zusätzlich noch den Typen einiger Kindknoten des konvertierten Knoten:

1. Testfallsatz in Testfall
   (a) Datentreiber in ausgeschaltete Sequenz
   (b) Testfall in Testschritt
2. Testfall in Testfallsatz
   (a) Alle Kindknoten werden in einen neuen Testfall gepackt.
3. Test in Testfallsatz rekursiv

   (a) Wenn es nur Datentreiber und Test Kindknoten gibt, dann wird der Datentreiber in einen ausgeschalteten Test umgewandelt.
   (b) Sonst werden alle Kinder in einen neuen Testfall gepackt.

4. Test in Testfall
   (a) Datentreiber in ausgeschaltete Sequenz

58.3 Zusätzliche Konvertierungen unter Extrasequenzen

Folgende Konvertierungen sind nur unter Extrasequenzen möglich:

58.3.1 Konvertierungen ohne Seiteneffekte

1. Sequenz in Aufräumen
2. Sequenz in Vorbereitung
3. Sequenz in Testfall
4. Aufräumen in Prozedur
5. Vorbereitung in Prozedur
6. Prozedur in Sequenz
7. Prozedur in Testfall

58.3.2 Konvertierungen mit Seiteneffekten

Die folgenden Konvertierungsmöglichkeiten ändern zusätzlich noch den Typen einiger Kindknoten:

1. Testfallsatz in Package
   (a) Aufräumen in ausgeschaltete Prozedur
   (b) Datentreiber in ausgeschaltete Prozedur
   (c) Bezug auf Abhängigkeit in ausgeschaltete Abhängigkeit, die den vorigen Bezug auf Abhängigkeit enthält
   (d) Vorbereitung in ausgeschaltete Prozedur

   (e) Test in Prozedur
   (f) Testfall in Prozedur
   (g) Testaufruf in ausgeschaltete Prozedur, die den Testaufruf enthält
2. Testfall in Prozedur
   (a) Aufräumen in ausgeschaltete Sequenz
   (b) Abhängigkeit in ausgeschaltete Sequenz
   (c) Bezug auf Abhängigkeit in ausgeschaltete Sequenz
   (d) Vorbereitung in ausgeschaltete Sequenz
3. Test in Prozedur
   (a) Datentreiber in ausgeschaltete Sequenz
4. Test in Package
   (a) Alle Kindknoten werden in eine Prozedur gepackt.
5. Test in Sequenz
   (a) Datentreiber in ausgeschaltete Sequenz
6. Package in Testfallsatz
   (a) Package in Testfallsatz
   (b) Prozedur in Testfall

Kapitel 59

Details des Algorithmus zum Bildvergleich

3.3+

59.1 Einführung

Der klassische Check Abbild(828) Knoten ist nur minimal tolerant gegenüber Abweichungen. Mit diesem Algorithmus, der einen Vergleich Pixel für Pixel durchführt, ist es nicht möglich, Bilder zu vergleichen, die nicht-deterministisch generiert werden oder nicht die exakt gleiche Größe aufweisen.

Mittels dem Attribut Algorithmus zum Bildvergleich(831) ist es möglich, einen speziellen Algorithmus zu definieren, der tolerant gegenüber gewissen Änderungen Abbild-Vergleiche durchführt. Das Attribut muss mit der Algorithmus-Definition in Form von algorithm=<algorithm> starten. Anschließend folgen alle notwendigen Parametern, die jeweils mit Semikolons voneinander getrennt sind.
Die Parameter können in beliebiger Reihenfolge angegeben werden, auch die Verwendung von Variablen ist erlaubt.

```
algorithm=<algo>;parameter1=value1;parameter2=value2;
expected=$(expected)
```

3.5.1+ Seit QF-Test 3.5.1 muss die Definition nicht mehr mit algorithm=<algorithm> starten, sondern kann einfach mit <algorithm> beginnen.
Es ist auch nicht mehr notwendig, den Parameter 'expected' zu definieren. QF-Test benutzt einen Default-Wert, wenn keiner angegeben ist. Mehr Informationen dazu finden Sie unten.

Eine detaillierte Beschreibung der verfügbaren Algorithmen und ihrer Parameter folgt im folgenden Abschnitt. Zum besseren Verständnis werden die Algorithmen beispielhaft auf dieses Bild angewendet:

Abbildung 59.1: Ausgangsbild

Im Protokoll (vgl. Abschnitt 7.1[138]) eines fehlgeschlagenen Checks haben Sie die Möglichkeit die Ergebnisbilder des benutzten Algorithmus sowie die Übereinstimmungswahrscheinlichkeit zu analysieren.
Wenn die Option Erfolgreiche tolerante Abbildvergleiche in das Protokoll schreiben[589] aktiviert ist, werden alle toleranten Abbildvergleiche in das Protokoll geschrieben und stehen für weitere Analysen zur Verfügung.

59.2 Beschreibung der Algorithmen

59.2.1 Klassischer Bildvergleich

Beschreibung

Der klassische Bildvergleich prüft für jedes einzelne Pixel, ob der erwartete Farbwert dem erhaltenen Farbwert entspricht. Sollte mindestens ein erhaltenes Pixel nicht dem erwarteten Pixel entsprechen, so schlägt der Check fehl. Die Option Erlaubte Abweichung beim Check von Abbildern[545] erlaubt dabei das Setzen einer Toleranzschwelle für den Pixelvergleich.

Einsatzzweck

Dieser pixelbasierte Vergleich ist immer dann geeignet, wenn ein nahezu exaktes Abbild erwartet wird und nur geringe oder keine Toleranzen oder Abweichungen zugelassen sind. Sobald Ihre Anwendung jedoch das Aussehen der zu prüfenden Komponente nicht mit jedem Testlauf identisch generiert, ist dieser Algorithmus nicht geeignet.

Beispiel

Der klassische Bildvergleich führt keine Manipulationen durch, daher sieht das Ergebnis aus wie das Ausgangsbild.

Abbildung 59.2: Klassischer Bildvergleich

Der klassische Bildvergleich wird verwendet, wenn das Algorithmus zum Bildvergleich(831) Attribute leer ist.

59.2.2 Pixelbasierter Vergleich

Beschreibung

Dieser Algorithmus ist ähnlich zum klassischen Bildvergleich mit dem Unterschied, dass er eine gewisse prozentuale Anzahl von abweichenden Pixeln akzeptiert. Zuerst teilt er jedes Pixel in seine drei Farbbestandteile rot, grün und blau. Anschließend wird jeder dieser drei Werte mit den jeweils erwarteten Farbwerten verglichen. Das Ergebnis ist die Anzahl der identischen Pixel geteilt durch die Gesamtanzahl der Pixel.

Einsatzzweck

Falls Ihre zu prüfenden Bilder nicht mit jedem Lauf identisch generiert werden, Sie aber eine prozentuale Anzahl von abweichenden Pixeln akzeptieren, könnte dieser Algorithmus sinnvoll sein.
Er ist jedoch nicht brauchbar, wenn das erhaltene Abbild gegenüber dem erwarteten Abbild Verschiebungen oder Zerrungen aufweist.

Beispiel

Da der pixelbasierte Vergleich keine Manipulationen durchführt, sieht das Ergebnis bei Verwendung des beispielhaften Algorithmus
`algorithm=identity;expected=0.95`
aus wie das Ausgangsbild.

Abbildung 59.3: Pixelbasierter Vergleich

Parameter

algorithm=identity
Gibt an, dass der 'Pixelbasierte Vergleich' verwendet werden soll.

expected (optional, aber empfohlen)
Gibt an, welche Mindestübereinstimmung erwartet wird.
Gültig ist ein Wert zwischen 0.0 und 1.0. Falls nicht definiert, wird 0.98 verwendet.

resize (optional)
Gibt an, dass das erhaltene Bild auf die gleiche Größe gebracht wird wie das erwartete Bild.
Gültige Werte sind "true" und "false".

find (optional)
Gibt an, dass eine Bild-in-Bild Suche durchgeführt werden soll.
Eine genaue Beschreibung dieses Parameters ist in Abschnitt 59.3.1 (1323) erläutert.

59.2.3 Pixelbasierte Ähnlichkeitsanalyse

Beschreibung
Dieser Algorithmus zerlegt jedes Pixel in seine drei Farbbestandteile rot, grün und blau. Anschließend wird jeder dieser drei Werte mit den jeweils erwarteten Farbwerten verglichen und eine prozentuale Ähnlichkeit ermittelt. Die prozentualen Abweichungen werden aufsummiert und bilden nach Berechnung des Durchschnitts über die frei Farbwerte sowie über alle Pixel des Abbildes die gesamte Ähnlichkeit. Dieser erhaltene Wert wird mit einem erwarteten Wert verglichen und bestimmt somit den Erfolg des Vergleichs.

Einsatzzweck
Falls Ihre zu prüfenden Abbilder nicht mit jedem Lauf identisch sind, Sie aber

eine gewisse Abweichung akzeptieren, ist dieser Algorithmus ein möglicher Kandidat für den Einsatz.
Falls für einzelne Pixel eine große Farbabweichung akzeptiert wird, solange die mittlere Abweichung in einem definierten Rahmen bleibt, ist dieser Algorithmus ebenfalls geeignet.
Ungeeignet ist er jedoch, wenn das erhaltene Abbild gegenüber dem erwarteten Abbild Verschiebungen oder Zerrungen aufweist.

Beispiel

Da die pixelbasierte Ähnlichkeitsanalyse keine Manipulationen durchführt, sieht das Ergebnis bei Verwendung des beispielhaften Algorithmus
`algorithm=similarity;expected=0.95`
aus wie das Ausgangsbild.

Abbildung 59.4: Pixelbasierte Ähnlichkeitsanalyse

Parameter

algorithm=similarity
Gibt an, dass die 'Pixelbasierte Ähnlichkeitsanalyse' verwendet werden soll.

expected (optional, aber empfohlen)
Gibt an, welche Mindestübereinstimmung erwartet wird.
Gültig ist ein Wert zwischen 0.0 und 1.0. Falls nicht definiert, wird 0.98 verwendet.

resize (optional)
Gibt an, dass das erhaltene Bild auf die gleiche Größe gebracht wird wie das erwartete Bild.
Gültige Werte sind "true" und "false".

find (optional)
Gibt an, dass eine Bild-in-Bild Suche durchgeführt werden soll.
Eine genaue Beschreibung dieses Parameters ist in Abschnitt 59.3.1 (1323) erläutert.

59.2.4 Blockbildung mit Vergleich

Beschreibung

Bei Verwendung dieses Algorithmus werden quadratische Blöcke mit frei wählbarer Größe gebildet. Der Farbwert der einzelnen Blöcke berechnet sich dabei aus dem Durchschnitt der Farbwerte, die der jeweilige Block überdeckt. Falls die Breite beziehungsweise Höhe des untersuchten Abbilds nicht ein Vielfaches der Blockgröße ist, werden die Blöcke am rechten beziehungsweise unteren Rand dementsprechend abgeschnitten und mit niedrigerem Gewicht gewertet.
Die erhaltenen Blöcke werden gegen die erwarteten Blöcke geprüft. Als Ergebnis wird der Anteil der identischen Blöcke im Vergleich zur Gesamtanzahl unter Berücksichtigung eventuell gewichteter Randblöcke ermittelt.

Einsatzzweck

Dieser Algorithmus ist dafür vorgesehen, wenn ein Teil Ihres zu prüfenden Abbildes sich von Lauf zu Lauf ändert, aber der Rest des Bildes jedes mal identisch ist.

Beispiel

Der beispielhafte Algorithmus
`algorithm=block;size=10;expected=0.95`
führt zu folgendem Ergebnisbild:

Abbildung 59.5: Blockbildung mit Vergleich

Parameter

algorithm=block
Gibt an, dass ’Blockbildung mit Vergleich’ verwendet werden soll.

size
Definiert die zu verwendende Größe der einzelnen Blöcke.
Gültig ist ein Wert zwischen 1 und der der Bildgröße.

expected (optional, aber empfohlen)
Gibt an, welche Mindestübereinstimmung erwartet wird.
Gültig ist ein Wert zwischen 0.0 und 1.0. Falls nicht definiert, wird 0.98 verwendet.

resize (optional)
Gibt an, dass das erhaltene Bild auf die gleiche Größe gebracht wird wie das erwartete Bild.
Gültige Werte sind ”true” und ”false”.

find (optional)
Gibt an, dass eine Bild-in-Bild Suche durchgeführt werden soll.
Eine genaue Beschreibung dieses Parameters ist in Abschnitt 59.3.1(1323) erläutert.

59.2.5 Blockbildung mit Ähnlichkeitsanalyse

Beschreibung
Auch bei diesem Algorithmus werden zunächst quadratische Blöcke mit frei wählbarer Größe gebildet. Der Farbwert der einzelnen Blöcke berechnet sich dabei aus dem Durchschnitt der Farbwerte, die der jeweilige Block überdeckt. Falls die Breite beziehungsweise Höhe des untersuchten Abbilds nicht ein Vielfaches der Blockgröße ist, werden die Blöcke am rechten beziehungsweise unteren Rand dementsprechend abgeschnitten und mit niedrigerem Gewicht gewertet.
Die erhaltenen Blöcke werden gegen die erwarteten Blöcke geprüft. Anschließend werden die Farbwerte der Blöcke miteinander verglichen und die Ähnlichkeit analysiert. Die Mittelwerte der Ähnlichkeiten kombiniert mit der Gewichtung der jeweiligen Blöcke bilden das Ergebnis.

Einsatzzweck
Mittels der Mittelwertsbildung einzelner Quadrate sowie eine Ähnlichkeitsuntersuchung ebendieser eignet sich der Algorithmus allgemein zum Vergleich von variierenden Bildern, die jedoch eine übereinstimmende Verteilung ähnlicher Farben aufweisen.

Beispiel
Der beispielhafte Algorithmus
`algorithm=blocksimilarity;size=5;expected=0.95`
führt zu folgendem Ergebnisbild:

Abbildung 59.6: Blockbildung mit Ähnlichkeitsanalyse

Parameter

algorithm=blocksimilarity
Gibt an, dass 'Blockbildung mit Ähnlichkeitsanalyse' verwendet werden soll.

size
Definiert die zu verwendende Größe der einzelnen Blöcke.
Gültig ist ein Wert zwischen 1 und der der Bildgröße.

expected (optional, aber empfohlen)
Gibt an, welche Mindestübereinstimmung erwartet wird.
Gültig ist ein Wert zwischen 0.0 und 1.0. Falls nicht definiert, wird 0.98 verwendet.

resize (optional)
Gibt an, dass das erhaltene Bild auf die gleiche Größe gebracht wird wie das erwartete Bild.
Gültige Werte sind "true" und "false".

find (optional)
Gibt an, dass eine Bild-in-Bild Suche durchgeführt werden soll.
Eine genaue Beschreibung dieses Parameters ist in Abschnitt 59.3.1 (1323) erläutert.

59.2.6 Häufigkeitsanalyse mittels Histogramm

Beschreibung

Eine Farbhäufigkeitsanalyse zerlegt das Abbild zuerst in seine drei Grundfarbtöne rot, grün und blau. Anschließend werden für jedes Pixel die Farbwerte dieser drei Farbtöne ermittelt. Diese Farbwerte werden in eine relativ frei wählbaren Anzahl von Kategorien aufgeteilt. Anschließend wird der Füllgrad dieser Kategorien ermittelt. Das Ergebnis dieses Algorithmus ist der Vergleich der Füllgrade jeder Kategorie und somit implizit ein Vergleich der Häufigkeiten von Farbkategorien.

Einsatzzweck

Histogramme dienen sehr unterschiedlichen Zwecken. Es ist mit ihnen zum Beispiel möglich, Farbtendenzen zu ermitteln oder Helligkeitsuntersuchungen durchzuführen.
Nicht geeignet sind sie jedoch zum Vergleich von sehr einfarbigen Bildern.

Beispiel

Der beispielhafte Algorithmus
`algorithm=histogram;buckets=64;expected=0.95`
führt zu folgendem Ergebnisbild:

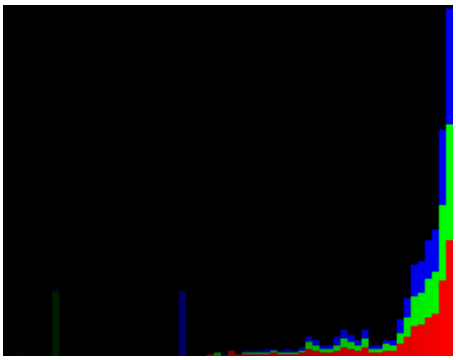


Abbildung 59.7: Häufigkeitsanalyse mittels Histogramm

Parameter

algorithm=histogram

Gibt an, dass eine 'Häufigkeitsanalyse mittels Histogramm' durchgeführt werden soll.

buckets

Gibt an, wieviele Farbabstufungen verwendet werden sollen.
Gültig ist eine Potenz zur Basis 2 von 2 bis 256.

expected (optional, aber empfohlen)

Gibt an, welche Mindestübereinstimmung erwartet wird.
Gültig ist ein Wert zwischen 0.0 und 1.0. Falls nicht definiert, wird 0.98 verwendet.

resize (optional)

Gibt an, dass das erhaltene Bild auf die gleiche Größe gebracht wird wie das erwartete Bild.
Gültige Werte sind "true" und "false".

find (optional)
Gibt an, dass eine Bild-in-Bild Suche durchgeführt werden soll.
Eine genaue Beschreibung dieses Parameters ist in Abschnitt 59.3.1(1323) erläutert.

59.2.7 Bildanalyse mittels Diskreter Kosinustransformation

Beschreibung
Die Diskrete Kosinustransformation (DCT) ist eine reellwertige, diskrete, lineare, orthogonale Transformation, die das diskrete Signal vom Ortsbereich in den Frequenzbereich transformiert.
Nachdem ein Bild transformiert wurde, lassen sich niederwertige (schnell schwingende) Frequenzen eliminieren und nur die höherwertigen (langsam schwingende) Frequenzen sowie der Gleichanteil (0. Frequenz = 0*cos(x) + y) betrachten. Sie können frei bestimmen, wieviele Frequenzen jeweils pro Farbkanal zum Bildvergleich herangezogen werden und mit welcher Toleranz zwei Kosinus-Schwingungen als identisch angesehen werden. Bei der Berechnung der Übereinstimmungswahrscheinlickeit werden niederwertige Frequenzen mit deutlich geringerem Anteil gewichtet als höherwertige Frequenzen.

Einsatzzweck
Die Diskrete Kosinustransformation eignet sich im Prinzip für alle Arten von Bildvergleichen, die eine gewisse Toleranz erfordern, da sowohl Farbverteilungen, Farbhäufungen als auch Farbwiederholungen berücksichtigt werden. Je mehr Frequenzen zur Untersuchung herangezogen werden, desto schärfer wird der Bildvergleich durchgeführt.

Beispiel
Der beispielhafte Algorithmus
`algorithm=dct;frequencies=20;tolerance=0.1;expected=0.95`
führt zu folgendem Ergebnisbild:

Abbildung 59.8: Bildanalyse mittels Diskreter Kosinustransformation

Parameter

algorithm=dct
Gibt an, dass eine 'Bildanalyse mittels Diskreter Kosinustransformation' durchgeführt werden soll.

frequencies
Gibt an, wie viele Frequenzen betrachtet werden sollen.
Gültig ist ein Wert zwischen 0 (nur Gleichanteil) und der Fläche des Bildes.
Je weniger Frequenzen betrachtet werden, desto unschärfer wird der Vergleich. Außerdem gilt zu beachten, dass die Schärfe einer bestimmten Anzahl von Frequenzen von der Bildgröße abhängt.

tolerance
Gibt an, bis zu welcher (nicht linearen) Abweichung jede einzelne Frequenz als gleich angesehen wird.
Gültig ist ein Wert zwischen 0.0 und 1.0.
Der Wert 1.0 bedeutet, dass jedes Bild akzeptiert wird, da je Frequenz die maximale Abweichung als gültig erlaubt ist. Ein Wert von 0.0 bedeutet, dass keine Abweichung pro Frequenz akzeptiert wird und der Vergleich somit nur identische Schwingungen als identisch akzeptiert. Der Wert 0.1 gilt als Standard, da dieser nur extrem ähnliche Frequenzen als identisch akzeptiert.

expected (optional, aber empfohlen)
Gibt an, welche Mindestübereinstimmung erwartet wird.
Gültig ist ein Wert zwischen 0.0 und 1.0. Falls nicht definiert, wird 0.98 verwendet.

resize (optional)
Gibt an, dass das erhaltene Bild auf die gleiche Größe gebracht wird wie das erwartete Bild.
Gültige Werte sind "true" und "false".

find (optional)
Gibt an, dass eine Bild-in-Bild Suche durchgeführt werden soll.
Eine genaue Beschreibung dieses Parameters ist in Abschnitt 59.3.1(1323) erläutert.

59.2.8 Blockbildung zur Bildanalyse mittels Diskreter Kosinustransformation

Beschreibung
Bei Verwendung dieses Algorithmus wird das Bild in einzelne frei definierbare quadratische Bereiche eingeteilt (vgl. Abschnitt 59.2.4(1314)). Anschließend wird für

jeden dieser Bereiche eine Diskrete Kosinustransformation (vgl. Abschnitt 59.2.7[1318]) durchgeführt. Der Mittelwert der Ergebnisse der einzelnen Diskreten Kosinustransformationen aller Bereiche sowie gewichteten Randbereiche bildet das Gesamtergebnis.

Einsatzzweck

Die Diskrete Kosinustransformation über das gesamte Bild reagiert stark auf Helligkeitsabweichungen, die im zentralen Bereich des Bildes auftreten, da damit der Gleichanteil (0. Frequenz), die den am höchsten gewichteten Anteil darstellt, stark variiert. Die Einteilung in einzelne Bereiche umgeht dieses Phänomen, da nur noch der betroffene Bereich eine starke Abweichung aufweist, die anderen Bereiche jedoch davon nicht betroffen sind.

Beispiel

Der beispielhafte Algorithmus

```
algorithm=dctblock;size=32;frequencies=4;tolerance=0.1;
expected=0.95
```

führt zu folgendem Ergebnisbild:

Abbildung 59.9: Blockbildung zur Bildanalyse mittels Diskreter Kosinustransformation

Parameter

algorithm=dctblock

Gibt an, dass eine 'Blockbildung zur Bildanalyse mittels Diskreter Kosinus Transformation' durchgeführt werden soll.

frequencies

Gibt an, wie viele Frequenzen betrachtet werden sollen.

Gültig ist ein Wert zwischen 0 (nur Gleichanteil) und der Fläche eines Blockes.

Je weniger Frequenzen betrachtet werden, desto unschärfer wird der Vergleich. Außerdem gilt zu beachten, dass die Schärfe einer bestimmten Anzahl von Frequenzen von der Bildgröße abhängt.

tolerance
Gibt an, bis zu welcher (nicht linearen) Abweichung jede einzelne Frequenz als gleich angesehen wird.
Gültig ist ein Wert zwischen 0.0 und 1.0.
Der Wert 1.0 bedeutet, dass jedes Bild akzeptiert wird, da je Frequenz die maximale Abweichung als gültig erlaubt ist. Ein Wert von 0.0 bedeutet, dass keine Abweichung pro Frequenz akzeptiert wird und der Vergleich somit nur identische Schwingungen als identisch akzeptiert. Der Wert 0.1 gilt als Standard, da dieser nur extrem ähnliche Frequenzen als identisch akzeptiert.

size
Definiert die zu verwendende Größe der einzelnen Blöcke.
Gültig ist eine Potenz zur Basis 2 zwischen 2 und der Bildgröße.

expected (optional, aber empfohlen)
Gibt an, welche Mindestübereinstimmung erwartet wird.
Gültig ist ein Wert zwischen 0.0 und 1.0. Falls nicht definiert, wird 0.98 verwendet.

resize (optional)
Gibt an, dass das erhaltene Bild auf die gleiche Größe gebracht wird wie das erwartete Bild.
Gültige Werte sind ”true” und ”false”.

find (optional)
Gibt an, dass eine Bild-in-Bild Suche durchgeführt werden soll.
Eine genaue Beschreibung dieses Parameters ist in Abschnitt 59.3.1(1323) erläutert.

59.2.9 Bilinearer Filter

Beschreibung
Dieser Algorithmus verkleinert das zu prüfende Bild auf eine wählbare prozentuale Größe. Anschließend wird das Bild wieder auf die ursprüngliche Größe vergößert unter Verwendung eines bilinearen Filters. Dieser Filter bewirkt eine Unschärfe, da beim Strecken des Bildes Nachbarpixel berücksichtigt werden und somit ein Farbverlauf entsteht.
Die transformierten Bilder werden anschließend einem pixelbasierten Ähnlichkeitsvergleich unterzogen.

Einsatzzweck
Abhängig von der gewählten Schärfe gehen beliebig viele Bildinformationen verloren. Somit ist dieser Algorithmus, je nach Schärfe, für nahezu beliebige Einsatzzwecke geeignet.

Beispiel

Der beispielhafte Algorithmus

```
algorithm=bilinear;sharpness=0.2;expected=0.95
```

führt zu folgendem Ergebnisbild:

Abbildung 59.10: Bilinearer Filter

Parameter

algorithm=bilinear

Gibt an, dass der ’Bilinearer Filter’ auf die Bilder angewendet werden soll.

sharpness

Gibt an, wie scharf das Ergebnis des bilinearen Filters sein soll.
Gültig ist ein Wert zwischen 0.0 (kompletter Informationsverlust) und 1.0 (kein Informationsverlust).
Die Schärfe ist eine lineare Angabe, so geht bei einem Wert von 0.5 genau die Hälfte (plusminus Rundung auf ganze Pixel) der Bildinformation verloren.

expected (optional, aber empfohlen)

Gibt an, welche Mindestübereinstimmung erwartet wird.
Gültig ist ein Wert zwischen 0.0 und 1.0. Falls nicht definiert, wird 0.98 verwendet.

resize (optional)

Gibt an, dass das erhaltene Bild auf die gleiche Größe gebracht wird wie das erwartete Bild.
Gültige Werte sind ”true” und ”false”.

find (optional)

Gibt an, dass eine Bild-in-Bild Suche durchgeführt werden soll.
Eine genaue Beschreibung dieses Parameters ist in Abschnitt 59.3.1(1323) erläutert.

59.3 Beschreibung der speziellen Funktionen

59.3.1 Bild-in-Bild Suche

Beschreibung

Die Bild-in-Bild Suche ermöglicht es, ein erwartetes Abbild innerhalb eines (größeren) Gesamtbildes zu suchen. Der Vergleich ist erfolgreich, wenn das erwartete Abbild irgendwo innerhalb des erhaltenen Abbildes mit Hilfe des definierten Algorithmus gefunden wurde.
Außerdem kann man die Position des Treffers ermitteln.
Folgende Parameterkombinationen sind gültig: `find=best` oder `find=anywhere`
`find=best(resultX, resultY)` oder `find=anywhere(resultX, resultY)`

Einsatzzweck

Die Bild-in-Bild Suche ermöglicht es, Abbilder zu vergleichen, wenn man die genaue Position nicht kennt. Da die Suche mit jedem Algorithmus kombiniert werden kann, ist sie in beliebigen Situationen anwendbar.

Beispiel

Der beispielhafte Algorithmus
`algorithm=similarity;expected=0.95;find=best(resultX,resultY)`
verwendet die pixelbasierte Ähnlichkeitsanalyse (vgl. Abschnitt 59.2.3(1312)) um ein Abbild des Q auf dem Gesamtbild zu suchen. Das erhaltene Abbild inklusive Bereichsmarkierung kann im Protokoll eingesehen werden und sieht wie folgt aus. Außerdem werden die Variablen resultX und resultY auf die Positionsinformationen gesetzt.

Abbildung 59.11: Bild-in-Bild Suche: Erwartetes Abbild

Abbildung 59.12: Bild-in-Bild Suche: Erhaltenes Abbild

Parameter

find=best
Gibt an, dass der beste Treffer als Ergebnis gewertet wird.

find=anywhere
Gibt an, dass der erste Treffer, der besser als die erwartete Wahrscheinlichkeit ist, als Ergebnis gewertet wird.
Da die Bild-in-Bild Suche mit mehreren Threads durchgeführt wird, kann das Ergebnis hier nicht deterministisch sein.

resultX
`resultX` ist ein beliebiger Variablenname. Nach dem Vergleich beschreibt er die x-Position des gefundenen Bildes.
Wenn eine Variable für die x-Position definiert ist, dann muss auch eine Variable für die y-Position angegeben sein (siehe Syntax oben).

resultY
`resultY` ist ein beliebiger Variablenname. Nach dem Vergleich beschreibt er die y-Position des gefundenen Bildes.
Wenn eine Variable für die y-Position definiert ist, dann muss auch eine Variable für die x-Position angegeben sein (siehe Syntax oben).

Kapitel 60

Resultatslisten

3.2+

60.1 Einführung

Suchoperationen wie das Finden von Referenzen oder eine Suche nach einem bestimmten Wert können sehr viele Resultate hervorbringen. Genauso verhält es sich mit Umbenennungen von Komponenten bzw. Prozeduren, die sehr viele Anpassungen erfordern können. Um Ihnen nun einen besseren Überblick über die Treffer zu geben, zeigt QF-Test eine Resultatsliste am Ende jeder Such- bzw. Anpassungsaktion an. Diese Liste enthält alle Knoten, welche von der entsprechenden Operation betroffen sind. Neben einen besseren Überblick erlaubt Ihnen die Liste auch Massenoperationen, wie das Markieren mehrerer Knoten oder das Löschen aller gefundenen Knoten, schnell durchzuführen. Diese Aktionen können mittels Bearbeiten Menü oder Kontextmenü der Tabelle ausgeführt werden.

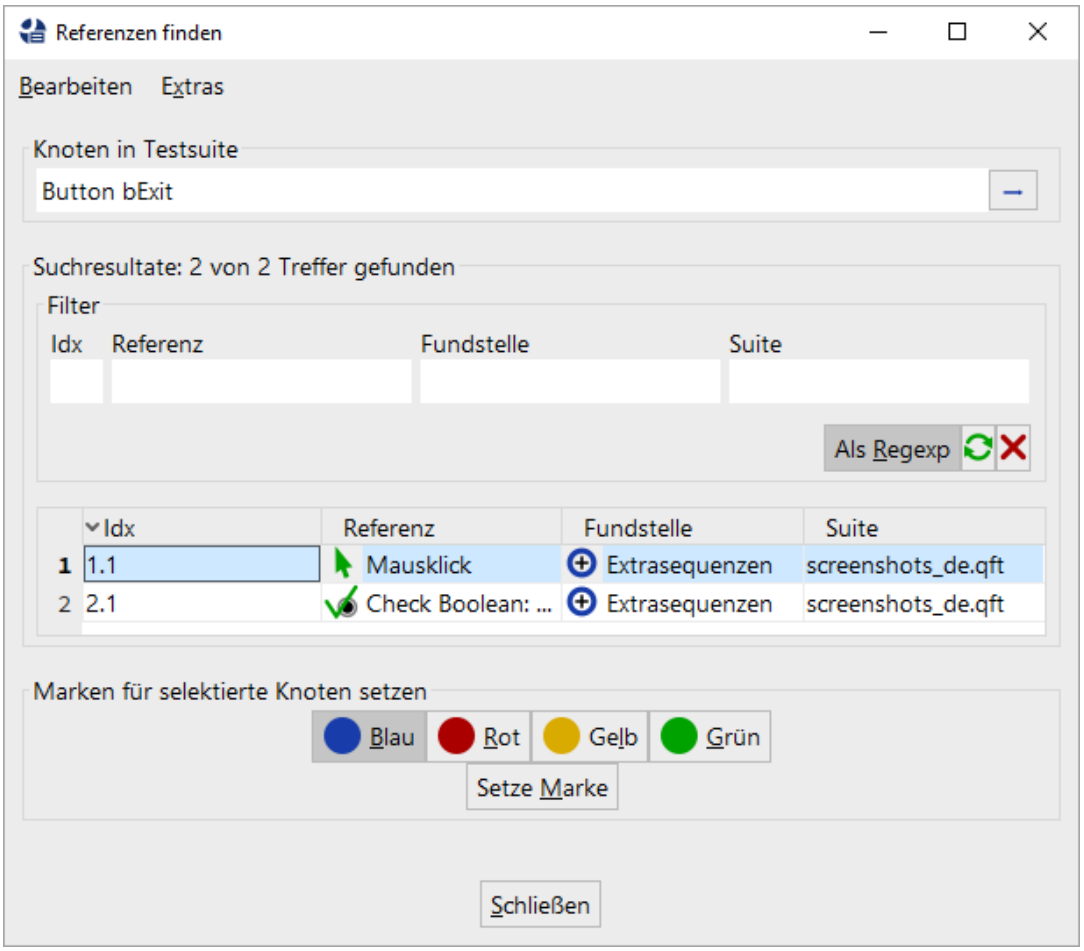


Abbildung 60.1: Ergebnis von 'Referenzen finden'

Die Tabelle verfügt auch über ein eigenes Kontextmenü. Dieses Menü ermöglicht Ihnen unterschiedliche Aktionen auf den angezeigten Knoten durchzuführen. Wenn Sie einen Eintrag in der Resultatsliste selektieren und einen Rechtsklick durchführen, können Sie diese Aktionen, z.B. zum Zielknoten zu springen oder diesen auszuschalten, ausführen. Je nach Art des Resultats finden Sie auch spezifische Aktionen, die nur im jeweiligen Zusammenhang Sinn machen. Eine Liste aller möglichen Aktionen finden Sie im Abschnitt 60.2(1327).

Resultatslisten werden von folgenden Aktionen angezeigt:

- Wenn Sie *Ergebnisliste anzeigen* auf dem Such- bzw. Ersetzendialog anklicken.
- Als Ergebnis der Referenzen finden Aktion auf Knoten wie Prozeduren, Tests oder Komponenten.
- Als Ergebnis jedes Kommandos, welches mehrere Knoten anpasst.

- Als Ergebnis eines Ersetzenvorganges vom Ersetzendialog.
- Als Ergebnis der Analyseaktionen im Menü Weitere Knotenoperationen.
- Wenn Sie die Breakpointliste unter Debugger→Liste aller Breakpoints... öffnen.
- Wenn Sie doppelte Komponenten einfügen oder eine Testsuite mit solchen Duplikaten geöffnet wird.
- Wenn eine Knotenkonvertierung den Typ von Kindknoten verändert.
- Wenn Sie Bearbeiten→Fehlerliste öffnen im Protokoll ausführen.
- Fehler, die beim Aktualisieren von Komponenten mittels Komponenten aktualisieren aufgetreten sind.

Wenn Sie sehr viele Treffer vorfinden, können Sie diese auch filtern und sich so nur auf die Ergebnisse konzentrieren, die Ihrem Filter entsprechen. Wenn Sie den Filter zurücksetzen, sehen Sie wieder alle Treffer.

60.2 Spezielle Listenaktionen

60.2.1 Alle Listen

Folgende Aktionen können Sie auf jeder Liste ausführen:

- Knoten markieren
- Knoten ein-/ausschalten
- Zum Zielknoten in der Testsuite springen
- Einen Breakpoint setzen
- Knoten aus der Liste löschen
- Knoten aus der Testsuite löschen
- Das Resultat in einer `.qcv` Datei speichern, siehe Abschnitt 60.3(1328) für Details.
- Wenn die zugrundeliegenden Aktionen von einem Knoten ausgingen, können Sie auch zu diesem springen.

60.2.2 Ersetzen

Sie können einen Ersetzvorgang auch nur auf die selektierten Einträge in der Tabelle anwenden.

60.2.3 Fehlerliste

Die Fehlerliste im Protokoll ermöglicht Ihnen, mehrere fehlgeschlagene Checks auf einen Schlag zu aktualisieren. Natürlich erlaubt diese Liste Ihnen nicht, Knoten aus dem Protokoll zu löschen.

60.3 Resultate exportieren und laden

3.5+

Im Falle von sehr vielen Ergebnissen und entsprechend großen Listen, für deren Bearbeitung Sie mehrere Sitzungen benötigen, können Sie die Ergebnisse der Liste mittels der Menüaktion Extras→Exportieren speichern. Dieser Export erzeugt eine `.qcv` Datei, welche mehr oder weniger eine CSV-Datei ist, die interne Informationen von QF-Test enthält. Um später mit der Bearbeitung fortzufahren, können Sie mittels Operationen→Resultatsliste laden... diese `.qcv` nach QF-Test laden.

Kapitel 61

Generische Klassen

4.0+

QF-Test abstrahiert die Klassen von aufgezeichneten Komponenten von den konkreten technischen Klassen zu sogenannten generischen Klassen, welche für alle unterstützten GUI-Engines einheitlich sind. Mit diesem Konzept soll zum die Lesbarkeit und das Verständnis der Komponenten erhöht werden. Des weiteren kann damit erreicht werden, dass bereits erstellte Tests auch bei einem Wechsel der Technologie bzw. dem Parallelbetrieb mehrerer Technologien wiederverwendet werden können.

Neben der generischen Klasse zeichnet QF-Test auch generische Typen auf, welche eine konkretere Angabe über die Art der Komponenten beinhalten. Ein typischer Anwendungsfall hierfür sind Passwortfelder. Diese Felder sind von der generischen Klasse `TextField`. Allerdings besitzen Passwortfelder noch den speziellen Charakter, dass sie für Passworteingaben gedacht sind, deshalb hat ein Passwortfeld zusätzlich den Typ `PasswordField`. Die komplette generische Klasse lautet somit `TextField:PasswordField`. Mit diesen Typangaben können bei der Wiedererkennung mit generischen Klassen bestimmte Kategorien von Komponenten genauer eingegrenzt und somit genauer erkannt werden.

Ein großer Vorteil dieser Typen ist, dass sie im Web-Umfeld bei der Zuordnung generischer Klassen frei vom Anwender vergeben werden können und dann später, beim Mapping anderer generischer Klassen, darauf referenziert werden kann. Ein Beispiel hierzu finden Sie in CustomWebResolver – Tabelle(1096).

Insbesondere bei Web-Anwendungen bietet die Verwendung von generischen Klassen die folgenden Vorteile:

- Die Aufnahme zusätzlicher Wiedererkennungsmerkmale, speziell des Merkmal Attributs und `qfs:label*`-Varianten in Weitere Merkmale, je nach generischer Klasse.
- Die Bereitstellung klassenspezifischer Checks, zum Beispiel der Check einer ganzen Zeile bei einer Komponente mit der generischen Klasse `Table`.

- Die Indexierung von Unterelementen bei der Aufnahme,
  d.h. bei der Aufnahme eines Klicks auf eine Tabellenzelle wird nur für die Tabelle selbst ein Komponente Knoten angelegt und die Tabellenzelle im Mausevent Knoten über die Tabelle und einen Index referenziert.
- Die Aufnahme des generischen Typs, soweit sinnvoll.
- Bei Mausklicks entweder die Aufnahme der am besten geeignete Position oder der genauen Koordinaten, worauf der Klick später abgespielt werden soll.
- Allein durch die Zuordnung einer aufgenommenen Komponente zu einer generischen Klasse wird die Komponentenerkennung gegenüber unspezifischen HTML-Klassen geschärft.

Welche Informationen im Einzelnen abgespeichert werden, ist in den folgenden Abschnitten hinterlegt. Die in der Rubrik ”qfs:label*” verwendeten Begriffe werden in Tabelle 5.3(75) den konkreten Wiedererkennungsmerkmalen zugeordnet.

61.1 Accordion

Dient als Navigator zwischen Komponenten, die auf- bzw. zugeklappt werden können.

Web Das HTML-Mapping ist in Abschnitt 51.1.8(1107) beschrieben.

Art: Komponente

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Alle Elemente	Alle auswählbaren Einträge des Accordion	items	Alle
Ausgewähltes Element	Selektiertes Element	current_item	Alle
Alle Elemente mit Selektion	Alle Elemente inklusive deren Selektionstatus	items_with_selection	Zur Zeit nicht bei Web

Tabelle 61.1: Checktypen für Accordion

61.2 BusyPane

Wird über andere Komponenten gelegt, um diese zu sperren.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.3 Button

Dient als Schaltfläche, die eine Aktion auslöst und keinen speziellen Status besitzt.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Tooltip, Icon-Beschreibung, Label in der Nähe

qfs:type:

qfs:type	Beschreibung
Button:CalendarButton	Buttons innerhalb eines Calendar
Button:ComboBoxButton	Buttons innerhalb einer ComboBox
Button:PaginatorButton	Buttons zum Blättern in Paginator
Button:ScrollBarButton	Buttons zum Blättern in einem ScrollPane

Tabelle 61.2: Spezielle qfs:type Typen für Buttons

Zusätzliche Checks: Keine

61.4 Calendar

Dient zur Datumsauswahl.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks: Keine

61.5 CheckBox

Besitzt einen bestimmten Status. Typischerweise ist eine CheckBox an- und abhakbar.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
CheckBox:ComboListItemCheckBox	CheckBox innerhalb eines ComboListItems
CheckBox:ListItemCheckBox	CheckBox innerhalb eines ListItems
CheckBox:MenuItemCheckBox	CheckBox innerhalb eines MenuItems
CheckBox:TableCellCheckBox	CheckBox innerhalb einer Tabellenzelle
CheckBox:TreeNodeCheckBox	CheckBox innerhalb eines Baumknotens

Tabelle 61.3: Spezielle qfs:type Typen für CheckBoxes

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Ausgewählt-Status	Prüfung, ob Komponente selektiert ist	checked	Alle

Tabelle 61.4: Checktypen für CheckBoxes

61.6 Closer

Ein Klick auf den Closer, zum Beispiel den 'X' Button eines Fensters, schließt eine andere Komponente.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
Closer:AccordionCloser	Closer Element für Accordions
Closer:TabPanelCloser	Closer Element für Tabs
Closer:WindowCloser	Closer Element für Fenster

Tabelle 61.5: Spezielle qfs:type Typen für Closer

Zusätzliche Checks: Keine

61.7 ColorPicker

Dient zur Farbauswahl.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks: Keine

61.8 ComboBox

Besteht typischerweise aus einem Textfeld und einer Liste auswählbarer Optionen.

Web

Das HTML-Mapping ist in Abschnitt 51.1.7(1105) beschrieben.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Zugeordnetes Label, Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Placeholder

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Aktueller Wert	Aktuell selektierter Wert der ComboBox	value	Alle
Mögliche Werte	Auswählbare Werte der ComboBox	items	Alle

Tabelle 61.6: Checktypen für ComboBox

61.9 Divider

Trennt Bereiche mehrerer Komponenten voneinander ab und ist verschiebbar.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.10 Expander

Dient zum Auf- und Zuklappen von Komponenten, z.B. bei Baumknoten.

Art: Wird nicht aufgezeichnet SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Wird nicht direkt aufgezeichnet

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Keines

qfs:type:

qfs:type	Beschreibung
Expander:TreeNodeExpander	Auf/Zuklapp Expander eines TreeNode

Tabelle 61.7: Spezielle qfs:type Typen für Expander

Zusätzliche Checks: Keine

61.11 FileChooser

Dient zur Dateiauswahl. Besteht typischerweise aus einer Liste und mehreren Buttons.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.12 Graphics

Zeigt eine Grafik oder ein Diagramm an. Klicks können Aktionen auslösen, müssen aber nicht.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type: Keiner

Zusätzliche Checks: Keine

61.13 Icon

Zeigt ein Bild an. Klicks können Aktionen auslösen, müssen aber nicht.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
Icon:ComboListItemIcon	Icon eines ComboListItems
Icon:IndicatorIcon	Icon eines Indicators
Icon:ListItemIcon	Icon eines ListItems
Icon:MenuItemIcon	Icon eines MenuItems
Icon:TableCellIcon	Icon einer Tabellenzelle
Icon:TreeNodeIcon	Icon eines Baumknotens

Tabelle 61.8: Spezielle qfs:type Typen für Icon

Zusätzliche Checks: Keine

61.14 Indicator

Zeigt eine Meldung nach einer Eingabe an. Typischerweise wird diese nach Eingabe in ein Textfeld erscheinen. Kann auch ein Icon beinhalten.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
Indicator:ErrorIndicator	Zeigt einen Fehler an
Indicator:InfoIndicator	Zeigt eine Information an
Indicator:WarningIndicator	Zeigt eine Warnung an

Tabelle 61.9: Spezielle qfs:type Typen für Indicator

Zusätzliche Checks: Keine

61.15 Item

Element einer Liste. Kann auswählbar sein.

Art: Element oder Syntax SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type:

qfs:type	Beschreibung
Item:AccordionItem	Auswählbarer Tab eines Accordions
Item:ComboBoxListItem	Eintrag einer ComboBox-Liste
Item:ListItem	Eintrag einer Liste
Item:TabPanelItem	Auswählbarer Tab eines TabPanel

Tabelle 61.10: Spezielle qfs:type Typen für Item

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Text des Elements	Prüfung für den angezeigten Text	item	Alle
Element sichtbar	Prüfung, ob das Element sichtbar ist	item_visible	Alle
Selektion des Elements	Prüfung, ob das Element selektiert ist	item_selected	Alle
Ausgewählt-Status des Elements	Prüfung, ob z.B. ein Element angehakt ist	item_checked	Alle
Abbild des Elements	Prüfung für das Abbild des Elements	item_image	Nicht bei Web

Tabelle 61.11: Checktypen für Item

61.16 Label

Dient zur Anzeige einer Beschriftung. Klicks können Aktionen auslösen, müssen aber nicht.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
Label:CalendarLabel	Innerhalb eines Calendars
Label:Caption	Eine eindeutige Beschriftung
Label:PaginatorLabel	Anzeige der aktuellen Seite in Paginators
Label:PanelTitle	Titel eines Panels
Label:WindowTitle	Titel eines Fensters

Tabelle 61.12: Spezielle qfs:type Typen für Labels

Zusätzliche Checks: Keine

61.17 Link

Dient zur Navigation auf einen anderen Bereich.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
Link:BreadcrumbLink	Spezielle Breadcrumb Links für die Navigation

Tabelle 61.13: Spezielle qfs:type Typen für Links

Zusätzliche Checks: Keine

61.18 List

Zeigt mehrere Optionen an, welche auswählbar sein können.

Web Das HTML-Mapping ist in Abschnitt 51.1.6[1103] beschrieben.

Art: Komponente

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type:

qfs:type	Beschreibung
List:ComboBoxList	Liste einer ComboBox

Tabelle 61.14: Spezielle qfs:type Typen für List

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Alle Elemente	Alle Elemente in der Liste	items	Alle
Alle Elemente mit Selektion	Alle Elemente in der Liste, die selektiert sind	items_with_selection	Alle
Ausgewähltes Element	Gerade selektiertes Element	current_item	Alle

Tabelle 61.15: Checktypen für List

61.19 LoadingComponent

Dient zur Anzeige, dass Ihre Anwendung gerade beschäftigt ist, z.B. um etwas einzulesen.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.20 Maximizer

Maximiert einen Bereich, zum Beispiel ein Fenster.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
Maximizer:WindowMaximizer	Maximizer Element für Fenster

Tabelle 61.16: Spezielle qfs:type Typen für Maximizer

Zusätzliche Checks: Keine

61.21 Menu

Beinhaltet mehrere auswählbare Menüeinträge.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type:

qfs:type	Beschreibung
Menu:MenuBar	Eine Menüleiste
Menu:PopupMenu	Ein Menü, welches ein- und ausgeblendet wird

Tabelle 61.17: Spezielle qfs:type Typen für Menu

Zusätzliche Checks: Keine

61.22 MenuItem

Wird in Menüs angezeigt. Ein Klick löst normalerweise eine Aktion aus oder wechselt den Zustand der Applikation.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type: Keiner

Zusätzliche Checks: Keine

61.23 Minimizer

Minimiert einen Bereich, zum Beispiel ein Fenster.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
Minimizer:WindowMinimizer	Minimizer Element für Fenster

Tabelle 61.18: Spezielle qfs:type Typen für Minimizer

Zusätzliche Checks: Keine

61.24 ModalOverlay

Repräsentiert den Hintergrund eines pseudo-modalen Dialogs, der auf DOM-Ebene implementiert wird. Interaktionen mit Komponenten, die hinter einer `ModalOverlay`-Komponente liegen, lösen eine `ModalDialogException`[959] aus.

Hinweis Pseudo-modale Dialoge unterscheiden sich stark von echten modalen Dialog-Fenstern (Window:Dialog). Die `ModalOverlay`-Komponente ist dabei nicht der Dialog selbst, sondern der Bereich darum herum, der zum Wegfiltern von Mausevents außerhalb des Dialogs dient.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.25 Panel

Beinhaltet weitere Komponenten. Wird zur Organisation eines Fensters verwendet.

Art: Komponente SmartID: Klasse muss angegeben werden, außer bei Panel:TitledPanel

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Titel, falls vorhanden; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Titel

qfs:type:

qfs:type	Beschreibung
Panel:AccordionContent	Beinhaltet die Komponenten eines Accordions.
Panel:Breadcrumb	Beinhaltet Breadcrumb-Links für schnelle Navigation
Panel:CollapsiblePanel	Kann auf- und zu geklappt werden
Panel:Footer	Eine Fußzeile
Panel:Form	Ein Eingabeformular
Panel:Header	Eine Kopfzeile
Panel:Legend	Enthält Komponenten einer Legende für eine Grafik
Panel:MainPanel	Eindeutiges Hauptpanel der Anwendung
Panel:Paginator	Beinhaltet PaginatorButtons zum Blättern
Panel:OptionGroup	Beinhaltet mehrere RadioButtons
Panel:ScrollPanel	Verfügt über einen ScrollBar
Panel:TabPanelContent	Beinhaltet die Komponenten eines selektierten TabPanelItems eines TabPanel.
Panel:TitledPanel	Verfügt über einen dedizierten Titel

Tabelle 61.19: Spezielle qfs:type Typen für Panel

Zusätzliche Checks: Keine

61.26 Popup

Zeigt Komponenten an, wird typischerweise nach Klick auf einen Button angezeigt und hängt direkt bei diesem Button, z.B. bei einer ComboBox.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Titel

qfs:type:

qfs:type	Beschreibung
Popup:CalendarPopup	Beinhaltet einen aufklappbaren Kalender
Popup:ColorPickerPopup	Beinhaltet einen aufklappbaren ColorPicker
Popup:ComboBoxPopup	Beinhaltet die Liste der ComboBox

Tabelle 61.20: Spezielle qfs:type Typen für Popup

Zusätzliche Checks: Keine

61.27 ProgressBar

Zeigt den Fortschritt einer Aktion an.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Wert	Der aktuelle Wert	value	Alle

Tabelle 61.21: Checktypen für ProgressBar

61.28 RadioButton

Eine auswählbare Option, wird typischerweise bei Mehrfachauswahlen für eine bestimmte Option verwendet.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
RadioButton:ComboListItemRadioButton	RadioButton innerhalb eines ComboListItems
RadioButton:ListItemRadioButton	RadioButton innerhalb eines ListItems
RadioButton:MenuItemRadioButton	RadioButton innerhalb eines MenuItems
RadioButton:TableCellRadioButton	RadioButton innerhalb einer Tabellenzelle
RadioButton:TreeNodeRadioButton	RadioButton innerhalb eines Baumknotens

Tabelle 61.22: Spezielle qfs:type Typen für RadioButtons

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Ausgewählt-Status	Prüfung, ob Komponente selektiert ist	checked	Alle

Tabelle 61.23: Checktypen für RadioButtons

61.29 Restore

Stellt die Ursprungsgröße einer Komponente wieder her, zum Beispiel bei einem Fenster.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
Restore:WindowRestore	Restore Element für Fenster

Tabelle 61.24: Spezielle qfs:type Typen für Restore

Zusätzliche Checks: Keine

61.30 ScrollBar

Bereich, der für das Scrollen von Komponenten benutzt wird. Beinhaltet typischerweise einen Schieberegler.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.31 Separator

Trennt Bereiche mehrerer Komponenten voneinander ab und ist nicht verschiebbar.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.32 Sizer

Verändert die Größe einer Komponente, zum Beispiel bei einem Fenster.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type:

qfs:type	Beschreibung
Sizer:WindowSizer	Sizer Element für Fenster

Tabelle 61.25: Spezielle qfs:type Typen für Sizer

Zusätzliche Checks: Keine

61.33 Slider

Dient zur Auswahl eines Wertes mittels eines Schiebereglers.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Wert	Der aktuelle Wert	value	Alle

Tabelle 61.26: Checktypen für Slider

61.34 Spacer

Stellt einen Einrückeffekt dar, typischerweise bei Baumknoten.
Art: wird nicht direkt aufgezeichnet SmartID: Klasse muss angegeben werden
Koordinaten für Mausklick: wird nicht direkt aufgezeichnet
Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]
qfs:label*: Keines
qfs:type:

qfs:type	Beschreibung
Spacer:TreeNodeSpacer	Spacer eines TreeNode

Tabelle 61.27: Spezielle qfs:type Typen für Spacer

Zusätzliche Checks: Keine

61.35 Spinner

Dient zur Auswahl eines Wertes mittels Navigationsbuttons und eines Textfeldes.
Art: Komponente
Koordinaten für Mausklick: Genaue Koordinaten
Merkmal: Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]
qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip
qfs:type: Keiner
Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Wert	Der aktuelle Wert	value	Alle

Tabelle 61.28: Checktypen für Spinner

61.36 SplitPanel

Beinhaltet weitere Komponenten, kann diese aber in eigenständig vergrößerbare Bereiche unterteilen.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.37 Table

Dient zur Auswahl mehrerer Einträge, ist typischer mehrdimensional.

Web Das HTML-Mapping ist in Abschnitt 51.1.3[1096] beschrieben.

Art: Komponente

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Spalte	Alle Werte aus einer bestimmten Spalte	column	Alle
Spalte sichtbar	Prüfung, ob eine Spalte vorhanden ist	column_visible	Alle
Spalte mit Selektion	Alle Werte aus einer bestimmten Spalte inklusive Selektionstatus	column_with_selection	Alle
Spaltentitel	Der Titel der Spalte	header	Alle
Zeile	Alle Werte aus einer bestimmten Zeile	row	Alle

Tabelle 61.29: Checktypen für Table

Für die beiden Check-Typen `column` und `row` ist es möglich, nur einen Teil der Einträge zu prüfen. Die entsprechende Syntax ist in Name des Check-Typs(820) erläutert.

61.38 TableCell

Ein Wert in einer Tabelle, wird per Spalte und Zeile identifiziert.

Art: Element oder Syntax SmartID: Wird als Unterelement der Tabelle über Syntax referenziert

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Zelle	Der Text der Zelle	item	Alle
Zelle sichtbar	Prüfung, ob Zelle sichtbar ist	item_visible	Alle
Selektion der Zelle	Prüfung, ob Zelle selektiert ist	item_selected	Alle
Editable-Status der Zelle	Prüfung, ob Zelle editierbar ist	item_editable	Alle
Ausgewählt-Status der Zelle	Prüfung, ob Zelle angehakt ist	item_checked	Alle
Abbild der Zelle	Prüfung des Abbildes der Zelle	item_image	Alle

Tabelle 61.30: Checktypen für TableCell

61.39 TableFooter

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.40 TableHeader

Kopfzeile einer Tabelle.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Alle Elemente	Alle Einträge in der Titelleiste	items	Alle

Tabelle 61.31: Checktypen für TableHeader

61.41 TableHeaderCell

Spaltenname einer Tabelle.

Art: Element oder Syntax SmartID: Wird als Unterelement der Tabelle über Syntax referenziert

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Spaltentitel	Der Text des Spaltentitel	item	Alle
Titel sichtbar	Prüfung, ob Titel vorhanden ist	item_visible	Alle
Abbild des Titels	Prüfung des Abbilds des Titels	item_image	Alle

Tabelle 61.32: Checktypen für TableHeaderCell

61.42 TableRow

Zeile einer Tabelle.

Art: wird nicht aufgezeichnet SmartID: Nicht über SmartID ansprechbar

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.43 TabPanel

Dient als Navigator zwischen Komponenten. Diese Komponenten liegen in separaten Tabs und es kann jeweils nur ein Tab sichtbar sein.

Web Das HTML-Mapping ist in Abschnitt 51.1.8[1107] beschrieben.

Art: Komponente

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Alle Tabs	Alle Tabs des TabPanel	items	Alle
Ausgewählter Tab	Selektierter Tab	current_item	Alle

Tabelle 61.33: Checktypen für TabPanel

61.44 Text

Beinhaltet einen Fließtext. Es ist keine Eingabe möglich.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Placeholder

qfs:type:

qfs:type	Beschreibung
Text:IndicatorText	Text eines Indicators
Text:ToolTipText	Text eines Tooltips

Tabelle 61.34: Spezielle qfs:type Typen für Text

Zusätzliche Checks: Keine

61.45 TextArea

Ermöglicht mehrzeilige Textanzeigen bzw. -eingaben.

Art: Komponente

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Zugeordnetes Label, Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Placeholder

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Alle Zeilen	Alle Zeilen der TextArea	items	Alle
Editable	Prüfung, ob Komponente editierbar ist	editable	Alle

Tabelle 61.35: Checktypen für TextArea

61.46 TextField

Ermöglicht einzeilige Textanzeigen bzw. -eingaben.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Zugeordnetes Label, Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip, Placeholder

qfs:type:

qfs:type	Beschreibung
TextField:CalendarTextField	Eingabefeld eines Calendars
TextField:ComboBoxTextField	Textfelder von ComboBoxes
TextField:PasswordField	Textfelder für Passworteingaben
TextField:SpinnerTextField	Textfelder von Spinner

Tabelle 61.36: Spezielle qfs:type Typen für TextField

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Editable	Prüfung, ob Komponente editierbar ist	editable	Alle

Tabelle 61.37: Checktypen für TextField

61.47 Thumb

Stellt einen Schieberegler dar.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.48 ToggleButton

Dient als Schaltfläche, die einen speziellen Status besitzt. Ein Klick kann auch eine Aktion auslösen.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Zugeordnetes Label, Eigener Text, Label in der Nähe, Tooltip, Icon-Beschreibung

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Ausgewählt-Status	Prüfung, ob Komponente ausgewählt ist	checked	Alle

Tabelle 61.38: Checktypen für ToggleButtons

61.49 ToolBar

Stellt eine Werkzeugleiste dar. Beinhaltet typischerweise Menüeinträge oder wichtige Buttons für Aktionen.

Art: Komponente SmartID: Klasse muss angegeben werden

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks: Keine

61.50 ToolBarItem

Stellt ein klickbares Element in einer ToolBar dar.

Art: Komponente

Koordinaten für Mausklick: Am besten geeignete Position oder in die Mitte

Merkmal: Eigener Text oder Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Tooltip, Icon-Beschreibung, Label in der Nähe

qfs:type: Keiner

Zusätzliche Checks: Keine

61.51 ToolTip

Ist ein Fenster, welches als Hinweis geöffnet wird, z.B. wenn man mit der Maus darüber fährt.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Eigener Text, Tooltip; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Zugeordnetes Label, Eigener Text, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks: Keine

61.52 Tree

Zeigt auswählbare Elemente in Baumdarstellung an. Dies dient zur Kategorisierung der Elemente.

Web Das HTML-Mapping ist in Abschnitt 51.1.4(1099) beschrieben.

Art: Komponente

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Alle sichtbaren Knoten	Alle sichtbaren Knoten eines Baumes	items	Alle
Alle sichtbaren Knoten mit Selektion	Alle sichtbaren Knoten eines Baumes inklusive deren Selektionstatus	items_with_selection	Alle
Alle Knoten mit Einrückung	Alle Knoten eines Baumes inklusive deren Einrückung	nested_nodes	Alle
Alle sichtbaren Knoten mit Einrückung	Alle sichtbaren Knoten eines Baumes inklusive deren Einrückung	visible_nested_nodes	Alle

Tabelle 61.39: Checktypen für Tree

61.53 TreeNode

Ein Element eines Baumes.

Art: Element oder Syntax SmartID: Wird als Unterelement des Baums über Syntax referenziert

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Keines

qfs:type: Keiner

Zusätzliche Checks:

Name im Popup	Beschreibung	Name des Checktypes	Engine
Knoten	Text des Knotens	item	Alle
Knoten sichtbar	Prüfung, ob Knoten existiert	item_visible	Alle
Selektion des Knotens	Prüfung, ob Knoten selektiert ist	item_selected	Alle
Ausgewählt-Status des Knotens	Prüfung, ob Knoten ausgewählt ist	item_checked	Alle
Alle Unterknoten mit Einrückung	Alle Kindknoten des Knotens inklusive deren Einrückung	nested_nodes	Alle
Alle sichtbaren Unterknoten mit Einrückung	Alle sichtbaren Kindknoten des Knotens inklusive deren Einrückung	visible_nested_nodes	Alle
Abbild des Knotens	Das Abbild des Knotens	item_image	Alle

Tabelle 61.40: Checktypen für TreeNode

61.54 TreeTable

Zeigt auswählbare Elemente in Baumdarstellung. Dies dient zur Kategorisierung der Elemente. Elemente werden typischerweise in mehreren Spalten dargestellt.

Web

Das HTML-Mapping ist in Abschnitt 51.1.5(1101) beschrieben.

Art: Komponente

Koordinaten für Mausklick: Unterelemente bzw. genaue Koordinaten

Merkmal: Keines; für Web-Komponenten siehe Merkmal bei Web-Komponenten(72)

qfs:label*: Zugeordnetes Label, Label in der Nähe, Tooltip

qfs:type: Keiner

Zusätzliche Checks:

Es sind alle Checks von Table und Tree möglich.

61.55 Window

Stellt ein Fenster dar.

Art: Komponente

Koordinaten für Mausklick: Genaue Koordinaten

Merkmal: Titel, falls vorhanden; für Web-Komponenten siehe Merkmal bei Web-Komponenten[72]

qfs:label*: Titel

qfs:type:

qfs:type	Beschreibung
Window:Dialog	Ein unabhängiger Dialog für Eingaben bzw. Bestätigungen
Window:EmbeddedWindow	Eingebettetes Fenster einer externen Anwendung
Window:InternalWindow	Eingebettetes Fenster eines Bereiches der eigenen Anwendung
Window:Notification	Hinweisfenster

Tabelle 61.41: Spezielle qfs:type Typen für Window

Zusätzliche Checks: Keine

Kapitel 62

Doctags

Neben den Eigenschaften der unterschiedlichen Knoten bietet QF-Test noch weitere Möglichkeiten an, um das Verhalten bei der Testausführung bzw. die Darstellung der Knoten im Report zu beeinflussen. Im Attribut Bemerkung von Knoten können Doctags, jeweils in einer eigenen Zeile und in der Form `@teststep` oder `@noreport node` eingetragen werden. Diese müssen nach der allgemeinen Beschreibung des Knotens stehen.

62.1 Doctags für Reporting und Dokumentation

Doctags, die für die Formatierung der Testdokumentation von Testfallsätzen und Testfällen genutzt werden können, sind in Abschnitt 24.2(336) beschrieben, für die Formatierung der Dokumentation für Packages und Prozeduren in Abschnitt 24.3(337).

Die nachstehend beschriebenen Doctags beeinflussen die Darstellung der Knoten im Report.

Doctag	Knoten	Beschreibung
@teststep [name]	Alle Knoten	Ist dieses Doctag gesetzt, so wird dieser Knoten als Testschritt im Report behandelt. Es kann ein optionaler Name angegeben werden.
@report	Check Knoten, Sequenz mit Zeitlimit, Server-HTTP-Request Knoten	Ist dieses Doctag gesetzt, so wird dieser Knoten auf jeden Fall im Report als Check aufgeführt.
@noreport [type],[errorlevel]	Alle Sequenzknoten wie Testfallsatz,Testfall oder Testschritt sowie alle Check-Knoten, Sequenz mit Zeitlimit, Request Knoten und Prozeduraufruf-Knoten	Ist dieses Doctag gesetzt, so wird dieser Knoten nicht im HTML-Report aufgeführt. Details zur Konfiguration, siehe Abschnitt 62.1.1[1361].
@link [filePath/url]	Alle Knoten	Dieses Doctag kann - bei Bedarf mehrfach - verwendet werden, um eine externe Datei oder Ressource zu verknüpfen. Diese kann dann per Rechtsklick und Auswahl des entsprechenden Eintrags im Popup-Menü entweder mit dem passenden Standardprogramm geöffnet, oder im Datei-Manager angezeigt werden (Verknüpfung öffnen bzw. Datei im Explorer anzeigen). Dateien werden relativ zur aktuellen Testsuite aufgelöst.

Tabelle 62.1: Doctags für Report und Dokumentation

62.1.1 @noreport Doctag

4.2+

Das @noreport Doctag kann bestimmte Knoten aus dem Report herausfiltern. Um die Filterung genauer zu spezifizieren, können Sie auch die zwei optionalen Parameter `type` und `errorlevel` konfigurieren. Syntaxbeispiel: `@noreport tree;errorlevel<=WARNING`.

type

Hier können Sie entweder 'tree' oder 'node' angeben. 'tree' ist hier auch die Standardbelegung, falls nichts angegeben wurde. Die Angabe von 'tree' filtert diesen Knoten und alle Kindknoten aus der Reporterstellung. 'node' filtert nur

diesen einen Knoten aus dem Report heraus, die Kindknoten werden aber wieder im Report dargestellt.

errorlevel

Dieser Parameter ist nur bei Sequenzknoten wie Testfallsatz, Testfall oder Testschritt aktiv. Mit diesem Parameter ist es möglich, Knoten nur zu filtern, wenn eine Fehlerstufe erreicht wurde. Hier können Sie die Fehlerstufen `EXCEPTION`, `ERROR`, `WARNING` oder `MESSAGE` mit >,<,<= oder >= angeben. Mit `errorlevel<ERROR` wird der Knoten nur gefiltert, wenn kein Fehler und keine Exception aufgetreten ist, dies ist auch die Standardbelegung. `errorlevel>=MESSAGE` wird der Knoten immer gefiltert, egal, welches Ergebnis. Das ist natürlich sehr gefährlich und sollte nur bei sehr guten Gründen verwendet werden.

62.2 Doctags für Robot Framework

Die folgenden Doctags dienen dazu, Prozeduren(672) oder ganze Packages(680) von QF-Test für die Nutzung in Robot Framework auszuzeichnen. Weitere Informationen zur Integration mit Robot Framework finden Sie in Kapitel 30(410).

Doctag	Knoten	Beschreibung
@keyword [name]	Prozeduren	Bei Verwendung in einem Prozedur Knoten wird der Name der Prozedur - oder der optional nach dem Doctag angegebene Name - als Keyword an Robot Framework übergeben. Es genügt dabei, eine der von Robot Framework unterstützten Formen anzugeben, die anderen konvertiert Robot Framework selbst. So kann z.B. eine Prozedur namens `doClick` automatisch auch über die Keywords ”Do Click” oder ”do_click” aufgerufen werden. Das Doctag kann mehrfach angegeben werden, um die Prozedur für mehrere Keywords zu nutzen. Der implizite Parameter ”__keyword” gibt dabei immer das aufgerufene Keyword in der von QF-Test spezifizierten Form an.
@keyword	Packages	In Package Knoten dient das @keyword Doctag ohne Argument dazu, alle direkt oder indirekt enthaltenen Prozeduren - basierend auf ihrem Namen - als Keywords auszuzeichnen.
@tag [name]	Prozeduren	Mit dem Doctag @tag in Prozedur Knoten angegebene Namen werden an Robot Framework als Tags durchgereicht.

Tabelle 62.2: Doctags für die Robot Framework Integration

62.3 Doctags für die Ausführung

Dieses Doctag beeinflusst das Verhalten von Knoten während der Testausführung.

Doctag	Knoten	Beschreibung
@scope [QF-Test ID einer Komponente|SmartID]	Alle Knoten	Beim Betreten des Knotens wird die angegebene Komponente als Geltungsbereich definiert und wirkt bis zum Verlassen des Knotens auf die Erkennung von Komponenten über SmartIDs. Weitere Informationen finden Sie in Abschnitt 5.7(90).
@rerun [parameters]	Alle Knoten	Hiermit wird die sofortige Neuausführung von Knoten im Fehlerfall konfiguriert. Details, siehe Abschnitt 25.3.2(355).
@outputFilter keep [regexp]	Alle SUT-Client Starter Knoten, auch mehrfach	Nur die Zeilen in der Ausgabe des vom Knoten gestarteten Prozesses, welche den angegebenen regulären Ausdruck enthalten, werden im QF-Test Terminal angezeigt.
@outputFilter drop [regexp]	Alle SUT-Client Starter Knoten, auch mehrfach	Zeilen in der Ausgabe des vom Knoten gestarteten Prozesses, welche den angegebenen regulären Ausdruck enthalten, werden nicht im QF-Test Terminal angezeigt.
@dontcompactify	Alle Knoten	Bezeichnet den Knoten als relevant für das Protokoll, so dass er bei der Kompaktifizierung nicht gelöscht wird (siehe Option Kompakte Protokolle erstellen(590)).
@option [Name der Option] [Wert]	Alle Knoten, auch mehrfach	Setzt eine Option für die Dauer der Ausführung des entsprechenden Knotens auf den angegebenen Wert. Dies gilt für QF-Test selbst ebenso wie für alle aktiven SUT-Clients. Wird ein neuer Client gestartet, während eine Option auf diese Weise gesetzt ist, erhält der Client den angegebenen Wert der Option als Defaultwert und behält diesen, auch wenn die Ausführung den Knoten verlässt und die Option zurückgesetzt wird.

Tabelle 62.3: Doctags für die Ausführung

62.4 Doctags für das Editieren

Diese Doctags beeinflussen das Verhalten von QF-Test während des Editierens.

Doctag	Knoten	Beschreibung
@blue	Alle Knoten	Beim nächsten Laden der Testsuite wird dieser Knoten mit einer blauen Marke versehen.
@breakpoint	Alle Knoten	Setzt einen Breakpoint auf diesen Knoten beim nächsten Laden der Testsuite.
@green	Alle Knoten	Beim nächsten Laden der Testsuite wird dieser Knoten mit einer grünen Marke versehen.
@red	Alle Knoten	Beim nächsten Laden der Testsuite wird dieser Knoten mit einer roten Marke versehen.
@yellow	Alle Knoten	Beim nächsten Laden der Testsuite wird dieser Knoten mit einer gelben Marke versehen.

Tabelle 62.4: Doctags für das Editieren

62.5 Doctags für den Prozedurgenerator

Diese Doctags werden ausschließlich in der Definitionsdatei des Prozedurgenerators verwendet und beeinflussen dessen Wirkung. Eine detaillierte Beschreibung finden Sie in Kapitel 56(1297). Allgemeine Informationen zum Prozedurgenerator finden Sie in Kapitel 27(368).

Anhang A

FAQ - Häufig gestellte Fragen

Evaluation und Lizenzen

1. Kann ich eine Testversion herunterladen?

Ja, Sie finden diese unter www.qftest.com/qf-test/download.html.

2. Brauche ich sonst noch etwas?

Normalerweise wird eine Lizenzdatei benötigt, um QF-Test zu starten. Das Tool läuft auch ohne diese Datei, allerdings können Sie dann keine Dateien speichern und nur die mit QF-Test ausgelieferten Dateien lesen. Das genügt, um das Tutorial durchzuarbeiten und um erste Versuche zu unternehmen und Ihre Applikation unter QF-Test zu starten. Für alles Weitere benötigen Sie eine Lizenzdatei.

3. Und woher bekomme ich diese Lizenzdatei?

Sie können eine 4 Wochen gültige Testlizenz mit dem Formular unter www.qftest.com/qf-test/gratis-testen.html beantragen.

4. Wie viel kostet QF-Test?

Lizenztypen und -preise für QF-Test sind detailliert unter www.qftest.com/qf-test/preise.html aufgeführt.

5. Braucht man für QF-Test einen zusätzlichen Lizenz-Server?

Nein, nicht unbedingt. QF-Test verwaltet Lizenzen für den Multi-User-Betrieb in lokalen Netzen selbst, sofern IP Multicast funktioniert. Um Lizenzen über mehrere Standorte zu verteilen, oder für restriktive Netzwerke, ist ein spezieller Lizenzserver verfügbar. Der Server selbst ist kostenlos und Server-Lizenzen sind sehr preisgünstig. Bei Interesse wenden Sie sich bitte an QFS via <sales@qftest.com>.

Support, Schulung und Feedback

6. Wo bekomme ich Hilfe bei Problemen?

- Bevor Sie Fragen stellen, lesen Sie bitte diesen FAQ oder die allgemeinen FAQ https://www.qftest.com/qf-test/faq.html zu Ende durch. Vielleicht ist die Frage damit bereits beantwortet :-).
- Für den Einstieg gibt es das learning-by-doing Tutorial (u.a. Kapitel 10.6 Hilfe https://www.qftest.com/qf-test-tutorial/lc/tutorial-de-firsthelpweb.html, die Los geht's Anleitung https://www.qftest.com/los-gehts-mit-qf-test.html und bei tiefergehenden Fragen hilft das Handbuch.
- Webseminare für Anfänger jeden Montag um 16 Uhr (MEZ). Anmeldung über service@qftest.com.
- Für Anfänger wie Fortgeschrittene finden sich Themen in unseren Videos https://www.qftest.com/los-gehts-mit-qf-test/videos.html
- Es gibt auch einen Blog unter https://www.qftest.com/blog.html rund um QF-Test, welcher viele hilfreiche Beiträge enthält (Volltextsuche möglich).
- Während der Evaluation von QF-Test gewähren wir freien Support. Wählen Sie dafür in QF-Test im Hilfe-Menü "Support-Team kontaktieren" oder benutzen Sie das Online-Formular.
- Diese Supportmöglichkeiten bieten wir auch für Kunden nach Form unseres Softwarepflegevertrages (näheres unter www.qftest.com/qf-test-support/produktsupport.html).

7. Gibt es Schulungen für QF-Test?

Regelmäßig finden in deutscher und englischer Sprache QF-Test Schulungen für Anfänger wie Fortgeschrittene bei QFS statt. Auch besteht die Möglichkeit von Beratung und Schulung vor Ort oder per Webinar. Details finden Sie unter www.qftest.com/qf-test-support/schulung-beratung.html.

8. Wo kann ich Erweiterungen von QF-Test vorschlagen?

Erweiterungswünsche sind jederzeit an die Adresse support@qftest.com willkommen.

9. Wie berichte ich über einen Bug in QF-Test?

Bitte senden Sie uns eine Problembeschreibung über www.qftest.com/hilfe und wir sehen uns die Sache an. Bitte legen Sie so viele Informationen wie möglich bei, vor allen Dingen Testsuiten und Protokolle.

Ausführen von Tests

10. Warum schlagen heute Tests fehl, die gestern OK waren, obwohl sich nichts geändert hat?

Als ersten Schritt sollten Sie bitte sicher stellen, dass sich wirklich nichts geändert hat. Java oder Browser Updates finden gerne mal im Hintergrund statt, ohne dass man bewusst etwas mitbekommt.

Unabhängig davon kann es vorkommen, dass Tests nur vereinzelt fehlschlagen, ohne offensichtlichen Grund. Das mag nach einem Fehler in QF-Test klingen, ist es aber nur sehr selten. In komplexen multithreaded Umgebungen hängen viele Aktionen und Interaktionen vom Timing ab. Als erste Maßnahme sollten Sie in so einem Fall harte Verzögerungen an kritischen Punkten einbauen. Wenn das hilft, können Sie sich darauf konzentrieren, den Zeitverlust zu minimieren, indem Sie Check Knoten mit Wartezeit oder Warten auf Komponente Knoten verwenden, um auf einen bestimmten Zustand zu warten.

Wenn Verzögerungen nicht helfen, müssen Sie tiefer graben und versuchen, zu verstehen, was passiert. Es ist nicht unwahrscheinlich, das die Ursache ein Bug in Ihrer Anwendung ist - typischerweise ein kniffliger, der sich nur gelegentlich zeigt, abhängig vom Timing oder anderen Umständen. Die groben, offensichtlichen Fehler werden üblicherweise viel früher gefunden - diese kniffligen Bugs sind eines der Themen um die es beim Testen überhaupt geht. Die detaillierten Logs und Bildschirmabbilder von QF-Test helfen bei der Analyse. Unser Support kann Sie dabei unterstützen, die Daten zu interpretieren und relevante Informationen herauszusuchen, die Sie an die Entwickler weitergeben können.

11. Wie führe ich einen Test automatisch aus, z.B. aus der Kommandozeile, einem Testmanagementtool oder einem Skript?

Sie können QF-Test mit Hilfe des Kommandozeilenarguments `-batch`(976) im Batchmodus starten. Darin werden Tests automatisch ausgeführt und ein Protokoll erstellt. Diverse weitere Kommandozeilenargumente beeinflussen den Ablauf der Tests, ihr Ergebnis spiegelt sich im Rückgabewert von QF-Test wieder. Näheres hierzu finden Sie in Testausführung(340), Kommandozeilenargumente(976), Rückgabewerte von QF-Test(996) und Anbindung an Testmanagementtools(373) .

12. Ist es möglich, zwei Applikationen gleichzeitig mit QF-Test zu testen?

Ja, Sie müssen dazu lediglich zwei SUT-Clients mit verschiedenen Namen starten. Sie können dann beide von QF-Test aus kontrollieren.

13. Mein Test läuft über lange Zeit und QF-Test geht der Speicher aus. Wie kann ich das verhindern?

Um den verfügbaren Speicher zu vergrößern, starten Sie QF-Test mit dem Argument -J-Xmx1280m (oder einem noch größeren Wert; QF-Test verwendet normalerweise bis zu 1024 MB). Unter Windows können Sie alternativ das Werkzeug ”QF-Test Java Konfiguration” aus dem Windows Startmenü verwenden. Unter Linux hilft auch ein erneutes Ausführen des Installationsskipts (setup.sh) bei der Anpassung des Speichers. Natürlich hängt der insgesamt verfügbare Speicher von ihrem Rechner ab und wieviel maximal für Anwendungen bereitgestellt wird. Weitere Details finden Sie in Kapitel 1(2).

Es gibt einige Möglichkeiten, den Speicherverbrauch von QF-Test zu reduzieren:

- Setzen Sie die Option Kompakte Protokolle erstellen(590) um alle irrelevanten Teile aus Protokollen zu entfernen.
- QF-Test ermöglicht über das Wiedergabe Menü den Zugriff auf eine Anzahl von Protokollen. Lassen sie die Option Protokolle automatisch speichern(580) eingeschaltet, so dass QF-Test diese Protokolle in Dateien schreiben und aus dem Speicher entfernen kann.
- Schließen Sie die Fenster von Protokollen, die Sie nicht mehr benötigen, so dass der Speicher für diese Protokolle freigegeben werden kann.
- Für lang laufende Tests sind geteilte Protokolle die beste Option, die QF-Test auch standardmäßig nutzt. Damit kann QF-Test Teile eines Protokolls in Dateien schreiben, anstatt das gesamte Protokoll während eines Testlaufs im Speicher zu halten. Näheres hierzu finden Sie in Abschnitt 7.1.6(143).
- Ist die Option Protokoll komplett unterdrücken(590) gesetzt, wird gar kein Protokoll erzeugt. Diese Option ist mit Vorsicht zu genießen. Ohne Protokoll kann es schwierig sein, einen Fehler nachzuvollziehen. Verwenden Sie besser geteilte Protokolle.
- Wenn das SUT viel Text ausgibt, kann es sinnvoll sein, die Anzahl der Clients, die aufgehoben werden, über die Option Wie viele beendete Clients im Menü(536) zu reduzieren.

14. Harte Mausevents und Drag'n'Drop scheitern bei der Wiedergabe, Komponenten werden nicht gefunden, Bildschirmabbilder sind schwarz oder entstellt. Was muss ich bei der Testausführung beachten?

GUI-Tests benötigen einen ungesperrten, aktiven Desktop mit einer aktiven Benutzer-Session. Nur so kann sicher gestellt werden, dass das SUT sich genauso verhält wie bei einem normalen Benutzer.

Sie müssen daher sicherstellen, dass die Testumgebung diese Voraussetzungen erfüllt. Insbesondere bei Continuous-Integration- und Build-Tools wie Jenkins (vgl. Kapitel 29(397)) muss darauf geachtet werden, grundlegende Einstellungen korrekt durchzuführen. Es kann sonst zu ungewöhnlichen Problemen bei der Testausführung(340) kommen, wie zum Beispiel schwarze Bildschirmabbilder im Protokoll (vgl. Abschnitt 7.1(138)), nicht funktionierende Drag'n'Drop Operationen, fehlschlagende harte Events oder sogar Probleme bei der Komponentenerkennung (vgl. Kapitel 5(47)). Java-WebStart-Anwendungen starten unter Umständen überhaupt nicht. Unter Aufsetzen von Testsystemen(474) finden Sie nützliche Tipps und Tricks für die Einrichtung Ihrer Testsysteme.

Folgendes müssen Sie für die Wiedergabe beachten:

- QF-Test und das SUT müssen in einer aktiven, nicht gesperrten Benutzer-Session laufen.
- Die Ausführung der Tests darf unter Windows nicht in der Service-Session oder gänzlich ohne Benutzer-Session ausgeführt werden.
- Bei der Testausführung[340] mit Jenkins muss sichergestellt sein, dass der Jenkins Windows Node nicht als Dienst gestartet wird, sondern entweder mittels Autostart oder über den Windows Aufgabenplaner innerhalb einer Benutzer-Session, keinesfalls in der Service-Session. Stellen Sie hierfür sicher, dass Sie in den 'Sicherheitsoptionen' einen Benutzer ausgewählt haben und die Einstellung 'Mit den höchsten Privilegien ausführen' nicht aktiviert ist. Dieser Benutzer muss während der gesamten Testausführung eingeloggt sein, zum Beispiel mit automatischem Login, und sein Desktop darf nicht gesperrt werden.
- RDP-Verbindungen dürfen nicht minimiert oder gar geschlossen werden, das kommt einer Sperre der Benutzer-Session gleich. Anstelle von RDP sollten Sie VNC, Teamviewer oder ähnliches verwenden. Auch darf RDP nicht zum initialen Login und somit Start einer Session verwendet werden.

Hinweis

Ab Windows 10 bzw. Windows Server 2016 können Sie unter gewissen Voraussetzungen RDP verwenden, wenn Sie folgende Änderung in der Registry vornehmen. Unter `HKEY_CURRENT_USER\Software\Microsoft\Terminal Server Client` oder `HKEY_LOCAL_MACHINE\Software\Microsoft\Terminal Server Client` fügen Sie bitte einen neuen `DWORD` namens `RemoteDesktop_SuppressWhenMinimized` ein. Dort setzen Sie den Wert `2`. Das erlaubt dann das Minimieren des Fensters einer RDP-Verbindung, aber leider immer noch nicht das Schließen oder Trennen. Sie finden Tipps und Tricks zum Aufsetzen von Testsystemen im Handbuch unter Kapitel 39[474] und Kapitel 25[340].

Technische Hintergründe:
Das Stichwort lautet 'session 0 isolation'. Dies bedeutet, jeder Benutzer hat eine eigene Session ID zugewiesen, beginnend ab 1. Die Session mit ID 0 ist für Dienste und Anwendungen ohne konkrete Benutzerzuweisung reserviert und ist stark in ihrer Funktionalität eingeschränkt. Darin laufende Anwendungen werden von anderen Sessions isoliert. In dieser Session können keine Anwendungen mit GUI dargestellt werden und etwaige Versuche, dies doch zu machen, führen dazu, dass die Anwendung nicht dargestellt werden kann und daher nicht problemlos funktioniert. Eine Suche nach 'session 0 isolation' in diversen Suchmaschinen bietet tiefergreifende Informationen zu dieser Tatsache, vor allem für Windows Vista und neuer.
Der unter Windows XP und Windows 2000 funktionierende Workaround mittels tscon.exe und Umleiten der Session 0 ist seit Windows Vista aus Gründen der Sicherheit nicht mehr möglich.

In diesem Zusammenhang möchten wir auf den Einsatz von virtuellen Maschinen hinweisen, insbesondere aus sicherheitspolitischer Hinsicht. Oben genannte Bedingungen der Testausführungen gelten immer nur für die Maschine, auf der der Test tatsächlich ausgeführt wird. Es muss also auf der virtuellen Maschine ein Benutzer eingeloggt und eine nicht gesperrte Session vorhanden sein. Der Host, auf dem die virtuelle Maschine ausgeführt wird, kann natürlich gesperrt sein.

Skripting

15. Wie kann ich auf Objekte meiner Applikation zugreifen, die keine Komponenten sind?

Sie können ein Objekt nicht aus dem Nichts herbeizaubern. Es muss also einen Mechanismus geben, der das gewünschte Objekt über eine statische Klassenmethode zurückliefert. Typische Beispiele hierfür im standard Java-API sind `java.lang.Runtime.getRuntime()` und `java.awt.Toolkit.getDefaultToolkit()`.

16. Schön und gut, aber wie verwende ich das in Jython, Groovy bzw. JavaScript?

Das ist ganz normaler Jython Alltag. Importieren Sie die Klasse und rufen Sie die entsprechende Methode auf, z.B.

```
from java.lang import Runtime
runtime = Runtime.getRuntime()
```

In Groovy wird das Package `java.lang` sogar automatisch importiert:

```
def runtime = Runtime.getRuntime()
```

In JavaScript wird bei Modulen, die nicht importiert werden können, keine Exception geworfen und der Import mit `null` belegt.

```
import {Runtime} from 'java.lang';
runtime = Runtime.getRuntime();
```

Sie können analog auf jede Klasse Ihrer Applikation zugreifen, vorausgesetzt sie ist als `public` deklariert. Beachten Sie, dass Sie dazu einen SUT-Skript Knoten verwenden und keinen Server-Skript Knoten.

17. Wie kann ich in einem Skript auf zusätzliche Java-Klassen zugreifen.

Um weitere Java-Klassen für Jython, Groovy und JavaScript verfügbar zu machen, stellen Sie die entsprechenden jar Dateien in QF-Test's Plugin Verzeichnis (siehe Abschnitt 50.2(1029)).

18. Wie kann ich eine Exception in einem Skript werfen?

Dafür gibt es zwei Möglichkeiten:

- Jython:
  ```
  raise UserException("Beliebige Fehlermeldung")
  ```
  Groovy:
  ```
  import de.qfs.apps.qftest.shared.exceptions.UserException
  ```

```
throw new UserException(”Beliebige Fehlermeldung”)
```
JavaScript:
```
import {UserException} from
’de.qfs.apps.qftest.shared.exceptions’;
throw new UserException(”Beliebige Fehlermeldung”);
```

- `rc.check(Bedingung, ”Meldung”, rc.EXCEPTION)`
  wirft eine Exception wenn die Bedingung nicht erfüllt ist.

19. Welchen externen Editor soll ich verwenden?

Das ist eine Frage persönlichen Geschmacks, die für manche schon an Religion grenzt. Eine umfassende Liste mit Editoren für diverse Betriebssysteme, die Python Syntax unterstützen, finden Sie unter wiki.python.org/moin/PythonEditors. Neben vermutlich sehr vielen anderen Editoren bietet jEdit (www.jedit.org) Syntaxhervorhebung für sowohl Jython, Groovy als auch JavaScript.

Web

20. Woher weiß ich, welches UI Toolkit für meine Web-Anwendung genutzt wird und was tue ich, wenn dieses nicht direkt von QF-Test unterstützt wird?

Wenn möglich, befragen Sie bitte Ihre Entwickler zu den verwendeten Toolkits oder UI Komponenten.
Alternativ aktivieren Sie den automatischen Erkennungsmodus im Schnellstart-Assistenten(33) beim Erstellen Ihrer Startsequenz. QF-Test erkennt dann unterstützte Toolkits von selbst und gibt im Terminal eine entsprechende Meldung aus.
Wenn das Toolkit nicht unterstützt wird oder unbekannt bleibt, kann unser Support einen Blick auf den HTML-Code Ihrer Web-Anwendung werfen. Neue oder kundeneigene Toolkits können mit Hilfe eines CustomWebResolvers mit geringem Aufwand entweder durch Sie selbst (Verbesserte Komponentenerkennung mittels `CustomWebResolver`(1077)) oder unseren Service integriert werden.

21. Warum wird bei einem Datei-Upload/Datei-Download ein zusätzlicher Dialog angezeigt, bevor der eigentliche Dateiauswahl-Dialog erscheint?

Bevor der eigentliche Dateiauswahl-Dialog angezeigt wird, erscheint noch ein kleiner von QF-Test integrierterter Hilfsdialog mit einem OK-/Abbrechen-Button. Dieser Dialog wird von QF-Test benötigt um an die Daten des Dateiauswahl-Dialoges zu kommen, da dieser nativ vom Betriebssystem erzeugt wird. Nach Klicken des OK-Buttons im Hilfsdialog wird der native Dateiauswahl-Dialog angezeigt und Sie können den gewünschten Dateinamen eingeben bzw. die Datei direkt auswählen.

22. Warum führt die Wiedergabe einer bereits aufgenommenen Datei-Upload-/Datei-Download-Sequenz bei Verwendung eines anderen Browsers teilweise zu Fehlern und wie kann ich dies umgehen?

Je nach Implementierung des Datei-Uploads/Datei-Downloads auf der jeweiligen Webseite kann die Wiedergabe etwas knifflig und von Browser zu Browser unterschiedlich sein. Um dies zu umgehen befindet sich in der QF-Test Standardbibliothek `qfs.qft` die Prozedur `qfs.web.input.fileUpload`. Verwenden Sie diese Prozedur anstatt der aufgezeichneten Sequenz, falls auf ihrer Webseite Probleme bei der Wiedergabe auftreten.

23. Ich erhalte im Mozilla Firefox eine Fehlerseite die mich auf ein ungültiges Zertifikat hinweist. Der Standarddialog für das Hinzufügen von Ausnahmen scheint aber nicht zu funktionieren. Wie kann ich dieses Problem lösen?

Dieses Problem tritt ab QF-Test Version 3.5.1 nicht mehr auf, da SSL Zertifikate nun automatisch bestätigt werden. Für ältere Versionen nutzen sie bitte folgenden Workaround.

Um das Zertifikat als vertrauensvoll zu markieren gibt es verschiedene Ansätze:

Variante 1:

- Öffnen Sie im QF-Test Browser-Fenster die URL: `chrome://pippki/content/certManager.xul`.
- Es wird der Zertifikatsmanager geöffnet in dem Sie das Zertifikat der Webseite als vertrauenswürdig definieren können.
- Nach bestätigen mit OK schließt sich das Browser-Fenster.
- Beim nächsten Browserstart ist das Zertifikat der Webseite vertrauenswürdig eingestuft und die URL wird korrekt geladen.

Variante 2:

- Starten Sie den ”normalen” Firefox mit folgenden Kommandozeilenparameter
  `firefox -profile ”[Pfad zu Ihrem Benutzerprofil]/.qftest/mozprofile”`
  `(z.B: firefox -profile ”C:/Users/benutzer1/.qftest/mozprofile”)`
  (Es darf zu diesem Zeitpunkt kein anderes Firefox Fenster geöffnet sein.)
- Rufen Sie die URL auf und bestätigen das Zertifikat der Webseite als vertrauenswürdig.
- Beim nächsten Browserstart ist das Zertifikat der URL als vertrauenswürdig eingestuft und die URL wird korrekt geladen.

24. Ich bekomme OutOfMemoryError für den Browser. Wie erhöhe ich den Speicher für den QF-Test Browser?

Generell wird empfohlen, die Startsequenz mit Hilfe des Schnellstart-Assistenten zu erstellen. In der resultierenden Sequenz im Schritt ”Browser ohne Fenster starten” kann ”Browser starten” Knoten der maximale Speicher in den ”Java-VM” Programmparametern definiert werden, z.B. mittels -Xmx384m, was 384 MB maximaler Speicher bedeutet. Aktueller Standardwert sind 256 MB.

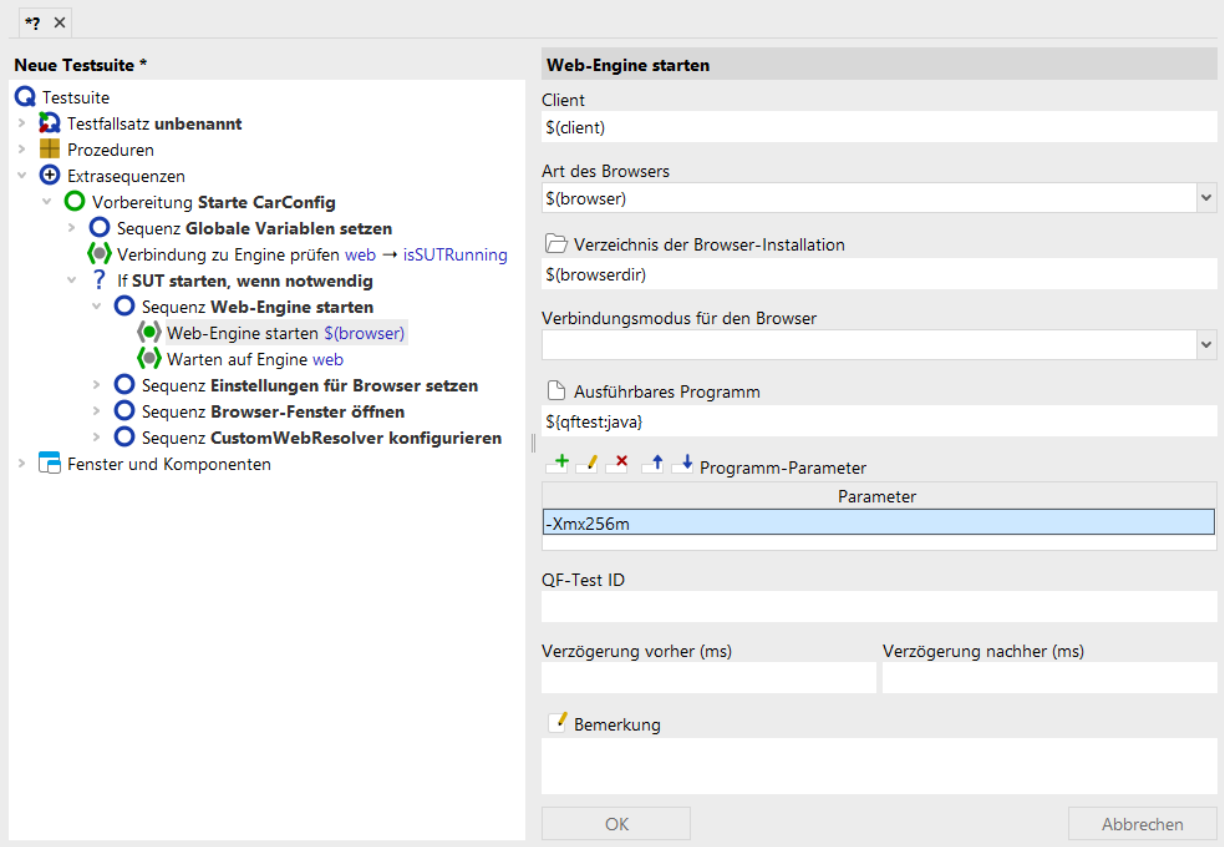


Abbildung A.1: Maximalen Speicher für Browser setzen

25. Die getestete Web-Anwendung öffnet ein Popup-Fenster. Allerdings werden Events nicht auf diesem Fenster sondern auf dem Hauptfenster abgespielt, z.B. wenn ich einen Fensterevent ”WINDOW_CLOSING” abspiele wird manchmal das Hauptfenster anstatt des Popup-Fensters geschlossen oder ein Mausklick klickt ins falsche Fenster. Wie kann ich sicherstellen, dass immer das korrekte Browser-Fenster geschlossen wird ?

Hierfür benötigt QF-Test noch eine zweite Information, nämlich welches Fenster für die Wiedergabe verwendet werden soll. Diese Information setzen Sie beim Warten auf Laden des Dokuments-Knoten wie auch bei der aufgezeichneten Webseite im Attribut Name des Browser-Fensters. Zur aufgezeichneten Webseite kommen Sie am einfachsten mittels Rechtsklick auf den Warten auf Laden des Dokuments-Knoten und Auswahl von

Komponente finden.

Am besten setzen Sie dieses Attribut auf `${default:windowname:}`.

Das Setzen des Attributs Name des Browser-Fensters bewirkt, dass QF-Test zusätzlich zur URL der Webseite auch dieses Attribut zur Erkennung der Webseite inkl. deren Kindkomponenten miteinbezieht. Somit kann QF-Test beide Fenster während der Wiedergabe unterscheiden. Die Angabe dieses Wertes beim Warten auf Laden des Dokuments Knoten setzt diesen Namen aus QF-Test Sicht, diesen können Sie auch in der Titelleiste des Browsers sehen.

Vor dem Abspielen des entsprechenden Warten auf Laden des Dokuments und der Events auf dem Popup-Fenster setzen Sie die Variable `windowname` auf einen beliebigen Wert z.B. auf "popup". Das Setzen dieser Variable kann in einem separaten Variable setzen-Knoten passieren. Bequemer geht es, wenn sie die betroffenen Eventknoten in eine Sequenz einpacken und die Variable `windowname` direkt in der Variablendefinitionen-Tabelle der Sequenz definieren.

Nun können Sie die Events abspielen.

Es ist zu empfehlen die Variable `windowname` nach Ausführung der Eventknoten wieder zurückzusetzen, da QF-Test sonst die nachfolgenden Events auch auf das Popup-Fenster abspielen würde, welches nun aber nicht mehr vorhanden ist. Hierfür genügt ein zweiter Variable setzen-Knoten für `windowname` in dem der Parameter "Defaultwert" leer gelassen wird. Falls die Variable in einer Sequenz definiert wurde, brauchen Sie die Variable nicht explizit zurücksetzen.

Falls Sie bereits bestehende Tests aufgezeichnet haben, können Sie einmal mit globalem Ersetzen das Attribut Name des Browser-Fensters von leer auf `${default:windowname:}` ändern. Hierfür wählen Sie Bearbeiten→Ersetzen, wechseln in das fortgeschrittene Ersetzen, in dem Sie den goldenen Pfeil auf der Toolbar klicken. Nun lassen Sie "Suchen nach" leer, "Ersetzen durch" setzen Sie auf `${default:windowname:}`, bei "Attribut" wählen Sie Name des Browser-Fensters aus und haken die Checkbox "Gesamtes Attribut vergleichen" an.

Der obige Ersetzvorgang muss in allen Testsuiten vorgenommen werden, die Events bzw. Komponenten für dieses Popupfenster beinhalten.

Anhang B

Release Notes

B.1 QF-Test Version 9.0

B.1.1 Änderungen mit möglichen Auswirkungen auf die Testausführung

Web

- Nachdem sie bereits in QF-Test 8.0 abgekündigt wurde, ist nun die Unterstützung von JavaScript-basierten ”WebResolvern” entfernt worden. Dies betrifft in keiner Weise die Definition und Ausführung von ”CustomWebResolvern” oder über das `resolvers`-Modul installierte Resolver.

Swing

- Die Unterstützung für WebStart in Oracle Java 8 ist seit QF-Test 7.1 abgekündigt und wurde nun entfernt. Dies hat keine Auswirkungen auf OpenWebStart, welches weiter unterstützt wird.

- Mit der aktualisierten HTML-Version des QF-Test Handbuchs und Tutorials, die nun eine schnelle, lokale Volltextsuche beinhaltet, ist der Bedarf für eine PDF-Variante nicht mehr gegeben. Die PDF-Version ist daher abgekündigt und wird in einer kommenden QF-Test Version entfernt werden.

Web

- QF-Test wertet nun standardmäßig das ARIA CSS-Attribut `aria-disabled` aus. Elemente mit dieser Markierung werden nun als deaktiviert angesehen, auch wenn sie tatsächlich bedienbar sind. Dadurch können vorher fehlerfreie Tests nun in eine `DisabledComponentException` laufen. Solche Fälle sind allerdings selten und sollten im Hinblick auf Barrierefreiheit tatsächlich als Fehler in der Anwendung oder dem verwendeten UI-Toolkit angesehen und idealerweise beseitigt werden.

  Eine schnelle Umgehung des Problems ist mit folgendem SUT-Skript möglich:

  ```
  rc.setOption(Options.OPT_WEB_IGNORE_ARIA_DISABLED, true)
  ```

B.1.2 Version 9.0.0 - 20. Februar 2025

Neue Features:

- Mit dieser Version führt QF-Test für Web-Anwendungen das Testen auf Barrierefreiheit ein, um automatisch die Konformität mit WCAG und anderen Standards zu prüfen. QF-Test integriert hierfür die bewährte axe-core-Bibliothek, geht aber mit eigenen Features wie der Prüfung von Farbkontrasten in Grafiken darüber hinaus. Ein besonderer Fokus liegt auf aussagekräftigen HTML-Reports mit Übersichts- und Einzel-Screenshots zu den aufgetretenen Fehlern. Als Einstieg dienen der Schnellstart-Assistent, eine Beispiel-Testsuite sowie die Prozeduren im Package `qfs.accessibility.web` in der Standardbibliothek `qfs.qft`.

- Variablen in QF-Test sind nun nicht mehr auf Zeichenketten beschränkt, sondern es können beliebige Objekte zugewiesen werden. Auf Objekt-Werte als solche kann nur in Skripten über die neuen `rc.getObj(...)` Methoden zugegriffen werden, die $-Expansion konvertiert die Werte bei der finalen Expansion in Strings. Beim Weiterreichen von Variablen, z.B. `client = $(client)`, wird deren Typ beibehalten. Einige Prozeduren in der Standardbibliothek `qfs.qft` geben nun Objekte statt Strings zurück mit einer Variablen vom Typ `List<String>` können mehrere Parameter auf einmal in einem Programm starten(731)-Knoten angegeben werden. Alle diese fundamentalen Änderungen sind vollständig rückwärtskompatibel. Weiterführende Informationen finden Sie in Kapitel 6(116), Abschnitt 11.3.3(191) und dem neuen Attribut Expliziter Objekttyp(874) in Variable setzen(871) und Return(678)-Knoten sowie im Blog über Objektvariablen.

- Die HTML-Varianten des QF-Test Handbuchs und Tutorials wurden grundlegend überarbeitet. Neben einem schöneren Layout mit einer Seitenleiste zur schnellen Navigation sowie einem dunklen Modus beinhalten sie nun auch eine schnelle, lokale Volltextsuche.

- Den HTML-Report gibt es nun ebenfalls im dunklen Modus und mit verbesserter Navigation. Miniaturen für die Screenshots im Report werden nun standardmäßig mit einem anderen Algorithmus basierend auf maximaler Breite und Höhe erzeugt (vgl. `-report-scale-thumbnails <Prozent>`(988)).

- Der visuelle UI-Inspektor verfügt nun über ein Suchfeld, mit dem die Knoten in der Baumdarstellung gefiltert werden kann. Er kann nun auch über ein Tastenkürzel geöffnet werden. Dessen Standardbelegung Umschalt-Strg-F11 kann über die Option Hotkey zum Öffnen des UI-Inspektors(575) umdefiniert werden.

- Dateinamen für inkludierte Testsuiten können nun Umgebungsvariablen oder System-Properties referenzieren. Die Syntax dafür lautet ${env:...} bzw. ${system:...}. Sie funktioniert ebenfalls für Einträge in der Option Verzeichnisse mit Testsuite-Bibliotheken(503).

- Über die neue Option Inkludierte Testsuiten in einer neuen Testsuite(504) kann nun festgelegt werden, welche Testsuiten beim Erstellen einer neuen Testsuite automatisch in der Liste Inkludierte Dateien(596) eingetragen werden.
- Das neue Doctag `@option` setzt eine QF-Test Option temporär auf einen angegebenen Wert. Weitere Informationen hierzu finden Sie in Abschnitt 62.3(1363).
- QF-Test kann nun mit der neuen Variablengruppe ”decrypt” verschlüsselte Daten überall dort verwenden, wo eine Variablen-Expansion möglich ist. Um dabei die Vertraulichkeit zu erhalten, werden Expansionen aus dieser Variablengruppe nicht implizit protokolliert.
- Die neue Option Maximale Länge der protokollierten Variablen-Werte(587) beschränkt nun die Größe von Protokoll-Einträgen für das Setzen oder die Expansion von Variablen.
- Aus einem Protokoll können nun Knoten entfernt werden, z.B. um Screenshots oder Nachrichten mit vertraulichem Inhalt vor der Weitergabe zu bereinigen. Aus Gründen der Nachvollziehbarkeit und zum Erhalt der Gesamtzahl an Fehlern werden für die entfernten Knoten Platzhalter angelegt.
- Über die Option Standardalgorithmus für Bildvergleiche(545) kann nun ein Standard-Algorithmus für Bildvergleiche festlegt werden.
- Das neue Package `qfs.pdf.file` in der Standardbibliothek `qfs.qft` enthält Prozeduren zum Umgang mit Anhängen von PDF-Dateien.
- Die spezifischen QF-Test Docker Images sind jetzt sowohl x64 als auch arm64 Architekturen auf Dockerhub verfügbar. Dies ermöglicht eine nahtlose Nutzung auf einer breiteren Palette von Geräten und Plattformen, einschließlich moderner ARM-basierter Systeme wie Apple Silicon.
- Das Ausführen eines -Knotens führt nicht mehr zu einem Fehler. Stattdessen wird analog zur Aktion im Kontextmenü die Komponente im SUT hervorgehoben.
- Die neue Option Klasse oder Typ von Komponenten anzeigen(493) legt fest, ob für einen Komponente(930)-Knoten im Baum nur die Klasse, der spezifischere Typ oder die Kombination aus beidem angezeigt wird.
- Beim Kopieren von Knoten aus einer Testsuite oder einem Run-Log ist die Textvariante in der Zwischenablage nun konsistenter und hilfreicher.

Web

- Im CustomWebResolver installieren(902)-Knoten wurde die YAML-Syntax für Vorfahren-Relationen mittels `ancestor` vereinfacht. Bestehende Konfigurationen bleiben gültig, können aber über ”Reformatieren” in die neue Syntax überführt werden. Außerdem können in einer `genericClasses`-Zuweisung nun `css` und `attribute` kombiniert werden.

Versions-Aktualisierungen:

- Das mit QF-Test für Linux und macOS ausgelieferte JRE wurde zu Temurin OpenJDK Version 17.0.14 aktualisiert.

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.35 bzw. ”2025-03” basieren.

- Jython wurde auf Version 2.7.4 aktualisiert.

Web
- Der eingebettete Chrome Browser für QF-Driver wurde auf CEF Version 131 aktualisiert.

Web
- Der Vaadin-Resolver wurde für Vaadin-Versionen ab 24.6 aktualisiert.

Web
- Die integrierte cdp4j Bibliothek wurde auf Version 7.1.6 aktualisiert.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser-Version 7.42.0 und 8.3.0.

- Die eingebettete JUnit Bibliothek wurde auf Version 5.11.4 aktualisiert.

- Die mit QF-Test gelieferte Bibliothek JSch wurde zu Version 0.2.13 aktualisiert und unterstützt damit rsa-sha2-256 und rsa-sha2-512.

Beseitigte Bugs:

- Der Wert von ${qftest:project.dir} war vereinzelt nicht verfügbar.

JavaFX
- Ein JavaFX-Spinner erhält nun die generische Klasse Spinner und die Buttons zum Hoch- und Runterzählen werden als Button-Komponenten aufgezeichnet.

Mac
- Unter macOS konnte das Zielverzeichnis für einen Report nicht über den Dateiauswahldialog angewählt werden.

- In der aktuellsten Windows-Version konnte QF-Test headless Edge-Browser-Instanzen nicht zuverlässig beenden.

B.2 QF-Test Version 8.0

B.2.1 Version 8.0.2 - 5. Dezember 2024

Neue Features:

- Es werden nun Tests für Anwendungen unterstützt, die auf Java 24 basieren.

- Das mit QF-Test für Linux und macOS ausgelieferte JRE wurde zu Temurin OpenJDK Version 17.0.13 aktualisiert.
- Groovy wurde auf Version 4.0.24 aktualisiert.

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.34 bzw. ”2024-12” basieren.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser-Version 8.2.0.

Web
- Die integrierte cdp4j Bibliothek wurde auf Version 7.1.5 aktualisiert.

Web
- Der Resolver für Vaadin wurde aktualisiert mit verbessertem Mapping von Accordion und Calendar für Versionen ab Vaadin 14 und korrigiertem Mapping von HierarchicalMenu für Versionen ab Vaadin 24.4.

Web
- Der Smart GWT Resolver wurde für Smart GWT Version 13.1p aktualisiert.

iOS
- Der eingebettete Device-Agent für iOS wurde auf WDA Version 8.11.1 aktualisiert.

Beseitigte Bugs:

- Das Laden von CSV-Dateien mittels eines CSV-Datei(664)-Knotens war in den QF-Test Versionen 8.0.0 und 8.0.1 defekt: Die Zeichensequenz ’\t’ wurde versehentlich als Tabulator-Zeichen interpretiert.
- Clients von Kind-Prozessen des SUT konnten unter Umständen den gleichen Namen erhalten.

Web
- Der Vaadin-Resolver konnte für spezielle Trees die Einrückung der Knoten nicht korrekt ermitteln.

B.2.2 Version 8.0.1 - 11. September 2024

Neue Features:

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.33 bzw. ”2024-09” basieren.

Web
- Die integrierte cdp4j-Bibliothek wurde auf Version 7.1.4 aktualisiert.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.41.

iOS
- Der eingebettete Device-Agent für iOS wurde auf WDA Version 8.9.1 aktualisiert.

Beseitigte Bugs:

Windows

- Das mit QF-Test für Windows ausgelieferte JRE wurde auf Temurin OpenJDK Version 17.0.11 zurückgesetzt, da die aktuelle Java-Version einen Fehler im Bezug auf virtuelle Bildschirme enthält, der zu HeadlessExceptions führen kann.
- Der Import von Komponenten konnte zu einer Exception führen, falls eine zusätzliche Ansicht für die Testsuite geöffnet war.

Windows-Tests

- Ein UI-Automation-Element vom Typ SpinButton wird nun als Button aufgezeichnet. Eine frühere Aufzeichnung bleibt gültig.

Mac

- Auf manchen macOS-Systemen wurde die Ausführung von Web-, PDF-, iOS- oder Android-Tests nach einiger Zeit extrem langsam.

Web

- Wenn der angegebene Browser unter macOS nicht gefunden werden konnte, zeigte QF-Test eine irreführende Fehlermeldung an.

B.2.3 Änderungen mit möglichen Auswirkungen auf die Testausführung

- QF-Test selbst benötigt für die Ausführung nun mindestens Java Version 17. Diese ist unabhängig von der Java-Version für das SUT, für das weiterhin die Rückwärtskompatibilität bis Java-Version 8 erhalten bleibt. Eine SUT-Anwendung basierend auf Java Swing, JavaFX oder SWT sollte immer mit ihrem eigenen JRE gestartet werden und nicht mit dem von QF-Test.

  Von Sonderfällen abgesehen, wie z.B. dem Bedarf für ein Plugin, das eine höhere Java-Version benötigt, sollte QF-Test immer mit dem mitinstallierten JRE gestartet werden.

Hinweis

  Das JRE von QF-Test enthält nicht länger die JavaFX Module. Die für die Ausführung der JavaFX Demos von QF-Test benötigten Module werden separat bereitgestellt. enthalten.

- Beim Start ignoriert QF-Test nun die Umgebungsvariable `CLASSPATH`. Bei Bedarf kann stattdessen `QFTEST_CLASSPATH` verwendet werden.
- Die Unterstützung für 32-Bit-Software wurde in QF-Test Version 7.0 abgekündigt und nun komplett entfernt, mit folgenden Ausnahmen:
  - Das Testen von nativen 32-Bit Windows-Anwendungen auf 64-Bit Windows-Systemen mit der QF-Test Windows-Engine bleibt weiterhin unterstützt.
  - Das Testen von Swing oder JavaFX Anwendungen, die in einer 32-Bit Java-VM laufen, funktioniert noch, wird aber nicht mehr offiziell unterstützt.

  Zum Testen von sonstiger 32-Bit-Software verwenden Sie bitte QF-Test Version 7.1.

Web

- Vor QF-Test 8.0 wurde in Web-Anwendungen das ID-Attribut eines DOM-Knotens nur dann als Name für eine QF-Test Komponente verwendet, wenn diese ID im hierarchischen Kontext ”eindeutig genug” war. Dies wurde aus Kompatibilitätsgründen lange beibehalten, obwohl es inzwischen bessere und effizientere Methoden zum Umgang mit nicht eindeutigen Namen gibt.

  Bei der neuen Variante der Option ID-Attribut als Name verwenden[567] ist das Standardverhalten, IDs generell als Namen zu nutzen, unabhängig von Eindeutigkeit. Um die Kompatibilität für Tests mit älteren Komponenten zu erhalten, wird diese Option bei der Migration einer bestehenden Konfigurationen auf den Wert ”Nur wenn eindeutig” gesetzt.

- Die separate Option für die Schriftgröße des Terminals in der QF-Test Workbench wurde durch die allgemeine Option Schriftgröße (pt)[493] ersetzt.

- Die veraltete GNU Regexp Bibliothek wurde zusammen mit der Option ”Alte GNU-Regexps verwenden” aus QF-Test entfernt.

- Die Arbeit ohne Workbench-Ansicht - also mit einem eignen Fenster für jede Testsuite - wurde abgekündigt. Die zugehörige Option wurde vom Menü Ansicht in den Optionen-Dialog verschoben.

Web

- Die Unterstützung von Javascript-basierten ”WebResolvern” wurde abgekündigt. Dies betrifft in keiner Weise die Definition und Ausführung von ”CustomWebResolvern” oder über das `resolvers`-Modul installierte Resolver.

Web

- Das Timeout für die meisten Anfragen im Verbindungsmodus CDP-Driver wurde von 10 auf 3 Sekunden herabgesetzt und entspricht damit nun dem Timeout des Verbindungsmodus WebDriver.

B.2.4 Version 8.0.0 - 8. August 2024

Neue Features:

iOS

- Mit der neuen iOS-Engine können nun iOS-Anwendungen im Simulator unter macOS oder auf echten Geräten, die an einen Rechner mit macOS angeschlossen sind, getestet werden. Nähere Informationen finden Sie in Kapitel 17[269].

JavaFX

- Die Unterstützung für JPro wurde stark verbessert und auf die JPro Version 2024.3 aktualisiert. JPro bringt JavaFX-Anwendungen in den Browser und QF-Test kann gleichzeitig so mit beiden Technologien interagieren, dass für JavaFX geschriebene Tests fast unverändert mit JPro und dem Browser ausgeführt werden können. Erklärungen zum Konzept finden Sie in Kapitel 20[307] und eine JPro Demo-Testsuite über den Menüeintrag Hilfe→Beispiel-Testsuiten erkunden..., Eintrag ”JPro JavaFX CarConfig Suite”.

- Das neue ”Solarized” UI-Theme ist ein Standard-Farbschema basierend auf dem Konzept, dieselben Vordergrund-Farben im hellen und dunklen Modus zu verwenden. Es wird von Vielen als angenehm empfunden und kann über das Menü Ansicht→UI-Theme aktiviert werden.
- QF-Test enthält nun eine eigene Assertion-Bibliothek für Skripte, die von Chai.js inspiriert wurde.
- JSON-Objekte können nun durch das neue JSON-Skriptmodul einfacher in Groovy und Jython verwendet und in QF-Test Variablen serialisiert werden.

Mac
- QF-Test wird auf macOS-Systemen mit Apple-Silicon-Prozessor nun als nativer ARM-Prozess ausgeführt, was zu einer deutlichen Beschleunigung führt.
- Das mit QF-Test ausgelieferte JRE wurde zu Temurin OpenJDK Version 17.0.12 aktualisiert.

Web
- Der eingebettete Chrome Browser für QF-Driver wurde auf CEF Version 126 aktualisiert.
- Die Unterstützung für Webswing wurde für die aktuelle Webswing-Version 24.1 aktualisiert.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser-Version 7.40 sowie der kommenden Version 8.
- Groovy wurde auf Version 4.0.22 aktualisiert.

Web
- Die integrierte cdp4j-Bibliothek wurde auf Version 7.1.3 aktualisiert.

Web
- Der in QF-Test integrierte GeckoDriver wurde auf Version 0.35.0 aktualisiert.

Electron
- QF-Test unterstützt nun den neuen Objekt-Typ `BaseWindow` von Electron Version 30.
- Die eingebettete WebP-Bibliothek für die Bildkompression wurde auf Version 1.4.0 aktualisiert.
- Die eingebettete Bibliothek Apache Commons IO wurde auf Version 2.16.1 und Apache Commons CSV auf 1.11.0 aktualisiert.

Web
- Die Option ID-Attribut als Name verwenden(567) hat nun einen dritten Wert. Der neue Standard ist, IDs generell als Namen zu nutzen, unabhängig von deren Eindeutigkeit. Falls Probleme mit der Kompatibilität auftreten, setzen Sie die Option auf den früheren Standardwert ”Nur wenn eindeutig” (vgl. Änderungen mit möglichen Auswirkungen auf die Testausführung(1381)).

Web
- Mappings für Generische Klassen im CustomWebResolver installieren(902)-Knoten können jetzt mehrere alternative HTML-Tagnamen oder CSS-Klassen auf einmal zuweisen.

Web
- Durch Mappen der passenden Komponente auf die neue generische Klasse `ModalOverlay` kann verhindert werden, dass QF-Test ein modales Overlay umgeht und Mausevents auf davon abgedeckte Elemente ausführt.

Web
- Das neue Interface `TreeIndentationResolver` kann implementiert werden, um QF-Test bei der Erkennung der Struktur von Baumknoten in einer Web-Anwendung zu unterstützen. Genauere Informationen hierzu finden Sie in Abschnitt 54.1.24(1186).

- Das Layout des HTML-Reports ist für kleinere Bildschirme und zum Drucken nun besser geeignet.

- Mit dem neuen Kommandozeilenargument `-noplugins`(984) kann vorübergehend die Einbindung von Plugins unterdrückt werden, um im Fall eines Problems herauszufinden, ob ein Plugin als Ursache in Frage kommt.

- Die zuletzt verwendeten Dateien wurden in ein Untermenü des Datei-Menüs verschoben und einige Standard-Lesezeichen für wichtige Bibliotheks- und Beispiel-Suiten ergänzt.

Beseitigte Bugs:

- Die Sortierreihenfolge von Attributen in den XML-Dateien für Testsuiten und Protokolle ist nun unabhängig von den Spracheinstellungen des Systems.

- Beim Abgreifen der IO-Streams `System.out` und `System.err` des SUT geht QF-Test nun vorsichtiger vor, um deren implizite Kodierung nicht zu verändern.

Android
- Die apk-Datei, welche QF-Test auf Android-Geräten für den Zugriff auf die Accessibility-Schnittstelle benötigt, wurde für die Kompatibilität mit neueren Android-Versionen aktualisiert und neu signiert.

Web
- Die Standard-Installation von Firefox unter Linux akzeptiert keine Profilverzeichnisse mehr außerhalb von `~/.mozilla`. QF-Test erkennt und umgeht diese Situation nun, indem es ein dediziertes Profilverzeichnis für Firefox-Tests in `~/.mozilla` statt im QF-Test Anwenderverzeichnis erstellt.

Web
- Auf Windows-Systemen werden headless Browser nun immer unskaliert gestartet, unabhängig vom Skalierungsfaktor für die aktuelle Desktop-Session.

Electron
- Die emulierte Web File System API für Electron-Anwendungen kann nun auch auf mehrere Dateien parallel lesend und schreibend zugreifen.

Mac
- In speziellen Fällen konnte QF-Test unter macOS blockieren, wenn eines seiner Anwendungsfenster nach vorne gebracht werden sollte.
- Die Prozedur `qfs.autowin.acrobat.saveAsText` in der Standardbibliothek `qfs.qft` funktioniert nun auch für den englischen Acrobat Reader in Version 24.512 und höher.
- Das QF-Test Gradle-Plugin übergibt die ”license” Property nun korrekt an QF-Test.

B.3 QF-Test Version 7.1

B.3.1 Version 7.1.5 - 16. Juli 2024

Neue Features:

SWT
- Der visuelle Inspektor zur Analyse von UI-Komponenten steht nun auch für Eclipse/SWT-Anwendungen und damit für alle QF-Test UI-Engines zur Verfügung.

Web
- Die integrierte cdp4j Bibliothek wurde auf Version 7.1.2 aktualisiert.

Web
- Der Auswahldialog für eine Suchmaschine in Chrome wird unterdrückt.

Beseitigte Bugs:

- Bei Verwendung eines Unterelements als Scope konnte es zu einer falsch-positiven Erkennung kommen, wenn das Element innerhalb des Scopes nicht existierte.

Web
- Komponenten innerhalb eines FRAME in einem zweiten Browser-Fenster wurden nicht zuverlässig erkannt.

B.3.2 Version 7.1.4 - 12. Juni 2024

Neue Features:

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.32 bzw. ”2024-06” basieren.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.37, 7.38 und 7.39.

Beseitigte Bugs:

- Beim Ausführen des Windows-Installers im Silent-Modus werden nun keine alten QF-Test Versionen mehr deinstalliert.
- Das Speichern von PDF-Dateien mit der Prozedur `qfs.autowin.acrobat.savePDF` war nicht möglich, wenn die Acrobat-Option "Online-Speicher beim Speichern von Dateien anzeigen" deaktiviert war.
- Beim Generieren von testdoc-Dokumentation mit deaktivierter Option für Testschritte werden diese nun auch aus der XML-Variante entfernt statt nur aus HTML.

Web
- Webdriver für neuere Versionen von Microsoft Edge werden nun wieder automatisch heruntergeladen.

Web
- Das automatische Scrollen in Webanwendungen wurde für den Sonderfall verbessert, dass Vorgänger-Komponenten in der Hierarchie unsichtbar sind.

Web
- In Web-Anwendungen mit mehreren Dokumenten konnte der QF-Test UI-Inspektor teilweise keine Detail-Informationen zu Knoten liefern.
- In manchen Fällen konnten node.js-Module nicht in JavaScript-Skripte importiert werden.

B.3.3 Version 7.1.3 - 24. April 2024

Neue Features:

- Es werden nun Tests für Anwendungen unterstützt, die auf Java 23 basieren.
- Das mit QF-Test ausgelieferte JRE wurde zu Temurin OpenJDK Version 17.0.11 aktualisiert.
- Groovy wurde auf Version 4.0.21 aktualisiert.
- Die eingebettete cdp4j-Bibliothek wurde auf Version 7.1.1 aktualisiert.

JavaFX
- Der visuelle Inspektor zur Analyse von UI-Komponenten steht nun auch für JavaFX-Anwendungen zur Verfügung und bringt einige kleinere Verbesserungen für alle unterstützten Engines mit.
- Die spezielle Variablen-Gruppe "qftest" enthält jetzt weitere Werte für Client-Properties (vgl. Abschnitt 6.8(127)).

Web
- Beim Start von Chrome aus QF-Test heraus wird dessen neues Privacy-Banner nun standardmäßig unterdrückt.

Beseitigte Bugs:

- Nach Vergrößern der Option Schriftgröße (pt)(493) wurden Attribute wie Bedingung(694) in If(693)-Knoten nicht mehr korrekt dargestellt.
- Das Doctag @noreport funktioniert nun auch an Prozeduraufruf(675)-Knoten.
- Der Zähler für übersprungene Tests war nicht korrekt, wenn ein Testfall(599) mit erzwungenem Aufräumen der Abhängigkeit aus einem Skript heraus mit `skipTestCase` verlassen wurde.

Web

- Das Einfügen einer JavaScript-Funktion mittels der abgekündigten Prozedur `updateCustomWebResolverProperties` in der Standardbibliothek `qfs.qft` klappte im Firefox nicht mehr.

B.3.4 Version 7.1.2 - 14. März 2024

Neue Features:

SWT

- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.31 bzw. ”2024-03” basieren.

Electron

- Die Erkennung von neuen Fenstern in Electron-Anwendungen wurde verbessert.
- Die interaktiven Terminalfenster für die verschiedenen Skriptsprachen in QF-Test und dem SUT werden nun zur besseren Verständlichkeit und Abgrenzung als Konsolen bezeichnet.

Beseitigte Bugs:

- Der Aufnahme-Button in der Toolbar wird nach einer Änderung der Größe der Toolbar-Icons nun wieder korrekt dargestellt.

B.3.5 Version 7.1.1 - 27. Februar, 2024

Die einzige Änderung in dieser Version ist das Entfernen von drei ausführbaren Dateien, die Teil der eingebetteten cdp4j-Bibliothek sind und die plötzlich von verschiedenen Scannern als bösartig detektiert wurden. Diese Dateien waren seit Version 6.0.4 (November 2022) Teil von QF-Test, wurden aber von QF-Test nie genutzt und sollten harmlos sein. Weitere Informationen folgen, wenn wir mehr über die Hintergründe wissen.

B.3.6 Änderungen mit möglichen Auswirkungen auf die Testausführung

- Die Unterstützung für WebStart in Oracle Java 8 wurde abgekündigt und wird in einer zukünftigen QF-Test Version entfernt. Dies hat keine Auswirkungen auf OpenWebStart.
- QF-Test unterstützt nun das Testen von Applets nicht mehr - diese konnten zuletzt nur noch mit dem Internet Explorer ausgeführt werden, und QF-Test hat mit Version 6.0 die Unterstützung des Internet Explorers abgekündigt und mit Version 7.0 entfernt.
- Der Start von QF-Test mit einem anderen als dem mitgelieferten JRE, insbesondere mit Java 8, ist abgekündigt. Dies hat keinen Einfluss auf die unterstützten Java-Versionen für das SUT.
- Die eingebettete cdp4j-Bibliothek wurde auf Version 7 aktualisiert. Dies beinhaltet einen Namespace-Wechsel der cdp4j-Klassen von `io.webfolder.cdp` nach `com.cdp4j`. Wenn solche Klassen direkt in SUT-Skripten referenziert werden, dann müssen die Import-Befehle entsprechend angepasst werden.

CustomWebResolver

Nach der Ablösung des schwer verdaulichen `qfs.web.ajax.installCustomWebResolver` Aufrufs durch den CustomWebResolver installieren(902) Knoten in QF-Test 7.0 wurde nun auch der darunter liegende Code optimiert und dabei entschlackt, wodurch in speziellen Fällen Anpassungen nötig sein können. Sollte dies bei Ihnen der Fall sein und Sie Hilfe benötigen, wenden Sie sich gerne an unser Support-Team. Konkret sind folgende Punkte betroffen:

Web
- Die Auswertungsreihenfolge der Definitionen in den CustomWebResolver-Kategorien ”genericClasses” und ”redirectClasses” ist nun wohldefiniert: Der erste Treffer gewinnt, basierend primär auf der Reihenfolge der Einträge im Knoten (Details siehe Abschnitt 51.1.2(1082)). Bei bestehenden CWR-Konfigurationen mit überlappenden Einträgen ist es möglich, dass dadurch nun Einträge berücksichtigt werden, die in früheren Versionen keine Wirkung hatten.

Web
- Der CustomWebResolver installieren(902) Knoten unterstützt die veralteten Kategorien ”indirectFeatureClasses”, ”insertClassesFront”, ”textRedirectInFetch” und ”goodClasses” nicht mehr, ebenso alle schon zuvor abgekündigten Kategorien inklusive ”ieHardClasses” und ”ieSemiHardClasses”.

B.3.7 Version 7.1.0 - 20. Februar 2024

Neue Features:

- QF-Test bietet nun auch ein UI-Theme mit hohem Kontrast, sowohl für den hellen als auch dunklen Modus.
- Die Darstellung der Baumknoten in Testsuiten und Protokollen wurde so überarbeitet, dass diese nun schlanker und frischer erscheinen. Details zu den verschiedenen neuen Optionen, über die diese Darstellung umfangreich angepasst werden kann, finden Sie in Darstellung(490).
- Es werden nun Tests für Anwendungen unterstützt, die auf Java 22 basieren.

Web

- Web-Anwendungen basierend auf neueren Versionen des Vaadin-Frameworks (ab Version 14) werden nun out-of-the-box unterstützt. Ebenfalls neu ist Unterstützung zur Aktivierung grundlegender Testfunktionen für das Flutter Web-Framework sowie generische Komponentenerkennung für Web-Anwendungen, welche den WCAG ARIA Richtlinien für Barrierefreiheit folgen.

Swing

- Der visuelle Inspektor zur Analyse von UI-Komponenten steht nun auch für Windows und Swing/AWT-Anwendungen sowie für in Java-Anwendungen eingebettete Web-Views zur Verfügung (vgl. Abschnitt 5.12.2(108)).
- Im UI-Inspektor kann die vorgeschlagene Smart-ID nun einfach kopiert werden.
- Bedingungen in If(693) und anderen Knoten können jetzt auch in anderen Skriptsprachen als Jython angegeben werden. Die Standard-Skriptsprache für die Bedingung von neuen Knoten kann über die Option Voreingestellte Sprache für Bedingungen(485) festgelegt werden.
- Bei den Knoten Fehler(853), Warnung(859) und Nachricht(865) kann nun auf Wunsch die Nachricht zusätzlich im Terminal ausgegeben werden.
- In Knoten, die einen SUT-Client starten, werden leere Argumente nun standardmäßig ignoriert (vgl. Leere Argument-Zeilen beim Start des Clients ignorieren(535)).

Windows

- Das Windows-Installationsprogramm für QF-Test unterstützt nun direkt das Entfernen älterer QF-Test Versionen.
- Das mit QF-Test ausgelieferte JRE wurde zu Temurin OpenJDK Version 17.0.10 aktualisiert.
- Groovy wurde auf Version 4.0.18 aktualisiert.

Web
- Der eingebettete Chrome Browser für QF-Driver wurde auf CEF Version 120 aktualisiert.

Web
- Die eingebettete cdp4j-Bibliothek wurde auf Version 7.0.1 aktualisiert. Dies beinhaltet ein Update der Chrome Devtools Protocol API auf r1245094.

Web
- Der in QF-Test integrierte GeckoDriver wurde auf Version 0.34.0 aktualisiert.

- Die CarConfigurator-Anwendungen, die für Demos und Trainings verwendet werden, haben ein moderneres Aussehen und aktualisierte Daten erhalten. Die Demo-Testsuiten für diese Anwendungen enthalten nun weitere Beispiele zur Verwendung von SmartIDs.

Web
- Für den Mobile Emulation Mode wurden die Spezifikationen neuerer Geräte hinzugefügt.

Web
- Das Attribute Methode(912) im Knoten Server-HTTP-Request(910) unterstützt nun auch variable Werte und im neuen Attribut Zusätzliche Header(912) können Header nun textuell angegeben werden, was auf Skript-Ebene einfacher ist als die Header(912)-Tabelle.

- Das neue Package `qfs.utils.json` in der Standardbibliothek `qfs.qft` stellt Hilfsprozeduren zum Vergleich von JSON-Dateien zur Verfügung.

- Die E-Mail-Prozeduren im Package `qfs.utils.email.pop3` der Standardbibliothek `qfs.qft` unterstützen jetzt auch SSL-verschlüsselte Verbindungen.

Android
- Mit Hilfe des Packages `qfs.autoscreen.android` ist es jetzt möglich, bildbasierte Tests für Android-Anwendungen auszuführen.

- Der Aufnahme-Button in der Toolbar zeigt nun an, ob der SmartID-Aufnahmemodus aktiv ist.

- Mit Hilfe der neuen Klasse `ImageRepDrawer` ist es nun möglich, auf Skript-Ebene einfache Zeichenoperationen auf `ImageRep` Objekten durchzuführen (siehe Abschnitt 54.9.3(1236)).

Web
- Die Flexibilität der CustomWebResolver-Konfiguration wurde in den Kategorien ”redirectClasses”, ”abstractCoordinatesClasses”, ”ignoreTags” und ”browserHardClickClasses” erhöht. So ist es zum Beispiel häufiger möglich, reguläre Ausdrücke in der Beschreibung zu verwenden.

Web
- Aufrufe der Prozedur `qfs.web.ajax.updateCustomWebResolverProperties` können nun ebenfalls in CustomWebResolver installieren(902) Knoten konvertiert werden.

Web
- Es ist nun möglich, eine Prozedur mit einem Aufruf von `installCustomWebResolver` als ”base” einer Konfiguration in einem CustomWebResolver installieren(902) Knoten zu verwenden.

Web
- Die QF-Test Pseudo-DOM-API wurde um die `callJS`-Methode erweitert, um im Web-Dokumenten-Kontext JavaScript-Code ausführen zu können, ohne dass implizit `window.eval()` aufgerufen wird.

Beseitigte Bugs:

- Beim Abspielen von Events werden die Timeouts der beiden Optionen Warten auf nicht vorhandene Komponente (ms)(555) und Warten auf nicht vorhandenes Element (ms)(555) nun individuell berücksichtigt und nicht nur als einfacher Summenwert.
- Die Methode `ImageWrapper.grabImage` kann nun auch eine ’QF-Test Komponenten-ID’ als String-Parameter erhalten.
- Nachdem ein Tabulator-Zeichen aus der Zwischenablage in ein Skript eingefügt wurde, konnte es zu einer Exception beim anschließenden Bewegen des Cursors kommen.

JavaFX
- Ein modales JavaFX Fenster wird nun mit dem Klassennamen ”Dialog” aufgezeichnet.

Web
- Mit dem Komponentenevent(790) Knoten können nun MOVED und SIZED Events an die HTML Komponente einer Webseite gesandt werden, um die Position und Größe des dargestellten Inhalts festzulegen.

Web
- Bei der Komponentenaufnahme in Web-Anwendungen zeichnet QF-Test nun keine unsichtbaren DOM-Knoten mehr auf. Das alte Verhalten kann über rc.setOption(Options.OPT_WEB_RECORD_INVISIBLE_ELEMENTS, true) wiederhergestellt werden.

Web
- Aufgrund von unerwarteten Fehlermeldungen, die der Chrome-Browser vereinzelt bei der Versionserkennung ausgab, konnte der automatische Chromedriver-Download fehlschlagen.

Web
- Reguläre Ausdrücke in der CustomWebResolver-Kategorie ”ignoreTags” werden nun korrekt ausgewertet.

Web
- Die Elementen mit der CSS-Klasse ”visually-hidden” werden jetzt als unsichtbare QF-Test Komponenten evaluiert.

Web
- Unter Umständen konnte QF-Test sich nicht mit dem Edge-Browser verbinden.

Web
- CSS-Styling-Informationen in STYLE Tags konnten fälschlicherweise als Text gewertet werden.

B.4 QF-Test Version 7.0

B.4.1 Version 7.0.8 - 5. Dezember 2023

Neue Features:

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.30 bzw. ”2023-12” basieren.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.36.

Beseitigte Bugs:

- Im Report werden Knoten, die mit einem leeren @teststep Doctag erzeugt wurden, nun mit expandierten Variablen dargestellt.

Mac
- Unter macOS konnte QF-Test beim Start einfrieren, wenn die WebP Bibliothek für Bildkompression nicht verfügbar war.

Web
- In Electron-Anwendungen konnten Klicks auf Elemente in einem Popup-Menü nur dann wiedergegeben werden, wenn zuvor andere Menüelemente angeklickt worden waren.

Web
- Der Microsoft Edge Browser stürzte ab, wenn händisch ein neues, leeres Tab geöffnet wurde.

Android
- Der SUT-Client für Android beendete sich nicht automatisch nach Schließen des Emulators, wenn das Aufnahmefenster zu diesem Zeitpunkt geöffnet war.

B.4.2 Version 7.0.7 - 11. Oktober 2023

Neue Features:

Web
- Der eingebettete Chrome Browser für QF-Driver wurde auf CEF Version 117 aktualisiert, in welcher die WebP Sicherheitslücke beseitigt wurde.

Web
- Die eingebettete Websocket-Bibliothek wurde auf Undertow 2.2.26 aktualisiert.
- Die Skript-Methode `rc.overrideElement` unterstützt nun auch das Überschreiben von verschachtelten SmartIDs. Sie wird nun durch die neue Methode `rc.getOverrideElement` ergänzt. Details hierzu finden Sie in Abschnitt 11.3.7[198] und Abschnitt 50.5[1030].

Beseitigte Bugs:

- Test-Reports, die im Batchmodus mit QF-Test Versionen von 7.0.4 bis 7.0.6 erstellt wurden, zeigten eventuell unsauberes HTML an, wenn sie via Navigation von der Zusammenfassung in einen Einzelreport geöffnet wurden.

B.4.3 Version 7.0.6 - 29. September 2023

Beseitigte Bugs:

- Die in QF-Test eingebettete WebP-Bibliothek für Bildkomprimierung in Protokollen und Testsuiten wurde auf Version 1.3.2 aktualisiert. In dieser Version wurde ein schwerwiegendes Sicherheitsproblem (CVE-2023-4863) behoben.

B.4.4 Version 7.0.5 - 20. September 2023

Neue Features:

- Die Prozeduren `qfs.autowin.acrobat.savePDF` und `qfs.autowin.acrobat.saveAsText` in der Standardbibliothek `qfs.qft` unterstützen nun auch Acrobat Reader in Version 23 und höher.

Web

- QF-Test unterstützt nun auch Tests mit JxBrowser 7.35.

Web

- QF-Test unterstützt nun auch das Aufnehmen und Abspielen von Klicks auf Popup-Menüs in Electron-Anwendungen.

Beseitigte Bugs:

- QF-Test im interaktiven Modus startete nicht, wenn ein Plugin eine inkompatible Version der `org.w3c.css.sac` Klassen enthielt.

Web

- QF-Test verwendet nun den neuen Headless-Modus von Chrome auch unter Linux. Ohne diesen startete Chrome 117 und höher im Headless-Modus nicht mehr.

B.4.5 Version 7.0.4 - 30. August 2023

Neue Features:

- Das mit QF-Test ausgelieferte JRE wurde zu Temurin OpenJDK Version 17.0.8.1_1 aktualisiert.

SWT

- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.29 bzw. ”2023-09” basieren.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.34.

Mac
- Auf Apple Silicon Geräten werden Browser im Verbindungsmodus CDP-Driver nun nativ für ARM gestartet, was die Geschwindigkeit deutlich verbessert.
- Die meisten Toolbar-Buttons haben nun einen ”Was ist das?”-Eintrag in ihrem Kontextmenü, der zur jeweils relevanten Dokumentation im Handbuch führt.

Beseitigte Bugs:

- Die `jackson.jar` Bibliothek für YAML und JSON Parser erzeugt nun keine Konflikte mehr, wenn eine andere Jackson-Bibliothek in das QF-Test Plugin-Verzeichnis gelegt wird.
- Das Hervorheben der Scope-Komponente über das Kontextmenü für den Knoten oder das @scope Doctag funktioniert nun wieder korrekt.

Web
- QF-Test kann nun wieder aktuelle ChromeDriver und WebDriver Versionen für Google Chrome und Microsoft Edge automatisch herunterladen. In beiden Fällen hatte sich die entsprechende URL bzw. das Layout der Seiten verändert.

Electron
- Beim Start einer Electron-Anwendung wartete QF-Test in seltenen Fällen bis zum Ende des Timeouts, auch wenn die Verbindung schneller hergestellt werden konnte.

B.4.6 Version 7.0.3 - 13. Juli 2023

Beseitigte Bugs:

- Ein Testlauf brach in seltenen Fällen mit einer Exception ab wenn QF-Test beim Schreiben eines geteilten Protokolls auf inkorrekt erstellte leere Abbilder stieß.
- Die Erkennung von Komponenten mit qfs:label im alten Modus funktionierte in einzelnen Fällen nicht korrekt, wenn ein `ExtraFeatureResolver` registriert war.
- Variablen in Doctags von Shell-Kommando ausführen(734) Knoten werden nun korrekt expandiert.

Swing
- In seltenen Fällen konnte es beim Adressieren einer Zeile als Unterelement einer `JTextArea` zu einer `NullPointerException` kommen.

Web
- Bei der Wiedergabe eines Geometrie auslesen Knotens auf ein nicht existierendes Unterelement wurde bei Web-Anwendungen fälschlicherweise die Geometrie der Parent-Komponente zurückgeliefert. Nun wird stattdessen korrekterweise eine `IndexNotFoundException` geworfen.

Android
- Die Prozedur `qfs.android.adbUtils.appPackage.getCurrentPackage` in der Standardbibliothek `qfs.qft` funktioniert nun auch auf Android-Geräten, die kein `grep`-Kommando beinhalten.

B.4.7 Version 7.0.2 - 22. Juni 2023

Neue Features:

- QF-Test unterstützt nun auch Tests mit JxBrowser 7.33.

Beseitigte Bugs:

- Nach einem Start von QF-Test 7.0.1 mit nur einem Protokoll und anschließendem Öffnen einer Testsuite wurde die vorherige Sitzung nicht wiederhergestellt.
- Unter Windows mit skaliertem Bildschirm zeigt der PDF-Client nun ein skaliertes Dokument an, bei dem Rahmen und Check-Hervorhebungen korrekt ausgerichtet sind. Abbild-Checks werden entsprechend der 100% Skalierung erstellt und sollten damit kompatibel zu Checks bleiben, die auf einem unskalierten Bildschirm erstellt wurden.

Web
- Die Installation eines CustomWebResolvers schlug mit einer Exception fehl, wenn dieser fehlerhaften JavaScript-Code enthielt. Nun wird stattdessen ein Fehler protokolliert.

Web
- Der CustomWebResolver installieren(902) Knoten konnte nicht ausgeführt werden, wenn eine jar-Datei im Plugin-Verzeichnis von QF-Test eine Jackson-Bibliothek in einer inkompatiblen Version enthielt.

B.4.8 Version 7.0.1 - 31. Mai 2023

Neue Features:

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.28 bzw. ”2023-06” basieren.

Mac
- Es werden nun auch Clients unterstützt, die auf macOS mit einem ARM-Java ausgeführt werden.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.32.
- QF-Test startet nun spürbar schneller, im interaktiven Modus ebenso wie im Batchmodus.

- Die neue spezielle Variable `${qftest:suite.name}` expandiert zum - in derem Wurzelknoten definierten - Namen der aktuellen Testsuite.

Web
- Neue CWR-Kategorien und Zuweisungen werden nun beim Cursor eingefügt anstatt ganz am Ende der Liste.

- Bei Verwendung von kompakten Protokollen (siehe Option Kompakte Protokolle erstellen(590)) können einzelne Knoten nun über den Doctag @dontcompactify (siehe Kapitel 62(1360)) von der Kompaktifizierung ausgeschlossen werden.

Beseitigte Bugs:

- QF-Test startet nun mit der System Property -Dsun.io.useCanonCaches=true um Canonical Filename Caches für Java-Version 17 zu aktivieren und damit - analog zu älteren Java-Versionen - Performance-Einbußen durch große Projekte auf langsamen Dateisystemen zu vermeiden.

- Einige Prozeduren in der Standardbibliothek qfs.qft schlugen wegen der Nutzung von `${qftest:engine.$(id)}` bei Aufruf mit einer SmartID fehl. SmartIDs ohne explizite Angabe einer Engine nutzen nun die GUI-Engine ”default” (vgl. Kapitel 45(998)).

- Einträge in Konfigurationsdateien waren nicht mehr sortiert, wenn QF-Test mit Java 17 lief.

Web
- Vereinzelt konnte der Checkmodus im Browser noch aktiv bleiben, selbst wenn die Aufnahme gestoppt wurde, wodurch weitere Interaktionen blockiert waren.

Web
- Bei Verwendung einer Webseite als Scope für eine SmartID wurde der Scope bei einer Navigation nicht korrekt aktualisiert.

Web
- Das Löschen von Dateien über die File System Access API konnte in sehr speziellen Fällen zu einer Exception führen.

Web
- Duplizierte Kategorien in einer CWR-Konfiguration werden nicht länger ignoriert, sondern lösen einen Fehler aus.

Swing
- Abbilder von Swing Fenstern werden nun mit höherer Qualität erstellt.

- Unter Windows mit skaliertem Bildschirm zeigt der PDF-Client nun ein skaliertes Dokument an, bei dem Rahmen und Check-Hervorhebungen korrekt ausgerichtet sind. Abbild-Checks werden entsprechend der 100% Skalierung erstellt und sollten damit kompatibel zu Checks bleiben, die auf einem unskalierten Bildschirm erstellt wurden.

B.4.9 Änderungen mit möglichen Auswirkungen auf die Testausführung

Neue Java-Version für QF-Test

QF-Test wird nun mit Java 17 als eigenes JRE ausgeliefert. Die Ausführung von QF-Test mit Java 8 wird für QF-Test 7 noch unterstützt, ist aber abgekündigt und kann in einer zukünftigen Version ganz entfernt werden.

Diese Änderung kann Tests von Java-Anwendungen betreffen, wenn diese nicht mit ihrer eigenen oder einer systemspezifischen Java-Version gestartet werden, sondern mit der von QF-Test. Falls Sie eine solche Situation haben, gibt es zwei Möglichkeiten:

Kurzfristige Umgehung: Sie können über die QF-Test Java Konfiguration oder die Kommandozeile QF-Test weiterhin mit Java 8 ausführen.

Dauerhafte Lösung: Die bevorzugte Variante ist, eine zur Anwendung passende Java-Version im Java-SUT-Client starten(724) Knoten anzugeben, oder noch besser einen SUT-Client starten(728) Knoten zu verwenden, der die Anwendung über ein Skript oder ein ausführbares Programm startet, das die korrekte Umgebung inklusive der passenden Java-Version bereitstellt.

Neuer Algorithmus zur Bestimmung der zugehörigen Beschriftung

Hinweis

In den meisten Fällen sollte die Erkennung der Beschriftung einfach weiter funktionieren. Die primäre Ausnahme sind `ExtraFeatureResolver`, die mit dem `ExtraFeature` qfs:label arbeiten. Diese müssen angepasst werden, wie in Abschnitt 54.1.11(1168) beschrieben. Für Hilfe mit diesen Änderungen oder eine Möglichkeit, den neuen Algorithmus komplett abzuschalten, wenden Sie sich bitte an unser Support-Team.

Der Algorithmus zur Bestimmung der zugehörigen Beschriftung für eine Komponente wurde von Grund auf neu implementiert, um bessere Performanz und Klarheit sowie mehr Flexibilität zu erhalten. Mit den neuen qfs:label*- Varianten wie qfs:labelLeft oder qfs:labelText können spezifische Beschriftungen adressiert werden, qfs:labelBest ist das Gegenstück zum alten weiteren Merkmal qfs:label. Detaillierte Informationen zu den neuen Möglichkeiten finden Sie in Abschnitt 5.4.4(74).

Um ein Maximum an Rückwärtskompatibilität zu gewährleisten, wird der alte Algorithmus weiter gepflegt und dafür genutzt, das weitere Merkmal `qfs:label` zu bestimmen, so dass Tests, die auf Komponente(930) Knoten basieren, nicht negativ betroffen sein sollten. Bei Bedarf kann die Aufnahme über die Option Aufnahme von qfs:label*-Varianten(561) auf den alten Algorithmus mit `qfs:label` umgestellt werden.

Für die Wiedergabe von SmartIDs ist die Situation etwas anders. Ohne expliziten Kennzeichner sowie mit dem Kennzeichner ”label=” oder ”qlabel=” werden SmartIDs basierend auf dem neuen Algorithmus mit `qfs:labelBest` aufgelöst. In den meisten Fällen sollte dies wie zuvor funktionieren. Wo sich dadurch Fehler ergeben, können Sie ent-

weder die betroffene SmartID neu aufnehmen oder ihren Kennzeichner auf ”qfs:label=” setzen, um die Nutzung des alten Algorithmus zu erzwingen.

Weitere inkompatible Änderungen:

Web
- Die Unterstützung für den Internet Explorer wurde in QF-Test Version 6.0 abgekündigt und nun ganz entfernt, da dieser ausgelaufen ist. Die IE-spezifischen Prozeduren `qfs.web.browser.settings.enableCompatibilityMode` und `qfs.web.browser.general.isIE6` wurden aus der Standardbibliothek `qfs.qft` entfernt.

Web
- Die Unterstützung für Firefox Version 43 und älter mit dem Verbindungsmodus QF-Driver wurde in QF-Test Version 6.0 abgekündigt und nun ganz entfernt.

- Die Unterstützung für 32-Bit-Software ist abgekündigt und wird in einer der nächsten QF-Test Versionen entfernt. Dies betrifft Java-Versionen für QF-Test ebenso wie alle unterstützen SUT Varianten. Falls Sie weiterhin 32-Bit-Anwendungen unterstützen müssen, wenden Sie sich bitte an unser Support-Team.

- Das Format der HTTP-Header, die der Server-HTTP-Request(910) zurückliefert, wurde so angepasst, dass diese einfacher zu parsen und auszuwerten sind. Beispiele dafür werden in den Prozeduren `checkHttpResponseHeader` und `getHeaderValue` in der Web-Services Demo-Testsuite bereitgestellt, welche Sie über den Menüeintrag Hilfe→Beispiel Testsuiten erkunden... öffnen können.

- Für QF-Test wird zur Beschleunigung des Starts nun die Unterstützung von IPv6 deaktiviert, da QF-Test selbst diese nicht benötigt. Auf das SUT hat dies keinen Einfluss. Sollte IPv6 für ein Plugin benötigt werden, kann es über das Kommandozeilenargument `-ipv6`(982) reaktiviert werden.

- Beim Auflösen einer verschachtelten SmartID bzw. einer SmartID mit Scope wurde die bereits aufgelöste Komponente fälschlicherweise erneut in die Suche einbezogen, so dass z.B. `#Panel:Irgendein Titel@#Panel:<0>` zum Panel ”Irgendein Titel” führte, statt zum ersten darin enthaltenen Panel.

- Optionen wie Aufnahme von SmartIDs(559), die sowohl in QF-Test als auch im SUT einen Effekt haben, sollten nun einfach in einem Server-Skript(717) Knoten gesetzt werden. Die Weitergabe an alle SUT Clients erfolgt nun automatisch.

B.4.10 Version 7.0.0 - 27. April 2023

Neue Features:

- Der neue dunkle Modus ist nur der sichtbarste Aspekt der modernisierten Benutzeroberfläche mit etwas größeren Schriften und Symbolen und allgemein mehr Platz. Siehe Menü Ansicht→UI Theme sowie die Option Schriftgröße (pt)[493].

- SmartIDs haben das Preview-Stadium verlassen. Basierend auf der Neuentwicklung des Algorithmus zum Auffinden der assoziierten oder nächstgelegenen Beschriftung einer Komponente kombinieren sie in vielen Situationen Einfachheit mit präziser und effizienter Komponentenerkennung. Die neuen qfs:label*-Varianten finden ebenfalls in den klassischen Komponente[930] Knoten Anwendung. Näheres finden Sie in Abschnitt 5.6[81] und Abschnitt 5.4.4[74] sowie bei den zugehörigen Optionen in Abschnitt 41.4[559].

Web

- Der neue CustomWebResolver installieren[902] Knoten zur Implementierung eines CustomWebResolvers für Web-Anwendungen löst den eher kryptischen Prozeduraufruf[675] für `qfs.web.ajax.installCustomWebResolver` ab. Existierende Aufrufe funktionieren natürlich weiterhin, können aber einfach über den Kontextmenü-Eintrag Knoten konvertieren in konvertiert werden, wie in Abschnitt 51.1.2[1082] beschrieben.

Web

- Es gibt nun einen visuellen Inspektor zur Analyse von UI-Komponenten in Web- und Android-Anwendungen (vgl. Abschnitt 5.12.2[108]).

- Mit Hilfe der neuen Knoten Fehler[853], Warnung[859] und Nachricht[865] ist es nun möglich, Fehler, Warnungen oder einfache Meldungen an beliebiger Stelle in der Testsuite direkt zu protokollieren, mit dem zusätzlichen Mehrwert von konfigurierbaren Screenshots und Diagnose-Ausgaben.

- QF-Test Tests können nun sehr einfach in JUnit 5 Tests eingebunden werden. Dies vereinfacht die Ausführung von QF-Test Tests innerhalb einer Entwicklungsumgebung wie IntelliJ oder Eclipse. Außerdem gibt es nun ein Gradle-Plugin, um QF-Test in Build Pipelines zu integrieren, die mit Gradle ausgeführt werden. Details hierzu finden Sie in Abschnitt 29.5[407].

- Das XML-Format zum Speichern von Testsuiten ist nun flexibel konfigurierbar. So werden neue Testsuiten z.B. standardmäßig mit UTF-8-Kodierung und längeren Zeilen gespeichert. Bestehende Testsuiten behalten standardmäßig ihr Format bei, können aber in einem Rutsch konvertiert werden. Das neue Format bleibt dabei für ältere QF-Test Versionen lesbar. Weitere Informationen finden Sie in Abschnitt 41.1.2[488] und Abschnitt 44.1[971].

- XML-Reports werden nun standardmäßig mit UTF-8-Kodierung gespeichert.

- Es werden nun Tests für Anwendungen unterstützt, die auf Java 21 basieren.

- Groovy wurde auf Version 4.0.11 aktualisiert.

Web
- Der eingebettete Chrome Browser für QF-Driver wurde auf CEF Version 108 aktualisiert.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.31.

- Die eingebettete JUnit Bibliothek wurde auf Version 5.9.2 aktualisiert.

Web
- Es wird nun das Webframework Fluent UI React Version 8 unterstützt.

Web
- QF-Test unterstützt nun den Zugriff auf lokale Dateien aus getesteten Web-Anwendungen heraus über die File System Access API im WebDriver und im CDP-Driver-Verbindungsmodus, so dass Aufnahme und Wiedergabe für diesen Fall nun out-of-the-box funktionieren sollten.

Web
- Mit Pseudo-Attributen können Sie Resolver vereinfachen, die über JavaScript Werte aus dem Browser ermitteln. In manchen Fällen können diese auch die Performanz verbessern. Weitere Informationen finden Sie unter Web – Pseudoattribute(1131).

- QF-Test stellt nun Vorlagen für typische Skripte wie Resolver mit sehr einfachem Zugriff über das Vorlagen(722) Popup in Server-Skript(717), SUT-Skript(720) und Unit-Test(896) Knoten bereit. Es können auch eigene Vorlagen definiert werden, die dann in dieser Liste erscheinen.

- Über die neue Option Typen bei Baumknoten mit Namen einblenden(492) können redundante Teile der Beschriftung von Baumknoten in Testsuiten und Protokollen ausgeblendet werden.

- Beim Klick auf eine Zeilennummer in einem Skript wird nun die gesamte Zeile selektiert.

- Mithilfe der neuen Option Knoten beim Erstellen automatisch öffnen(496) ist es nun möglich, neue Knoten bereits aufgeklappt einzufügen.

- Der Name der Ergebnisvariablen eines Prozeduraufruf(675) Knotens wird nun im Baum angezeigt. Das Ergebnis des Aufrufs kann durch Aktivieren der Option Rückgabewerte von Prozeduren anzeigen(580) zusätzlich im Protokoll hinter dem Name der Prozedur angezeigt werden.

- Die Buttons in der QF-Test Werkzeugleiste können nun durch Ziehen mit der Maus umsortiert werden. Das ursprüngliche Layout lässt sich über das Menü Ansicht→Werkzeugleiste wiederherstellen.

- Die neue spezielle Variable `${qftest:language}` expandiert zur aktuell eingestellten Sprache der Benutzeroberfläche von QF-Test.

- Die neue spezielle Variable `${qftest:project.dir}` expandiert zum Verzeichnis des Projekts, zu dem die aktuelle Testsuite gehört.
- Baumknoten im SUT können nun über eine Kombination aus numerischen, textuellen und aus regulären Ausdrücken bestehenden Indizes angesprochen werden.
- Ein Datentreiber(646) Knoten unterhalb eines Testschritt(621) Knotens kann nun in einen verschachtelten Testschritt eingepackt werden.
- Über das Einfügen Menü oder das Tastenkürzel Umschalt-Strg-7 kann ein Kommentar(851) Knoten nun auch oberhalb des aktuell selektierten Knotens eingefügt werden.
- Falls ein Warten auf Client(758) Knoten zum Warten auf eine bestimmte GUI-Engine dient, wird diese Engine nun im Baum angezeigt.

Mac
- Das QF-Test Anwendungs-Icon für macOS wurde an moderne Standards angepasst.

Web
- Für den Mobile Emulation Mode wurden die Spezifikationen vieler neuer Geräte hinzugefügt.

Beseitigte Bugs:

- In der Prozedur `qfs.utils.xml.compareXMLFiles` wurde eine Exception geworfen, wenn eine der beiden Dateien leer war.
- Jython Skripte, die sich auf die System Property `python.security.respectJavaAccessibility=false` verlassen, um auf private Methoden und Attribute von Klassen zuzugreifen, funktionieren nun auch mit Java 9 und höher.

Web
- Bei Webtests im CDP-Driver oder WebDriver-Verbindungsmodus konnten der Check und der Komponenten-Aufnahmemodus nicht gleichzeitig aktiviert werden.

Web
- Die Ausführung von Mobile-Emulation-Tests im CDP-Driver-Verbindungsmodus wurde für neuere Chrome-Versionen aktualisiert.

Web
- In speziellen Fällen kann nun die Ausführungsgeschwindigkeit von Tests für Web-Anwendungen durch Nutzung von `MainTextResolver` oder `WholeTextResolver` Varianten ohne Standard-Parameter spürbar beschleunigt werden.

Web
- Bei Verwendung von @::-Syntax im `genericClasses` Parameter der Prozedur `qfs.web.ajax.installCustomWebResolver` konnte in speziellen Fällen ungewollt eine CSS-Klasse als Klasse des Knotens definiert werden.

Android
- Die Texteingabe für Android-Geräte mit API-Version 33 und höher wurde verbessert.

B.5 QF-Test Version 6.0

B.5.1 Version 6.0.5 - 15. März 2023

Neue Features:

- Es werden nun Tests für Anwendungen unterstützt, die auf Java 20 basieren.

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.27 bzw. ”2023-03” basieren.
- Das mit QF-Test ausgelieferte JRE wurde zu Zulu OpenJDK Version 8_362 aktualisiert.
- Groovy wurde auf Version 4.0.10 aktualisiert.
- Die eingebettete JUnit Bibliothek wurde auf Version 5.9.2 aktualisiert.

Web
- QF-Test nutzt nun bei Chromium-basierten Browsern deren ”neuen Headless-Modus”.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.29 und 7.30.

Web
- Der in QF-Test integrierte GeckoDriver wurde auf Version 0.32.2 aktualisiert.

Web
- QF-Test kann nun auch das Öffnen eines externen Browsers abfangen, wenn dies in der Anwendung über die Methode `Desktop.open()` ausgelöst wurde.

Beseitigte Bugs:

- Jython funktioniert nun auch mit Java 9 oder höher, wenn die System-Property `python.security.respectJavaAccessibility` auf `false` gesetzt wird, um private Methoden und Attribute von Klassen direkt ansprechen zu können.
- Die SAX-Variante der Prozedur `qfs.utils.xml.compareXMLFiles` in der Standardbibliothek `qfs.qft` ignoriert nun Leerzeichen am Anfang und Ende des `noCheck` Parameters.

Web
- Bei Tests mit dem WebDriver-Verbindungsmodus waren teilweise Dialoge nicht sichtbar, die aus IFRAMEs heraus aufgerufen wurden.

Web
- Im CDP-Driver-Verbindungsmodus unter Windows fokussierte QF-Test vor dem Abspielen eines harten Events nicht den korrekten Browser-Tab. Außerdem blieben nach Downloads leere Tabs manchmal offen.

Electron
- Es ist nun möglich, das Arbeitsverzeichnis für den Start einer Electron-Anwendung zu setzen.
- Das Modul `qf` steht jetzt auch in JUnit Groovy Skripten zur Verfügung.

B.5.2 Version 6.0.4 - 29. November 2022

Neue Features:

- Das mit QF-Test ausgelieferte JRE wurde zu Zulu OpenJDK Version 8_352 aktualisiert.
- Für das SUT unterstützt QF-Test nun auch das Semeru OpenJDK von IBM.

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.26 bzw. ”2022-12” basieren.
- Im Handbuch sind in Kapitel 32(435) nun Informationen zu den offiziellen QF-Test Docker Images zu finden.
- Mit den neuen Methoden `rc.pushOption` und `rc.popOption` können Optionen nun zeitweise überschrieben werden, ohne bestehende Werte zu beeinflussen. Die Prozeduren in der Standardbibliothek `qfs.qft` nutzen nun diese Methoden anstelle der weniger geeigneten `setOption`/`unsetOption`.
- Bei der Aufnahme von SmartIDs (vgl. Abschnitt 5.6(81)) bestimmt nun die Option Für SmartID immer die Klasse aufnehmen(559), ob immer die Klasse vorangestellt wird. Diese ist standardmäßig aktiv, da damit neben der Klarheit auch die Performanz bei der Wiedergabe deutlich verbessert werden kann.
- Die eingebettete JUnit Bibliothek wurde auf Version 5.9.1 aktualisiert.

Web
- Die eingebettete cdp4j-Bibliothek wurde auf Version 6.2.0 aktualisiert.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.28.

Web
- Der in QF-Test integrierte GeckoDriver wurde auf Version 0.32.0 aktualisiert.

Web
- Der CustomWebResolver für Angular Material wurde für die aktuelle Version des Frameworks aktualisiert.

Web
- In Web-Tests mit dem CDP-Driver-Verbindungsmodus kann nun auch der Druckdialog von QF-Test gesteuert werden.

Web
- Ein Auswahl-Event vom Typ ”reload” kann nun gleichbedeutend zum Typ ”refresh” verwendet werden, um die angezeigte Webseite erneut zu laden.

Electron
- Die Prozeduren im Package `qfs.web.browser.external` in der Standardbibliothek `qfs.qft`, mit deren Hilfe Starts von externen Browsern aus dem SUT abgefangen und zu QF-Test umgebogen werden können, unterstützen nun auch Electron-Anwendungen.

Beseitigte Bugs:

Web
- Das automatische Mapping für Tables und TreeTables im Web-Resolver für Primefaces wurde für Primefaces Version 12.0 aktualisiert.

Web
- Chrome konnte im QF-Driver-Verbindungsmodus in speziellen Fällen abstürzen, wenn das Laden einer Webseite fehlschlug.

Android
- In seltenen Fällen konnten spezielle Zeichen im Text von Android Komponenten die Verbindung zum Android-Gerät beenden.

B.5.3 Version 6.0.3 - 6. September 2022

Neue Features:

- Das mit QF-Test ausgelieferte JRE wurde zu Zulu OpenJDK Version 8_345 aktualisiert.

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.25 bzw. ”2022-09” basieren.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.27
- Es wurden die Methoden `getFirstChild`, `getNextSibling`, `getPreviousSibling`, `getFirstElementChild`, `getNextElementSibling` und `getPreviousElementSibling` zur Pseudo DOM-API von QF-Test hinzugefügt. Details hierzu finden Sie in Abschnitt 54.10.1(1254).

Beseitigte Bugs:

Web
- Beim Hochladen einer Datei im CDP-Driver-Verbindungsmodus wird nun eine TestException geworfen, wenn die angegebene Datei nicht vorhanden ist.

Web
- In seltenen Einzelfällen ging bei der Aufnahme von Mausklicks in Web-Anwendungen die Information über Unterelemente verloren.

Web
- Vereinzelt wurde das Browserfenster fälschlicherweise aufgrund eines WebDriver Timeouts für einen anderen Frame geschlossen.

Web
- Overlays in Web-Anwendungen, die mit dem Angular-Framework erstellt sind, wurden teilweise nicht korrekt erkannt.

SWT
- Für SWT-Anwendungen unter Linux wurde bei der Wiedergabe einer Auswahl für ein ToolItem in einem vertikalen ToolBar ggf. das falsche Item ausgelöst.

B.5.4 Version 6.0.2 - 20. Juli 2022

Neue Features:

- Das saubere Beenden von laufenden QF-Test Instanzen - insbesondere im Batchmodus - wurde deutlich verbessert. Mit den neuen Kommandozeilenargumenten `-allow-shutdown [<Shutdown-ID>]`(978) und `-shutdown <ID>`(990) können nun einzelne Prozesse dediziert über ihre Prozess-ID oder eine vorher vergebene Shutdown-ID angesprochen werden. Die früheren Kommandozeilenargumente `-allowkilling`(978) und `-kill-running-instances`(983) funktionieren noch, sind aber abgekündigt.
- Das Kommandozeilenargument `-clearglobals`(979) funktioniert nun auch im Calldaemon-Modus (siehe Abschnitt 25.2.2(347)). Hierfür wurde die `DaemonRunContext` API um zwei neue Methoden erweitert, `setGlobals` und `clearGlobals` (siehe Abschnitt 55.2(1277)).

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.26.
- Die schnellere Variante der Prozedur `qfs.utils.xml.compareXMLFiles` in der Standardbibliothek `qfs.qft` unterstützt nun ebenfalls das Sortieren.
- Bei speziellen Merkmalen von Komponenten wie ”Tab: some tab” oder ”Label: some label” kann für SmartIDs das vorangestellte ”Tab:”, ”Label:” etc. nun weggelassen werden. Details zur Syntax von SmartIDs finden Sie in Abschnitt 5.6(81).

Beseitigte Bugs:

- Beim Protokollieren von Abbildern für einzelne Fenster nahm QF-Test fälschlicherweise auch Bilder für eingebettete Fenster von anderen GUI-Engines auf.
- Beim Bearbeiten von Optionen wird nun die doppelte Vergabe von Hot-Keys verhindert.

Web
- Die Pseudo-DOM Methode `DomNode.getElementsByTagName` gibt nun auch ”Slotted Elements” zurück.

Web
- Text aus ”Slotted Elements” wurde für das Merkmal von Web-Komponenten nicht berücksichtigt.

Web
- Im Direkt-Download-Modus werden die vorhandenen Dateien nicht mehr überschrieben, sondern bekommen eindeutige Namen.

SWT
- Für SWT unter Windows ist die Wiedergabe von TAB Tastendrücken nun deutlich schneller.

B.5.5 Version 6.0.1 - 9. Juni 2022

Neue Features:

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.24 bzw. ”2022-06” basieren.
- Mit einer neuen Variablengruppe können Sonderzeichen in SmartIDs bequem geschützt werden. `${quotesmartid:...}` kümmert sich um die Zeichen ’@’, ’&’ und ’%’ für Unterelemente ebenso, wie um die speziellen SmartID Zeichen ’:’, ’=’, ’<’ und ’>’.

Beseitigte Bugs:

Android
- Die Texteingabe für Android-Anwendungen wurde verbessert.

Web
- In Web-Anwendungen wurde ein GenericClassNameResolver, der für ”DOM_NODE” registriert war, nicht korrekt aufgerufen.

SWT
- Bei der Eingabe von Text mittels einzelner Events für SWT-Anwendungen unter Ubuntu 22 konnte die Reihenfolge der eingegebenen Zeichen durcheinander geraten. QF-Test spielt diese Events nun mit verbesserter Synchronisation ab.

B.5.6 Änderungen mit möglichen Auswirkungen auf die Testausführung

Web
- Die Unterstützung von Firefox im Verbindungsmodus QF-Driver, der auf Firefox bis Version 43 limitiert war, ist nun abgekündigt und wird in einer der nächsten QF-Test Versionen entfernt. Bitte verwenden Sie stattdessen aktuelle Firefox Versionen mit dem Verbindungsmodus WebDriver.

Web
- Die Unterstützung von Internet Explorer, der nun offiziell eingestellt wird, ist abgekündigt und wird in einer der nächsten QF-Test Versionen entfernt.

Web
- Die Unterstützung von Opera im Verbindungsmodus WebDriver ist nun abgekündigt und wird in einer der nächsten QF-Test Versionen entfernt. Bitte verwenden Sie stattdessen den Verbindungsmodus CDP-Driver.

Web
- Beim Test von Web-Anwendungen im CDP-Driver oder WebDriver-Verbindungsmodus werden Slottable Nodes nun nicht mehr als direkte Kinder des Host Node aufgeführt, sondern als Kinder des zugewiesenen Slot Node. Es ist unwahrscheinlich, dass diese Änderung bestehenden Tests schadet - falls doch, setzen Sie sich bitte mit dem Support von QFS in Verbindung.

Swing
- Die interne Adressierung von Spalten in einer Swing JTable Komponente wurde von model-basiert auf view-basiert umgestellt. Dies hat keine Auswirkung, wenn Spalten mit textuellem Index @... oder %... angesprochen werden oder wenn die Sortierung der Spalten in Model und View übereinstimmt. Falls ein auf numerischen Spaltenindizes &... basierender Test fehlschlägt, können Sie alternativ zur Anpassung der Indizes das alte Verhalten mit folgendem SUT-Skript wiederherstellen: `rc.setOption(Options.OPT_SWING_TABLE_USE_VIEW_COLUMN, false).`

Windows-Tests
- Die interne API der UI-Automation-Bibliothek wurde überarbeitet, um die dort verwendeten Klassennamen zu vereinfachen (aus ”AutomationWindow” wurde zum Beispiel einfach ”Window”). Wenn Sie das `uiauto` Modul direkt in Ihren Skripten verwenden und dort Klassen direkt referenzieren, müssen Sie ggf. die Klassennamen entsprechend der aktualisierten JavaDoc anpassen.

B.5.7 Version 6.0.0 - 17. Mai 2022

Neue Features:

- Mit der neuen Android Engine können mit QF-Test nun Android-Anwendungen im Emulator oder auf echten Geräten getestet werden. Nähere Informationen finden Sie in Kapitel 16(245).
- Auch wenn QF-Test noch mit JRE Version 8 - aktuell auf Stand 8_332 - ausgeliefert wird, kann QF-Test bereits mit Java 17 gestartet werden (siehe Kommandozeilenargument `-java <Programm>` (abgekündigt)(977)). Damit erhalten Sie eine gestochen scharfe Darstellung auf skalierten Monitoren sowie die Möglichkeit, Plugins zu nutzen, die eine neuere Java-Version benötigen.
- Es werden nun Tests für Anwendungen unterstützt, die auf Java 19 basieren.

- Um die Wahrscheinlichkeit zu reduzieren, dass während eines Testlaufs versehentlich sensible Daten als Screenshot aufgenommen werden, erstellt QF-Test nun nur noch Screenshots von relevanten Bildschirmen, auf denen ein Fenster von QF-Test oder einem verbundenen SUT dargestellt werden. Diese Default-Einstellung ist sinnvoll für die persönliche Arbeitsumgebung. Für reine Testsysteme kann es sinnvoll sein, diese über die neue Option Abbilder auf relevante Bildschirme beschränken(588) zu deaktivieren.

- Der HTML-Report wurde gründlich überarbeitet. Neben einem ansprechenderen Design in vielen kleinen Details zur besseren Lesbarkeit werden Bildschirmabbilder und Fehlermeldungen nun beim Anklicken als Overlay dargestellt, inklusive Navigation zwischen mehreren Abbildern.

- Die Erstellung von Reports kann nun auch über einen neuen Button in der Werkzeugleiste des Protokoll-Fensters ausgelöst werden.

- In Reports wird nun der Name einer Testsuite, der im Attribut Name des Wurzelknotens angegebenen werden kann, anstelle ihres Dateinamens verwendet. Dies kann im Dialog zur Generierung von Reports eingestellt werden bzw. im Batchmodus über das neue Kommandozeilenargument `-report-include-suitename`(988).

- Zum besseren Verständnis des Laufzeitverhaltens eines Tests kann nun im Protokoll die Anzeige der relativen Dauer von Knoten über einen neuen Button in der Werkzeugleiste oder das Ansicht Menü aktiviert werden. Weitere Optionen finden Sie in Abschnitt 7.1.3(141) sowie bei den Optionen Relative Dauer anzeigen(579) und Anzeigeform für relative Dauer(579).

- Nach Aktivieren der neuen Option Bei Warnungen Screenshots erstellen(589) werden Bildschirmabbilder auch für Warnungen im Protokoll erstellt, zusätzlich zu denen für Fehler und Exceptions.

- Knoten einer Testsuite können nun über das @link Doctag mit externen Ressourcen oder Dateien verknüpft werden. Per Rechts-Klick kann das Ziel dann im Browser oder mit dem Dateityp assoziierten Anwendung geöffnet werden. Weitere Informationen finden Sie in Doctags für Reporting und Dokumentation(1360).

- Groovy wurde auf Version 4 aktualisiert.

- Die neuen Parameter `warningDelay` und `errorDelay` in der Prozedur `qfs.utils.logMemory` in der Standardbibliothek `qfs.qft` steuern eine kurze Verzögerung bei Überschreitung von `warningLimit` oder `errorLimit`, gefolgt von einer weiteren Garbage-Collection und einer erneuten Prüfung.

- Die Performanz und das Speicherverhalten der Prozedur `qfs.utils.xml.compareXMLFiles` wurden verbessert.

Web
- Die Darstellung und ”Schwuppdizität” der Hervorhebungen im Check-Modus wurde im CDP-Driver und WebDriver Verbindungsmodus verbessert.

Web
- Die Verarbeitung von WebComponenten mit ShadowDOMs und Slots wurde für den Test von Web-Anwendungen im CDP-Driver und WebDriver-Verbindungsmodus verbessert (Im QF-Driver Modus noch in Entwicklung): Shadow Root Nodes werden jetzt als einziges Kind des entsprechenden Host Node aufgeführt und Slotted Nodes werden als Kinder des zugewiesenen Slot Node repräsentiert.

Web
- Das Abrufen von Text im CDP-Driver-Verbindungsmodus ist nun signifikant schneller und die DOM Hierarchie ist konsistent mit den anderen Verbindungsmodi.

Web
- Abbild-Checks in ”headless” Browsern sind nun performanter.

Web
- QF-Test unterstützt nun mehrere parallele Downloads in Webtests mit dem CDP-Driver-Verbindungsmodus.

Web
- Die neue Prozedur `qfs.web.browser.settings.setDirectDownload` in der Standardbibliothek `qfs.qft` erlaubt, Dateien direkt ohne Download-Dialog in das angegebene Verzeichnis zu speichern. Aktuell ist diese Funktion nur für den CDP-Driver-Verbindungsmodus verfügbar.

Web
- Der eingebettete Chrome Browser für QF-Driver wurde auf CEF Version 100 aktualisiert.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.23 und 7.24.

Web
- Die eingebettete cdp4j-Bibliothek wurde auf Version 5.5.0 aktualisiert.

Web
- Der in QF-Test integrierte GeckoDriver wurde auf Version 0.31.0 aktualisiert.

Web
- Die Methode `FrameNode.getFrameElement()` wurde der QF-Test Pseudo-DOM API hinzugefügt.

Web
- Über den Parameter `consoleOutputValue` der Prozeduren `qfs.web.browser.settings.doStartupSettings` und `qfs.web.browser.settings.setTerminalLogs` in der Standardbibliothek `qfs.qft` kann nun auch der Typ der Terminalausgaben definiert werden.

Windows-Tests
- Die eingebettete UI-Automation-Bibliothek wurde auf Version 0.7.0 aktualisiert.

Mac
- Die unter macOS beim Test mit Safari eventuell auftretenden Hinweis-Dialoge bezüglich der Browser-Automatisierung werden nun von QF-Test automatisch behandelt, so dass sie die Testausführung nicht mehr blockieren.

Preview Features:

Die folgenden Features sind noch nicht vollständig umgesetzt, aber bereits so weit ausgereift, dass sie großen Mehrwert bringen und die freigegebenen Funktionen ohne Sorge um Rückwärtskompatibilität genutzt werden können.

- SmartIDs ermöglichen eine flexible, einfache Wiedererkennung von Komponenten direkt aus der QF-Test ID der Komponente, ohne vorher Information über Komponenten aufzuzeichnen. Detaillierte Informationen hierzu finden Sie in Abschnitt 5.6[81] und Komponente-Knoten versus SmartID[51].
- Dank der neuen Integration mit Robot Framework können QF-Test Prozeduren als Robot Framework Keywords verwendet werden.

Beseitigte Bugs:

Windows-Tests
- Die Mini-Installer Dateien für Windows - `minisetup.exe` und `minisetup_admin.exe` - können nun wie die gesamte Installation im silent und very-silent Modus ausgeführt werden.

Mac
- QF-Test konnte beim Abspielen harter Tastaturevents unter macOS abstürzen.
- Bei Verwendung von geteilten Protokollen wurde die Maximalzahl von Bildschirmabbildern bei ausgeschalteter Option Bildschirmabbilder für geteilte Protokolle separat zählen[588] nicht immer eingehalten.
- Die Prozedur `qfs.autowin.acrobat.saveAsText` in der Standardbibliothek `qfs.qft` funktioniert nun auch für Acrobat Reader Versionen 22.1 und höher.
- Die Ausführung von `rc.clearTestRunListeners` in einem Server-Skript unterbrach bei einem Aufruf im Batchmodus mit dem `-verbose [<level>]`[994] Kommandozeilenargument die Ausgaben.

Web
- Die Erkennung des Chrome-Fensters für semiharte Klicks wurde verbessert.

Web
- Für aktuelle Opera Versionen wird im WebDriver-Verbindungsmodus nun der automatische Download der benötigten ChromeDriver Version unterstützt.

Electron
- Beim Test einer Electron-Anwendung im CDP-Driver-Verbindungsmodus wurde die Fenstergröße versehentlich beim Start automatisch angepasst.

Electron
- Popups wurden nicht erkannt, wenn eine Electron-Anwendung im CDP-Driver-Verbindungsmodus getestet wurde.

B.6 QF-Test Version 5.4

B.6.1 Version 5.4.3 - 11. März 2022

Neue Features:

- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.23 bzw. ”2022-03” basieren.
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.22.

Beseitigte Bugs:

- In sehr speziellen Fällen konnte die Wiederausführung eines Knotens, ausgelöst durch das @rerun Doctag, zu falschen Werten auf dem Variablen-Stapel führen.

B.6.2 Version 5.4.2 - 18. Februar 2022

Neue Features:

Web
- QF-Test unterstützt nun Tests mit Opera 84.

Web
- Der Kendo UI und Smart GWT CustomWebResolver wurden für die aktuelle Version des jeweiligen Frameworks aktualisiert.

Web
- Die eingebettete cdp4j Bibliothek wurde auf Version 5.5.0 aktualisiert.

Web
- Die QF-Test Pseudo-DOM-API wurde um die Methode `FrameNode.getFrameElement()` ergänzt.

Web
- Bei der Prozedur `qfs.web.browser.settings.setTerminalLogs` in der Standardbibliothek `qfs.qft` kann nun über den Parameter `consoleOutputValue` auch der Typ der Terminal-Logs definiert werden.

Beseitigte Bugs:

Web
- Unter Windows wurden für Chrome und Edge ab Version 98 die Cookies beim Browserstart nicht korrekt gelöscht.

Web
- Beim Umsortieren der Elemente einer Webseite wurde die Sortierung in QF-Test vereinzelt inkorrekt synchronisiert.

- Die Prozedur `qfs.daemon.startRemoteSUT` in der Standardbibliothek `qfs.qft` funktionierte nicht korrekt, wenn der QF-Test Daemon mit einem Keystore zur Absicherung der Kommunikation via TLS gestartet wurde.

Swing

- Bei der Ermittlung des Merkmals einer Swing Komponente hatte durch einen Fehler in den QF-Test Versionen 5.3.4 bis 5.4.1 ein Tooltip eine höhere Gewichtung als ein explizit zugewiesenes Label.

B.6.3 Version 5.4.1 - 20. Januar 2022

Neue Features:

Web

- QF-Test unterstützt nun Tests mit Opera 83.

Web

- Die eingebettete cdp4j Bibliothek wurde auf Version 5.4.1 aktualisiert.

Beseitigte Bugs:

Web

- Im CDP-Driver-Verbindungsmodus konnte QF-Test 5.4.0 vereinzelt Elemente auf einer Webseite nicht finden, die nach dem Laden der Seite hinzugefügt wurden.

Web

- Es ist nun möglich, auch dann Console-Log-Ausgaben eines Browsers, der mit dem CDP-Driver-Verbindungsmodus gestartet wurde, im QF-Test Terminal auszugeben, wenn diese nicht vom Typ String sind.

Web

- Die Chrome-DevTools können nun abgekoppelt werden, wenn ein Web-Test im CDP-Driver-Verbindungsmodus entwickelt wird.

Electron

- In einigen Fällen wurden Dialoge in Electron-Anwendungen im WebDriver-Verbindungsmodus nicht korrekt geschlossen.

Electron

- Die Verbindung zu einer Electron-Anwendung konnte im CDP-Driver-Verbindungsmodus unter Windows nicht hergestellt werden, falls die Anwendung langsam startete.

Mac

- Der Befehl `automac.sendText` führte auf neueren macOS-Systemen zu einem Absturz von QF-Test.

Mac

- `${qftest:os.version}` liefert jetzt für Windows 11 sowie macOS 11 und neuer korrekte Werte.

B.6.4 Änderungen mit möglichen Auswirkungen auf die Testausführung

- Es wurde ein Fehler im `qftest` Startskript für Linux beseitigt. Durch diesen wurden beim Verarbeiten von Kommandozeilenargumenten mit einem geschützten $-Ausdruck im Wert eines -variable oder -option Arguments der $-Ausdruck versehentlich expandiert.

Swing
- Für Komponenten in einer Swing `JScrollPane`, insbesondere `JTree` und `JTable`, war die Zuordnung des weiteren Merkmals qfs:label inkonsistent.

Web
- Tests mit dem Browser Microsoft Edge (Legacy) werden nicht mehr unterstützt, da diese Version des Edge Browsers endgültig eingestellt wurde. Dies beeinflusst in keiner Weise die Unterstützung des aktuellen Microsoft Edge Browsers.

Web
- Bei der Ausführung des Knotens Warten auf Laden des Dokuments(880) wird nun besser geprüft, ob ein Dokument tatsächlich neu geladen wurde. Wenn ein bestehender Test bisher die schwache Prüfung voraussetzte, können dort nun Fehler bei der Testausführung auftreten. In solchen Fällen ist es ratsam, die betroffenen Warten auf Laden des Dokuments(880) Knoten zu überprüfen und ggf. zu deaktivieren bzw. zu löschen oder durch Warten auf Komponente(876) Knoten zu ersetzen. Alternativ kann das alte Verhalten durch Aktivieren der Option Internen Ladestatus von Web-Dokumenten beim Rescan zurücksetzen (vor 5.4)(573) wiederhergestellt werden.

Web
- Bei Web-Anwendungen werden nun automatisch die Attribute ”aria-checked” und ”aria-selected” für Check Boolean(812) Knoten vom Typ selected bzw. checked berücksichtigt.

B.6.5 Version 5.4.0 - 15. Dezember 2021

Neue Features:

- Es werden nun Tests für Anwendungen unterstützt, die auf Java 18 basieren.

Electron
- Electron-Anwendungen können nun über den CDP-Driver-Verbindungsmodus getestet werden, was wesentlich effektiver ist und keine Einbindung des problematischen Moduls @electron/remote in die zu testende Electron-Anwendung erfordert.

- Im Dialog für die Einstellungen von QF-Test wird nun eine Suchfunktion angeboten.

- Man kann nun Abbilder von und nach QF-Test kopieren, insbesondere bei Check Abbild(828) Knoten und Bildschirmabbildern im Protokoll.

- Mausevent(775) Knoten mit einem Wert von 4 im Attribut Modifiers(778), entsprechend einem Klick mit der rechten Maustaste, werden im Baum nun als ”Rechtsklick” dargestellt.

Web
- Der eingebettete Chrome Browser für QF-Driver wurde auf CEF Version 95 aktualisiert.
- Groovy wurde auf Version 3.0.9 aktualisiert.
- Die JUnit Bibliothek wurde auf Version 5.8.1 aktualisiert.

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.22 bzw. ”2021-12” basieren.

Web
- QF-Test unterstützt nun Tests mit Opera 80, 81 und 82.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.20 und 7.21.

Web
- Für eine Web-Anwendung kann das Attribut Name des Check-Typs(814) eines Check Boolean(812) Knotens nun auch den Wert ”attribute:<name>” haben, um den Wahrheitswert des Attributs <name> im Zielknoten zu prüfen.
- Das neue Doctag @outputFilter kann in Knoten zum Start eines Clients verwendet werden, um unerwünschte Ausgaben aus dem QF-Test Terminal zu filtern. Weitere Informationen hierzu finden Sie in Abschnitt 62.3(1363).
- Wenn das Attribut Defaultwert(874) in einem Variable setzen(871) Knoten die QF-Test ID einer Komponente in der Form ${id:...} enthält, kann nun über einen Rechts-Klick und entsprechende Auswahl im Kontextmenü zur Zielkomponente gesprungen oder diese im SUT hervorgehoben werden.

Beseitigte Bugs:

- Die Suche nach ungenutzten aufrufbaren Knoten berücksichtigte nicht alle Referenzen und konnte somit Knoten melden, die eigentlich noch benötigt wurden.
- QF-Test versucht nun zu vermeiden, dass Nicht-Daemon-Threads im SUT erzeugt werden - einschließlich der vom RMI Modul generierten Threads. Solche Threads konnten verhindern, dass ein Prozess nach Schließen des letzten Fensters im SUT sauber terminiert.
- Performanz und Speicherverbrauch wurden an diversen Stellen verbessert.

Swing
- Der Titel eines Swing `JPanel` mit einer `TitledBorder` wird nun korrekt als Merkmal ermittelt.

Swing
- Die Reihenfolge der Komponenten in einer Swing `JSplitPane` konnte von der Reihenfolge der Erzeugung oder des Austauschens dieser Komponenten abhängen. QF-Test nimmt nun unabhängig davon immer die Reihenfolge links->rechts bzw. oben->unten.

Web
- QF-Test unterstützt nun auch den automatischen Download von ChromeDriver für die Google Chrome Varianten ”Dev” und ”Canary”.

Web
- Nach einer Frame-Navigation im CDP-Driver Modus zeichnete QF-Test Events vereinzelt nicht korrekt auf.

Web
- Es wurde eine Exception behoben, die zu Problemen bei der Initialisierung eines Dokuments im CDP-Driver Modus führen konnte.

Web
- Das ”label”-Attribute eines OPTION Elements wird nun bei der Ermittlung von dessen Namen berücksichtigt.

Web
- Wenn ein Browser-Fenster im CDP-Verbindungsmodus abstürzt wird nun automatisch ein Fehler geloggt und das Fenster geschlossen.

Web
- Die Behandlung von Unload-Dialogen bei Web-Tests im CDP-Driver-Verbindungsmodus wurde verbessert.

Web
- Bei der Ausführung von Web-Tests mit einem headless Browser im CDP-Driver-Verbindungsmodus versucht QF-Test nun nicht mehr, für Datei Up- oder Download temporär einen Swing Dialog anzuzeigen. Damit sollten reine headless Web-Tests mit CDP-Driver im Batchmodus nun auch in einem Container komplett ohne X-Server lauffähig sein.

Web
- Im CDP-Driver-Verbindungsmodus wurde die automatische Synchronisierung mit Animationen in Web-Anwendungen wurde verbessert.

Web
- Konsolenausgaben von Firefox im Webdriver-Verbindungsmodus wurden in früheren QF-Test Versionen auf die Standardausgabe des Prozesses umgeleitet, um diese auf JavaScript Fehler prüfen zu können. Aufgrund der potenziell hohen CPU und Speicherbelastung wurde dieses Verhalten deaktiviert. Über den Parameter `consoleOutputValue` mit Wert 1 im Aufruf der Prozedur `qfs.web.browser.settings.doStartupSettings` in der Standardbibliothek `qfs.qft` kann es wiederhergestellt werden.

B.7 QF-Test Version 5.3

B.7.1 Version 5.3.4 - 30. September 2021

Neue Features:

- Der in QF-Test integrierte GeckoDriver wurde auf Version 0.30.0 aktualisiert.

Beseitigte Bugs:

- Es wurde ein Speicherleck in QF-Test beseitigt, welches mit Version 5.3.3 hereingekommen war.
- Die Prozedur `qfs.autowin.acrobat.saveAsText` in der Standardbibliothek `qfs.qft` funktioniert nun auch für Acrobat Reader Versionen 21.6 und höher.

B.7.2 Version 5.3.3 - 14. September 2021

Neue Features:

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.21 bzw. ”2021-09” basieren.

Web
- QF-Test unterstützt nun Tests mit Opera 78 und 79.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.17, 7.18 und 7.19.

Web
- Die integrierte cdp4j Bibliothek wurde auf Version 5.4.0 aktualisiert.

Beseitigte Bugs:

- Der Dialog zum Bearbeiten der Details eines Knoten wird nun immer korrekt im Bereich der sichtbaren Bildschirme angezeigt, auch wenn z.B. ein vorher angeschlossener Monitor entfernt oder auf eine RDP-Session gewechselt wurde.

B.7.3 Version 5.3.2 - 21. Juli 2021

Beseitigte Bugs:

Web
- Einige Webseiten, die spezielle benutzerdefinierte HTML-Elemente enthalten, konnten nicht im CDP-Driver Verbindungsmodus getestet werden.

Web
- Im CDP-Driver-Verbindungsmodus wurden in seltenen Fällen Leerzeichen im Text nicht korrekt eingelesen.

Web
- Im CDP-Driver-Verbindungsmodus wird das keyCode Feld von KeyEvents, die für Texteingaben generiert werden, nun korrekt gesetzt.

Web
- Im CDP-Driver-Verbindungsmodus wurde die Position von Elementen in verschachtelten IFRAMEs fehlerhaft berechnet.

Web

- Das Attribut Name des Browser-Fensters[883] im Knoten Warten auf Laden des Dokuments[880] wurde ignoriert.

B.7.4 Version 5.3.1 - 15. Juni 2021

Neue Features:

- Es werden nun Tests für Anwendungen unterstützt, die auf Java 17 basieren.

SWT

- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.20 bzw. ”2021-06” basieren.
- Die Performanz im CDP-Verbindungsmodus bei dynamischen Änderungen des Webseiten-Inhaltes wurde verbessert.
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.16.

Electron

- QF-Test unterstützt nun auch Electron-Anwendungen mit Electron Version 14 oder neuer, wenn in die App das `@electron/remote` Modul eingebunden wurde.
- QF-Test unterstützt nun auch Tests mit Opera 77.

Beseitigte Bugs:

Web

- Bei Web-Anwendungen wurde die Gültigkeit von Texteingaben via einzelner Events durch zusätzliche explizit definierte Keycodes verbessert.

B.7.5 Änderungen mit möglichen Auswirkungen auf die Testausführung

- Jython Probleme mit Zeichenkodierungen wurden reduziert: Es ist nun möglich, Jython Literale (wörtliche Zeichenketten wie `"abc"`) als 16-Bit Unicode-Strings zu behandeln, entsprechend der Darstellung in Java und damit QF-Test. Aus Kompatibilitätsgründen bleibt die neue Option Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java)[485] standardmäßig deaktiviert, falls QF-Test eine bereits bestehende ältere Systemkonfiguration vorfindet.

  In Abschnitt 11.4.5[201] finden Sie detaillierte Informationen darüber, warum Sie diese Option aktivieren sollten und wie Sie mit eventuell daraus resultierenden Problemen umgehen. Die Chancen stehen gut, dass Ihre Jython Skripte einfach funktionieren und der Umgang mit Zeichenketten deutlich sauberer wird. Falls nicht, können Sie die Option wieder zurücksetzen oder die nicht kompatiblen Stellen beheben. Wir mussten dies in unseren über 1600 Testsuiten, von denen manche

über 20 Jahre alt sind, nur in einer handvoll Fälle tun. Der Abschnitt Behandlung von Problemen mit Jython und Zeichenkodierungen 11.4.5[203] beschreibt die gängigen Problemfälle und natürlich ist unser Support immer für Sie da.

- Das Standardverzeichnis für das Profil von Firefox heißt bei der Ausführung im WebDriver-Verbindungsmodus nun nicht mehr `mozProfile` sondern `firefoxProfile` und es wird nun direkt verwendet statt als temporäre Kopie. Dieses Verhalten ist konsistent zur Testausführung mit QF-Driver, hat aber den Nebeneffekt, dass Einstellungen nun von einem Test zum nächsten erhalten bleiben und bei Bedarf beim nächsten Browser-Start passend überschrieben werden müssen. Um das ursprüngliche Verhalten wieder herzustellen, muss die Option `OPT_WEBDRIVER_COPY_MOZPROFILE` vor dem Start des Browsers auf `true` gesetzt werden.

- Die Prozeduren im Package `qfs.utils.ssh` der Standardbibliothek `qfs.qft` und das zu Grunde liegende Jython Modul `ssh` wurden aktualisiert. Sie verwenden nun standardmäßig RSA Public Key Authentication mit der privaten Schlüsseldatei `/.ssh/id_rsa` statt des veralteten DSA, das von den meisten aktuellen ssh Servern nicht mehr unterstützt wird.

- Die Option Kompakte Protokolle erstellen[590] ist im interaktiven Modus nun standardmäßig deaktiviert. Bestehende Systemkonfigurationen sind davon nicht betroffen und im Batchmodus hat die Option keine Wirkung. Dort wird Kompaktifizierung über das Kommandozeilenargument `-compact`[979] gesteuert.

B.7.6 Version 5.3.0 - 20. Mai 2021

Neue Features:

Web

- Der neue Browser-Verbindungsmodus CDP-Driver ergänzt QF-Driver und WebDriver zur Ansteuerung von Chromium basierten Browsern über das Chrome DevTools Protokoll. Durch das direkte Ansprechen des Browsers ohne Umweg über das WebDriver Protokoll sind Geschwindigkeit, Stabilität und Funktionsumfang vergleichbar mit QF-Driver (und das nach dessen Performance-Optimierung, s.u.). Im Gegensatz zu QF-Driver, der auf Chrome unter Windows beschränkt ist, funktioniert der CDP-Driver mit Google Chrome, Microsoft Edge und Opera unter Window, Linux und macOS, was die Web-Testautomatisierung mit QF-Test auf eine neue Ebene hebt.

Web

- Die Performance von Web-Tests mit QF-Driver für Chrome wurde signifikant verbessert. Die beobachtete Erhöhung der Geschwindigkeit liegt zwischen 10% und über 500%.

- Die Oberfläche von QF-Test ist nun moderner, klarer und aufgeräumter, mit einem einheitlich flachen Look, weniger Linien und wunderschönen neuen Icons, welche die bekannte Bildsprache beibehalten und damit sofort wiedererkennbar bleiben. Auch die HTML-Versionen des Handbuchs und des Tutorials sowie der Reports und Test-Dokumentation haben eine Auffrischung erhalten.

- Die Verwendung internationaler Zeichensätze in Jython Skripten ist nun viel einfacher. Wenn die neue Option Literale (wörtliche Zeichenketten) in Jython sind Unicode (16-Bit wie in Java)[485] aktiviert ist, werden Literale (wörtlich definierte String-Konstanten wie `"abc"`) in Jython Skripten als 16-Bit Unicode-Strings interpretiert und sind damit äquivalent zu Strings in Java und den anderen Skriptsprachen von QF-Test. Detaillierte Informationen hierzu finden Sie in Abschnitt 11.4.5[201].

Web
- Fehler in der Browser-Konsole werden nun besser erkannt und, je nach Einstellung der Option Behandlung von Fehlern in einer Web-Anwendung[570], in das QF-Test Protokoll übernommen. Zudem kann über die neue Prozedur `qfs.web.browser.settings.setTerminalLogs` in der Standardbibliothek `qfs.qft` eingestellt werden, ob und wie Meldungen der Browser-Konsole im QF-Test Terminal angezeigt werden sollen.

Web
- Der eingebettete Chrome Browser für QF-Driver wurde auf CEF Version 89 aktualisiert.

Web
- Der in QF-Test integrierte GeckoDriver wurde auf Version 0.29.1 aktualisiert.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.14 und 7.15.

Web
- QF-Test unterstützt nun Tests mit Opera 76.

- Das mit QF-Test ausgelieferte JRE wurde zu Zulu OpenJDK Version 8_292 aktualisiert.

- Der Schnellstart-Assistent hat nun seinen eigenen Toolbar-Button. Dieser kann nun - wie jeder andere nicht benötigte Toolbar-Button - per Rechts-Klick aus der Werkzeugleiste ausgeblendet werden.

- Der 'Warten auf Verschwinden'-Knoten hat nun einen eigenen Eintrag im Einfügen Menü und seine Anwendung auf Unterelemente wurde vereinfacht: Die Ausführung ist erfolgreich, sobald entweder die Parent-Komponente des Unterelements nicht (mehr) vorhanden ist, oder das Unterelement selbst.

- Bei einem Testlauf mit dem Kommandozeilenargument `-verbose [<level>]`[994] werden QF-Test Variablen in den Namen von Knoten nun auch für die Ausgabe in der Konsole expandiert.

- Die Voreinstellung für den verfügbaren Arbeitsspeicher für QF-Test wurde auf 1024 MB hochgesetzt. Die Konfiguration bestehender QF-Test Installationen ist davon nicht betroffen.

Swing

- Unterelemente von Swing `JComboBox` Komponenten können nun relativ zur `JComboBox` adressiert werden, ohne dass die Popup-Liste dafür benötigt wird.

Linux

- Die Fähigkeit, Fenster des SUT bei Bedarf in den Vordergrund zu bringen und den Eingabefokus zu setzen ist für eine stabile Testautomatisierung sehr wichtig. Unter Linux verwendet QF-Test nun eine aktualisierte, zuverlässigere Methode, Fenster unabhängig von den Desktop-Einstellungen nach vorne zu bringen, sofern die Option Beim Nach-vorne-Bringen Fenster in den Vordergrund zwingen(543) nicht deaktiviert ist.

- Meldungen für Exceptions im Protokoll oder in Fehlerdialogen werden nun an Wortgrenzen umgebrochen. Diese Darstellung kann über die Option Zeilen in Meldungen für Exceptions umbrechen(591) deaktiviert werden.

Beseitigte Bugs:

- Das Einhängen eines Resolvers mit der generischen Methode `resovlers.addResolver()` funktionierte nicht in SUT-Skripten mit der Sprache JavaScript.

Web

- Die Zoomstufe des Browsers wird nun beim Löschen des Browser-Cache auf 100% zurückgesetzt.

- Die Prozedur `qfs.swing.startup.startWebstartSUT` sorgt nun dafür, dass der `jnlp` Parameter für die Verwendung auf der Kommandozeile von Linux Systemen korrekt geschützt wird, um Effekten durch eventuell enthaltene Sonderzeichen vorzubeugen.

Windows-Tests

- Das weitere Merkmal qfs:label wurde für Elemente in einem TabPanel einer nativen WPF-Anwendung mit nicht korrekt ermittelt.

Web

- Seit QF-Test Version 5.2.2 wurden in vereinzelten Fällen Elemente einer Webseite fälschlicherweise als unsichtbar eingestuft.

Mac

- QF-Test funktioniert nun auch wieder unter macOS Versionen älter als 10.14.

- Die Suche nach dem weiteren Merkmal qfs:label ist bei einem Label oberhalb der Zielkomponente nun etwas toleranter bezüglich der horizontalen Ausrichtung.

Swing

- Die schnelle Wiedergabe von mehreren Mausklicks auf dieselbe Stelle einer Webswing-Anwendung konnte bei Umleitung über den Browser zu ungewollten Doppelklicks führen.

B.8 QF-Test Version 5.2

B.8.1 Version 5.2.3 - 9. März 2021

Neue Features:

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.19 bzw. ”2021-03” basieren.

Beseitigte Bugs:

- In einzelnen Fällen konnten unglücklich platzierte Kommentar Knoten unerwünschte Seiteneffekte bei der Testausführung haben, z.B. die Ausführung eines ’Vorbereitung’- oder ’Aufräumen’-Knotens nur für den Kommentar.
- Das Timing beim Stoppen von Prozessen am Ende der Batchausführung wurde verbessert.

Web
- Im WebDriver-Verbindungsmodus wurden Veränderungen auf der Website eventuell übersehen, wenn die Seite zu viele Elemente enthielt.

B.8.2 Version 5.2.2 - 12. Februar 2021

Neue Features:

- Es werden nun Tests für Anwendungen unterstützt, die auf Java 16 basieren.

Web
- QF-Test unterstützt nun Tests mit Opera 74.

Web
- Der in QF-Test integrierte GeckoDriver wurde auf Version 0.29.0 aktualisiert.
- Über die Tastenkürzel Strg-/ bzw. Strg-7 kann ein neuer Kommentarknoten im Baum der Testsuite eingefügt werden.

Beseitigte Bugs:

Web
- Die Geschwindigkeit der Ausführung von Web-Tests wurde an einigen Stellen verbessert, insbesondere für Firefox unter Linux mit WebDriver.

Web
- In sehr seltenen Fällen wurde Chrome geschlossen, wenn während einer JavaScript-Ausführung ein Frame neu geladen wurde.

Web
- Das verzögerte Anhängen von Shadow DOM-Elementen wird nun korrekt erkannt.

Swing

- In einzelnen Sonderfällen konnten bei der verbesserten Event-Synchronisation für Swing Events übersehen werden, was zu langsamerer Testausführung führte.
- Wenn ein Testfall(599) mit gesetztem Attribut Fehlschlagen erwartet wenn...(604) nicht fehlschlägt, sollte das als Fehler behandelt werden. Dieser Fehler wurde fälschlicherweise seinerseits als erwarteter Fehler angezeigt.
- Die Reihenfolge von Parametern wird nun auch automatisch angepasst, wenn die Zielprozedur eines Prozeduraufruf(675) Knotens über den Auswahldialog geändert wird.

B.8.3 Version 5.2.1 - 3. Dezember 2020

Neue Features:

SWT

- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.18 bzw. ”2020-12” basieren.
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.12.

Beseitigte Bugs:

- Aufgrund von falschen Zugriffsrechten funktionierte Jython unter Linux nicht, wenn QF-Test von verschiedenen Anwendern installiert und gestartet wurde.
- Das mit QF-Test ausgelieferte JRE wurde zurück zu Zulu OpenJDK gewechselt. Die Version bleibt bei 8_275.
- Manchmal führte die Anzeige eines JavaScript-Dialogs zu einem Deadlock.
- Mehrere Details der Webswing Integration wurden verbessert, darunter das korrekte Filtern von KeyEvents, Fokuswechsel für eingebettete JavaFX Komponenten und eine bessere Trennung der Client-Prozess in den Demo-Testsuiten.
- Die Daemon-Verbindung wurde teilweise nicht aufgebaut, wenn auf Client- und Serverseite unterschiedliche Java-Versionen verwendet wurden.

B.8.4 Änderungen mit möglichen Auswirkungen auf die Testausführung

- Das Testen von Anwendungen, die mit Java 7 ausgeführt werden, wird nicht weiter unterstützt.

Windows-Tests
- Viele Prozeduren im Package qfs.qft.autowin in der Standardbibliothek wurden mit Hinblick auf die wesentlich besser geeignete Windows-Engine abgekündigt.
- Durch das Update des JRE von QF-Test werden Kurven und graphische Element im PDF-Client mit minimal anderem Anti-Aliasing dargestellt. Dis kann zu Fehlern bei Check Abbild[828] Knoten führen. Nachdem solche Probleme auch bei zukünftigen Updates des internen JRE nicht auszuschließen sind, setzen Sie für solche Checks am besten das Attribut Algorithmus zum Bildvergleich[831] auf ”algorithm=similarity;expected=0.98”.

B.8.5 Version 5.2.0 - 10. November 2020

Neue Features:

- QF-Test unterstützt nun das integrierte Testen von Swing und JavaFX-Anwendungen, die mit Hilfe der Technologien Webswing oder JPro im Browser dargestellt werden. Erklärungen zum Konzept finden Sie in Kapitel 20[307] und eine Webswing Demo-Testsuite über den Menüeintrag Hilfe→Beispiel-Testsuiten erkunden..., Eintrag ”Webswing SwingSet Suite”.

Mac
- QF-Test wird nun durch Apple beglaubigt und startet somit auf modernen macOS Systemen ohne Warnmeldung.

Web
- QF-Test unterstützt nun auch Tests mit dem Microsoft Edge Browser unter Linux.
- Das mit QF-Test ausgelieferte JRE wurde zu Liberica OpenJDK Version 8_275 aktualisiert.

Web
- Der eingebettete Chrome Browser für QF-Driver wurde auf CEF Version 85 aktualisiert.
- Groovy wurde auf Version 3.0.6 aktualisiert.
- Jython wurde auf Version 2.7.2 aktualisiert.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.11

Web
- Der eingebettete Geckodriver wurde auf Version 0.28.0 aktualisiert.

Web
- QF-Test unterstützt nun Tests mit Opera 72.

Web
- Für den Mobile Emulation Mode wurden die Spezifikationen vieler neuer Geräte hinzugefügt.
- Die JUnit Bibliothek wurde auf Version 5.7.0 aktualisiert.

Web

- Auf Windows Systemen mit skalierter Darstellung startet QF-Test einen Browser im QF-Driver Modus nun mit aktiviertem Kompatibilitätsmodus, so dass die skalierte Darstellung transparent durch Windows erfolgt und Tests mit Ausnahme von Abbild Checks analog zum unskalierten Modus funktionieren.
- Beim Start von QF-Test können nun Optionen über das Kommandozeilenargument `-option <Name>=<Wert>`(985) gesetzt werden.
- Kommandozeilenargumente für QF-Test können nun an beliebiger Stelle "." und "-" Zeichen enthalten. Zudem können Groß- oder Kleinbuchstaben nach Belieben verwendet werden.
- Beim Testen von Java-Anwendungen kann QF-Test nun Aufrufe abfangen, die ein natives Browserfenster öffnen, um stattdessen einen von QF-Test gesteuerten Browser mit der entsprechenden URL zu starten. Beispiele finden Sie in den Demo-Testsuiten "CarConfig Swing Testprojekt" und "CarConfig JavaFX Testprojekt", zu erreichen über das Menü Hilfe→Beispiel-Testsuiten erkunden....
- Die neue Prozedur `qfs.utils.waitForClientOutput` in der Standardbibliothek `qfs.qft` unterstützt bei der Synchronisation mit Terminal-Ausgaben im SUT.
- Viele neue Knotenkonvertierungen sind jetzt möglich.
- Der Server-HTTP-Request(910) Knoten unterstützt nun auch die `PATCH` Methode.
- Mit Hilfe der beiden neuen Prozeduren `qfs.utils.sendKey` und `qfs.utils.sendText` in der Standardbibliothek `qfs.qft` kann Text im aktuell fokussierten Element des aktiven Fensters eingegeben werden.
- Mit Hilfe der neuen Option Hinweis anzeigen, wenn keine Events aufgenommen wurden(509) kann der "Es wurden keine Events aufgenommen" Dialog unterdrückt werden.
- Beim Zusammenführen von Protokollen im Batchmodus kann nun das Kommandozeilenargument `-mergelogs-masterlog [<Datei>]`(983) mit dem Wert "append" für das Argument `-mergelogs-mode [<Modus>]`(984) kombiniert werden. Die angehängten Protokolle werden dabei als externe Protokolle gespeichert, was den Bedarf an Arbeitsspeicher sowohl für das Zusammenführen als auch das anschließende Öffnen des Protokolls minimiert.
- Analog zu Jython können in Skript-Knoten mit Groovy und JavaScript nun wichtige Exceptions auch ohne expliziten Import verwendet werden.
- Beim Anpassen der Parameter von aufrufenden Knoten über das Menü Operationen→Parameter von Referenzen anpassen kann nun explizit festgelegt werden, ob fehlende Parameter ergänzt, überzählige Parameter entfernt und/oder die Reihenfolge der Parameter angepasst werden soll.

Beseitigte Bugs:

- Wenn während eines automatischen Reruns das Protokoll geöffnet wurde, konnte es zu einer Exception kommen.
- Verschlüsselte Verbindungen zum QF-Test Daemon können nun auch über die externe Daemon-API hergestellt werden.
- Beim Erstellen von Reports wurden Miniaturbilder selbst dann erzeugt, wenn -report-thumbnails nicht angegeben war.
- Der Unit-Test(896) Knoten unterstützt nun in Jython-Skripten den Befehl `self.assertEqual` korrekt.
- Texteingaben in Swing und JavaFX Komponenten wurden bei Erkennung eines in Java eingebetteten Browsers verlangsamt.

Swing
- Die Synchronisation von Events unter starker Last bei Swing basierten Anwendungen wurde verbessert.

Swing
- Zeilenumbrüche werden bei der Texteingabe mit Einzelevents für eine Swing JTextArea jetzt korrekt wiedergegeben.

Windows-Tests
- Bei Windows-Anwendungen wurden Komponenten unter Umständen nicht sichtbar gescrollt, um einen harten Mausklick oder einen Abbildcheck durchzuführen.

Web
- Bei einem Browser im WebDriver Modus konnte es nach einem fehlgeschlagenen Frame-Fokuswechsel zu einer StackOverflowException kommen.

Web
- In einigen Fällen wurde der MSEdgeDriver nicht korrekt heruntergeladen.

Web
- Checks auf Elementen innerhalb eines Shadow DOMs konnten nicht aufgezeichnet werden.

Web
- Bedingte (unsichtbare) Trennstriche werden nun implizit ignoriert.

Electron
- In einigen Fällen wurden Dialoge in Electron-Anwendungen leer dargestellt.

JavaFX
- Die Sichtbarkeit von JavaFX Komponenten wurde gegebenenfalls nicht korrekt bestimmt, wenn diese in eine Swing Komponente eingebettet waren.

SWT
- Bei SWT-Version 4.17 unter Windows wurden Highlight-Rechtecke für Menüs nicht richtig zurückgesetzt.

B.9 QF-Test Version 5.1

B.9.1 Version 5.1.2 - 15. September 2020

Neue Features:

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.17 bzw. ”2020-09” basieren.

Beseitigte Bugs:

- In seltenen Fällen konnte QF-Test bei knappem Arbeitsspeicher während der Bildkomprimierung abstürzen.

B.9.2 Version 5.1.1 - 26. August 2020

Neue Features:

Web
- QF-Test unterstützt nun auch Tests mit Opera 70.

Web
- Der eingebettete GeckoDriver wurde auf Version 0.27.0 aktualisiert.

Web
- Für WebDriver-basierte Tests mit Chrome/Chromium wird die Seitenisolierung nun automatisch deaktiviert.

Web
- QF-Test unterstützt nun auch Tests mit JxBrowser 7.10.
- Ein Link zur JavaScript Dokumentation wurde im Hilfe-Menü eingefügt.

Beseitigte Bugs:

- Der Unit-Test(896) Knoten sucht auf dem Classpath nun auch nach JUnit 5 (Jupiter) Tests.

Electron
- In einigen Fällen wurden Klicks auf native Menüs in Electron-Anwendungen nicht korrekt aufgenommen.

Web
- Der Cache von Chromium-basierten Browsern wurde teilweise nicht korrekt gelöscht.
- Die Option Attribut ’Lokale Variable’ standardmäßig aktivieren(593) wird nun auch beim Einfügen von kopierten Prozedur(672) Knoten als Prozeduraufruf(675), generell bei Knotenkonvertierungen sowie bei der Aufnahme von Checks beachtet.

Web
- Wenn Embedded-Browser-Container (z.B. JxBrowser) gleichzeitig erstellt und gelöscht wurden, konnte es zu einem Deadlock kommen.
- Kommentar(851) Knoten in der Procbuilder-Konfigurationsdatei konnten die Aufnahme von Prozeduren behindern.

Mac
- Unter macOS werden JVM-Optionen (die mit "-J-" beginnen) nun korrekt ausgewertet.

Windows-Tests
- Die Texteingabe hat bei Windows-Anwendungen unter Umständen nicht richtig funktioniert, wenn die AltGr-Taste beteiligt war.

Web
- Beim Test in einem fensterlosen Browser wurden die unsichtbaren Dialogfenster teilweise nicht ordnungsgemäß geschlossen.

Web
- Bei Tests mit dem SWTBrowser konnte eine ClassNotFoundException auftreten.
- Bei der Erstellung von Prozeduren mittels Procbuilder unter Verwendung von FORCECREATION wurden die Trennpunkte der Packagestruktur in Unterstriche umgewandelt.
- Bei der Generierung von Prozeduren mittels Procbuilder ist es nun auch möglich, Werte aus den Weiteren Merkmalen von übergeordneten Knoten als Fallback zu verwenden.

B.9.3 Änderungen mit möglichen Auswirkungen auf die Testausführung

- Durch das Update des JRE von QF-Test werden Kurven und graphische Element im PDF-Client mit minimal anderem Anti-Aliasing dargestellt. Dis kann zu Fehlern bei Check Abbild(828) Knoten führen. Nachdem solche Probleme auch bei zukünftigen Updates des internen JRE nicht auszuschließen sind, setzen Sie für solche Checks am besten das Attribut Algorithmus zum Bildvergleich(831) auf "algorithm=similarity;expected=0.98".

  Das JRE Update kann außerdem zu Problemen bei der Kommunikation zwischen QF-Test und dem QF-Test Lizenzserver führen, falls der Lizenzserver mit einer sehr alten Java-Version betrieben wird, die mit den geforderten Schlüssellängen für SSL in aktuellen Java-Versionen nicht zurecht kommt. Am besten aktualisieren Sie in diesem Fall den QF-Test Lizenzserver auf die aktuelle QF-Test Version und nutzen das mitgelieferte JRE.

- Die Bibliothek `jniwrapper` wird standardmäßig nicht mehr geladen, da unsere alte `jniwrapper` Version QF-Test unter neueren JDKs zum Absturz bringt. Module mit nativen Abhängigkeiten wie `autowin` wurden umgeschrieben, so dass

Sie nicht länger darauf aufbauen. Zudem wurden alle Verweise auf `jniwrapper` aus der Standardbibliothek `qfs.qft` entfernt.

Falls Sie noch Skripte in Ihren Testsuiten haben, die jniwrapper nutzen, sollten Sie versuchen, diese so umzuschreiben, dass die Abhängigkeit nicht mehr besteht. Bitte wenden Sie sich an unseren Support, falls Sie dafür Hilfe benötigen.

Als Übergangslösung können Sie solche Skripte (mit alten JDKs bei denen jniwrapper nicht abstürzt) wie folgt wiederbeleben:

- Kopieren Sie die Dateien aus `misc/jniwrapper` im QF-Test Installationsverzeichnis nach `qftest` im QF-Test Plugin-Verzeichnis. Sie finden diese Verzeichnisse über den Hilfe→Info Dialog im Reiter 'Systeminfo' als `dir.version` und `dir.plugin`.
- Fügen Sie entweder ein Jython Server-Skript in Ihre Startsequenz ein mit

```
from com.jniwrapper import DefaultLibraryLoader
from java.io import File
DefaultLibraryLoader.getInstance().addPath \
  (File(rc.getStr("qftest", "dir.plugin") + "/qftest"))
```

oder die folgende Groovy Variante

```
import com.jniwrapper.DefaultLibraryLoader
DefaultLibraryLoader.getInstance().addPath
  (new File(rc.getStr("qftest", "dir.plugin") +
       "/qftest"))
```

- Der ChromeDriver für alte Chrome Versionen (älter als 72) wird nicht mehr mit QF-Test ausgeliefert.
- Das Testen von Anwendungen mit Java 7 wird mit dieser QF-Test Version noch unterstützt. Allerdings ist die weitere Unterstützung für Java 7 hiermit abgekündigt und wird mit QF-Test Version 5.2 eingestellt.

B.9.4 Version 5.1.0 - 8. Juli 2020

Video

Video:

QF-Test 5.1.0
https://www.qftest.com/de/yt/version-51-eingebettete-browser-51.html

Neue Features:

- Es werden nun Tests für Anwendungen unterstützt, die auf Java 15 basieren.

- Das mit QF-Test ausgelieferte JRE wurde zu Zulu OpenJDK Version 8_252 aktualisiert.

Web
- Die Aufnahme und Wiedergabe von Tests für eingebettete Browser wurde substantiell verbessert.
- Es wird nun JxBrowser Version 7, eingebettet in Swing, JavaFX oder Eclipse/SWT-Anwendungen unterstützt.

Electron
- Bei Electron-Anwendungen werden nun auch native Dialoge unterstützt.

Web
- QF-Test unterstützt nun Tests mit Opera 69.

Web
- Die Unterstützung für das Webframework Qooxdoo wurde für Qooxdoo Version 6 aktualisiert.
- HTML-Reports können nun mit Hilfe einer JavaScript-Datei namens `user.js` angepasst werden. Weitere Informationen hierzu finden Sie in Abschnitt 24.1.4(335).
- Das `automac` Modul wurde um Methoden zum Abspielen von Tastatur- und Mausevents erweitert. Weitere Informationen finden Sie in Kapitel 53(1147).
- Dem Wurzelknoten einer Testsuite kann über das Attribut Name(596) nun ebenfalls ein Name zugewiesen werden, der in der Baumansicht dargestellt wird.
- Die neue Option Attribut 'Lokale Variable' standardmäßig aktivieren(593) legt fest, ob das Attibut 'Lokale Variable' in neu erstellten Knoten aktiv vorbelegt wird.
- Die Prozedur `qfs.utils.dragAndDrop` in der Standardbibliothek `qfs.qft` hat einen neuen optionalen Parameter `eventDelay` zur Steuerung der Wiedergabegeschwindigkeit.
- Es ist nun möglich einen CSV-Datei Knoten in einen Excel-Datei Knoten zu konvertieren und umgekehrt.
- Testsuite-Reiter können nun über die Tastenkombinationen Umschalt-Strg-Bild hoch und Umschalt-Strg-Bild runter nach links oder rechts bewegt werden.

Beseitigte Bugs:

Web
- Die Prozedur `qfs.web.browser.settings.setLocale` funktioniert nun auch für WebDriver Verbindungen zu Chromium basierten Browsern.

B.10 QF-Test Version 5.0

B.10.1 Version 5.0.3 - 17. Juni 2020

Neue Features:

SWT

- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.16 bzw. ”2020-06” basieren.
- Die mitgelieferte Bibliothek `jsch.jar`, die vom Package `qfs.utils.ssh` in der Standardbibliothek `qfs.qft` benötigt wird, wurde zu Version 0.1.55 aktualisiert, um moderne Linux Systeme wie Ubuntu 20 zu unterstützen.

Beseitigte Bugs:

- Die eingebettete WebP Grafik-Komprimierungsbibliothek wurde auf Version 1.0.0 zurückgestellt um Inkompatibilitäten zu vermeiden.

Web

- Die Komponentenerkennung ist fehlgeschlagen wenn Komponenten nicht ganzzahlige Größen hatten.
- Ein Aufruf von `rc.callProcedure` in den Parametern eines Prozeduraufruf(675) Knotens konnte in seltenen Fällen dazu führen, dass die globalen Variablen vom Variablen-Stapel verschwinden.
- Die spezielle Syntax ${qftest:engine.<componentid>} zur Bestimmung der GUI-Engine einer Komponente funktioniert jetzt auch falls <componentid> ein ’@’, ’%’ oder ’&’-Zeichen enthält.
- Durch Selektieren eines Wertes in der Fehlerliste eines Protokolls und anschließende doppelte Ausführung von ”Als Filter setzen” konnte zu einer ArrayIndexOutOfBoundsException führen.

B.10.2 Version 5.0.2 - 5. Mai 2020

Neue Features:

- Die WebP Grafik-Komprimierungsbibliothek wurde auf Version 1.1.0 aktualisiert.
- Die Wiedergabe von Tastaturevents im JXBrowser ist nun stabiler.
- Der Kontrast von Icons in der Werkzeugleiste wurde verbessert, insbesondere für deaktivierte Buttons.

- QF-Test unterstützt nun Tests mit Opera 68.

Beseitigte Bugs:

- Angular 9 wird nun korrekt automatisch erkannt.
- Der CSV-Datei(664) Knoten liest eine UTF-8 kodierte CSV-Datei mit BOM nun auch dann richtig ein, wenn diese mit einem geschützten komplexen Ausdruck beginnt.

Windows-Tests
- Der Windows-Anwendung starten(743) Knoten kann nun wieder über eine im Attribut Fenstertitel angegebene Klasse (-class) die Verbindung mit einem Client herstellen.
- Die Fehlerbehandlung und Wiederholung im Fall eines fehlerhaften automatischen Downloads von WebDriver Bibliotheken wurde verbessert.

Windows-Tests
- Komponentenaufnahme mit flacher Hierarchie hat nicht funktioniert.
- Der PDF-Client kann jetzt Text Komponenten prüfen, die ausschließlich aus null-Zeichen ”\u0000” bestehen und behandelt diese als Leerstring.

B.10.3 Version 5.0.1 - 2. März 2020

Neue Features:

Windows-Tests
- Es wurde eine neue Demo-Testsuite für die ”Windows 10”-Anwendung ”Rechner” hinzugefügt.
- QF-Test unterstützt nun Opera 67 mit dem Operadriver 80.0.3987.100.

SWT
- QF-Test unterstützt nun auch Tests für Anwendungen, die auf Eclipse/SWT 4.15 bzw. ”2030-03” basieren.

Beseitigte Bugs:

Windows-Tests
- Bei der Komponentenaufnahme (Ganzes Fenster) wurden Elemente innerhalb eines WPF TabPanels weggelassen.
- Die Prozedur `qfs.database.executeSelectStatement` funktioniert nun wieder mit Datenbanken, die einen expliziten `db.commit()` Befehl benötigen.
- Bei Ausführung eines Server-HTTP-Request(910) Knotens wurde fälschlicherweise bei einem Server-Fehler der Rückgabewert nicht in eine Variable geschrieben.

JavaFX
- Bei der Berechnung eines Hashwerts für eine JavaFX Grafik konnte eine NullPointerException im Terminal erscheinen.

Windows-Tests

- Ein Windows TextField wurde eventuell vor der Texteingabe nicht geleert.

Windows-Tests

- In WPF Windows-Anwendungen konnte gelegentlich fälschlicherweise eine ModalDialogException geworfen werden.

B.10.4 Wesentliche neue Features in Version 5

Hinweis

Eine detaillierte Aufstellung der Neuigkeiten finden Sie in den Release Notes für QF-Test Version 5.0.0 weiter unten.

Die folgenden neuen Features wurden für Version 5 von QF-Test implementiert:

Beschreibung	Weiterführende Informationen
Neue GUI-Engine: Windows	Testen nativer Windows-Anwendungen(234)
Modernisierte Benutzeroberfläche für QF-Test	QF-Test sieht nun moderner aus
Tests mit Java 14	Anwendungen mit Java 14 können nun getestet werden
Testsuiten mit Kommentaren	Kommentar(851) Knoten direkt im Baum der Testsuite
Edge auf Chromium Basis	Tests mit dem finalen Edge auf Chromium Basis sind nun möglich
Datei-Download mit Hilfe des Server-HTTP-Request(910) Knotens	Attribut Antwort in Datei speichern(914)

Tabelle B.1: Neue Features in QF-Test 5

Änderungen mit möglichen Auswirkungen auf die Testausführung:

- Der Server-HTTP-Request(910) Knoten wirft nun eine Exception, wenn der Statuscode größer als oder gleich 400 ist. Dieses Verhalten kann über das neue Attribut Fehlerstufe bei HTTP-Statuscode >= 400(914) gesteuert werden.
- Die Optionen Über QF-Test Agent verbinden(594) und AWT EventQueue instrumentieren(594) werden nicht mehr in der Systemkonfigurationsdatei gespeichert und können nur noch zur Laufzeit per Skript geändert werden. Weiter Informationen hierzu finden Sie in Abschnitt 41.13(594).

Nicht mehr unterstützte Software:

Hinweis

Eine detaillierte Aufstellung der Systemvoraussetzungen und der unterstützten Technologieversionen finden Sie in Abschnitt 1.1(3).

- Das Testen von Anwendungen mit Java 6 wird nicht mehr unterstützt.

B.10.5 Version 5.0.0 - 6. Februar 2020

Neue Features:

- Mit der neuen Windows Engine kann QF-Test nun native Windows-Anwendungen testen.
- Es werden nun Tests für Anwendungen unterstützt, die auf Java 14 basieren.
- Mit dem neuen Kommentar(851) Knoten kann die Struktur und Lesbarkeit von Testsuiten und Protokollen verbessert werden.
- Mit Hilfe des neuen Attributs Antwort in Datei speichern(914) im Server-HTTP-Request(910) Knoten ist es nun möglich, Dateien herunterzuladen.

Windows-Tests

- Es wurde ein Package speziell für Windows-Anwendungen in der Standardbibliothek `qfs.qft` hinzugefügt.
- Unter Windows 10 wird QF-Test jetzt auf skalierten hochaufgelösten Monitoren korrekt dargestellt.

Web

- Im Handbuch wurde das Kapitel Testen von Webseiten(227) überarbeitet und ein Abschnitt (Abschnitt 51.1.2(1082)) mit der Beschreibung der Prozedur `qfs.web.ajax.installCustomWebResolver` in der Standardbibliothek hinzugefügt.
- Die Option Benachrichtigen nach(532) hat nun eine Auswahlmöglichkeit, um auch nach einem fehlerfreien Test einen Infodialog anzuzeigen.
- Beim Öffnen der Ergebnisliste nach einer Suche wird der Suchdialog nun automatisch geschlossen.
- Mit Hilfe der neuen Prozedur `qfs.util.click` in der Standardbibliothek `qfs.qft` ist es möglich, einen Klick auf eine beliebige Bildschirmkoordinate abzuspielen.
- Es ist jetzt möglich im Schnellstart-Assistenten für Electron auszuwählen, dass der benötigte ChromeDriver automatisch erkannt werden soll.
- Beim Kopieren eines Prozeduraufruf(675), Testaufruf(614) oder Bezug auf Abhängigkeit(635) Knotens wird nun zusätzlich der Name des Zielknotens als Text in die Zwischenablage übernommen.

Web

- Der mitgelieferte GeckoDriver wurde auf Version 0.26.0 aktualisiert.

Web

- Unter Windows können Webseiten mit Microsoft Edge ab Version 78 nun auch im Headless-Modus getestetet werden.

Web

- QF-Test unterstützt nun auch Opera 66 mit dem Operadriver 79.0.3945.79.
- Über die neue Variable `engine.$(componentId)` in der speziellen `qftest` Gruppe ist es möglich, herauszufinden, zu welcher GUI-Engine eine bestimmte Komponente gehört.
- Der Projektbaum in QF-Test wird jetzt mit Hilfe der natürlichen Sortierreihenfolge angeordnet.
- Im Schnellstart-Assistenten für Mobile Emulation wurden Daten für mehrere neue mobile Geräte ergänzt.

Beseitigte Bugs:

- Bei knappem Speicher wurden eventuell Abbilder nicht im Protokoll gespeichert.
- Eine sporadisch beim Erstellen einer Testsuite aus einem Protokoll auftretende Exception wurde beseitigt.
- Finally(711) Knoten in einem Try(704) werden jetzt auch dann ausgeführt, wenn innerhalb des Try(704) Knotens ein sofortiger Rerun ausgelöst wird.
- Die `ImageWrapper` Methoden loggen nun eine Warnung wann immer diese Methoden fehlschlagen.
- In sehr seltenen Fällen konnte es nach der Testausführung dazu kommen, dass die Strg Taste nach Beendigung der Wiedergabe im ”gedrückt” Status blieb.

Anhang C

Tastaturkürzel

C.1 Navigation und Editieren

Diese Tabelle gibt einen hilfreichen Überblick der von QF-Test angebotenen Tastaturkürzel zur Navigation und Editieren:

Windows/Linux	*macOS*	*Funktion*
Datei-Navigation		
Strg-N	⌘-N	Neue Testsuite
Strg-O	⌘-O	Öffnen
Strg-S	⌘-S	Speichern
-	⇧-⌘-S	Speichern unter
Editieren		
Strg-Z	⌘-Z	Rückgängig machen
Strg-Y	⇧-⌘-Z	Wiederherstellen
Suchen und Ersetzen		
Strg-F	⌘-F	Suchen
F3	F3	Suche wiederholen
Strg-G	⌘-G	Erneut suchen
Strg-H	^-H	Ersetzen
Workbench-Ansicht		
Strg-Bild ab	^-⇟	Zur nächsten Testsuite wechseln
Strg-Bild auf	^-⇞	Zur vorhergehenden Testsuite wechseln
Strg-Umschalt-PageDown	Strg-Umschalt-⇟	Akutelle Testsuite mit nächster Testsuite tauschen
Strg-Umschalt-PageUp	Strg-Umschalt-⇞	Aktuelle Testsuite mit vorheriger Testsuite tauschen
Alt-1, 2, ... 9	⌘-1, 2, ... 9	Zur 1./2./.../9. Testsuite wechseln

F5	⌘-R	Projektverzeichnis aktualisieren
Shift-F5	⇧-⌘-R	Projektverzeichnis neu einlesen
F6	F6	Fokus zwischen Suite und Projekt wechseln
Umschalt-F6	⇧-F6	Aktuelle Suite im Projektbaum selektieren, dazu falls nötig die Projektansicht öffnen
Strg-F6	^-F6	Zur zuletzt aktiven Testsuite wechseln Erneuter Druck von F6 bei gedrücktem Strg wechselt weiter zurück, gleichzeitiges Drücken von Umschalt kehrt die Richtung um
Strg-L	^-L	Letztes Protokoll öffnen
-	^-⌘-F	Vollbildmodus
-	⌘-,	Optionen öffnen
-	⌘-W	Testsuite schließen
Alt-F4	⌘-Q	QF-Test schließen
Baumansicht		
Hoch / Runter / Rechts / Links	↑ / ↓ / → / ←	Basisnavigation
Alt-Hoch	⌥-↑	Zum vorhergehenden Knoten der selben Ebene springen
Alt-Runter	⌥-↓	Zum nächsten Knoten der selben Ebene springen
Alt-Rechts	⌥-→	Knoten rekursiv ausklappen
Alt-Links	⌥-←	Knoten rekursiv einklappen
Umschalt-Hoch	⇧-↑	Auswahl nach oben erweitern
Umschalt-Runter	⇧-↓	Auswahl nach unten erweitern
Strg-Hoch	^-↑	Nach oben bewegen, ohne Einfluss auf Auswahl
Strg-Runter	^-↓	Nach unten bewegen, ohne Einfluss auf Auswahl
Strg-Rechts	^-→	Baum nach rechts scrollen
Strg-Links	^-←	Baum nach links scrollen
Leertaste	Leertaste	Auswahl des aktuellen Knoten umschalten
Strg-Backspace	^-⌫	Zurück zum zuletzt ausgewählten Knoten
Umschalt-Strg-Backspace	^-⇧-⌫	Nächsten Knoten anwählen
Strg-.	^-.	Baum aufräumen
Alt-Return	⌥-↩	Öffnen des Eigenschaftenfensters eines Knotens
Umschalt-F10 / Windows Kontextmenu-Taste	⇧-F10	Öffnen des Popupmenüs eines Knotens
F2	F2	Zum Namen oder der QF-Test ID des Knoten springen
Tabellen		

Umschalt-Einfügen	⌥-↓	Neue Zeile einfügen
Umschalt-Return	⇧-↩	Zeile bearbeiten
Umschalt-Entfernen	⇧-⌦	Zeile löschen
Umschalt-Strg-Hoch	^-⇧-↑	Ausgewählte Zeile nach oben verschieben
Umschalt-Strg-Runter	^-⇧-↓	Ausgewählte Zeile nach unten verschieben
F2	F2	Ausgewählten Eintrag bearbeiten
Return	↩	Änderungen bestätigen
Escape	⎋	Änderungen verwerfen
Umschalt/Strg-Hoch/Runter	^/⇧-↑/↓	Mehrfachselektion
Strg-X / C / V	⌘-X / C / V	Ausschneiden / Kopieren / Einfügen
Umschalt-Strg-Rechts	^-⇧-→	Bei Variablen: Werte durchreichen, also `x` -> `$(x)`
Code Editor		
Strg-Leertaste	^-Leertaste	Eine Liste der vorhandenen QF-Test Variablen für Skripte oder eine Liste von möglichen Methoden (nur für bestimmte Knoten)
Strg-P	^-P	Prozedurdefinition finden (nur für Zeilen mit Prozeduraufrufen)
Strg-T	^-T	Testdefinition finden (nur für Zeilen mit Testaufrufen)
Strg-W	^-W	Komponente finden (nur für Zeilen mit Komponentenverweis)
Alt-Hoch	^-⌘-↑	Ausgewählte Zeile(n) nach oben bewegen
Alt-Runter	^-⌘-↓	Ausgewählte Zeile(n) nach unten bewegen
Umschalt-Return	⇧-↩	Zeilenumbruch am Ende der Zeile
Mehrzeilige Textelemente		
Strg-TAB	^-→|	Fokus zum nächsten Attribut bewegen
Umschalt-Strg-TAB	^-⇧-→|	Fokus zum vorhergehenden Attribut bewegen
Strg-Return	⌘-↩	Änderungen bestätigen
Für Prozeduraufruf Knoten		
Strg-P	^-P	Prozedurdefinition finden
Für Knoten mit einem QF-Test ID der Komponente Attribut		
Strg-W	^-W	Komponente finden
Für Testaufruf Knoten		
Strg-P	^-P	Testdefinition finden
Für Bezug auf Abhängigkeit Knoten		
Strg-P	^-P	Abhängigkeitendefinition finden
Protokoll		
Strg-I	^-I	Fehlerliste öffnen
Strg-N	^-N	Nächsten Fehler finden
Umschalt-Strg-N	^-⇧-N	Vorherigen Fehler finden

Strg-T	^-T	Knoten in Testsuite finden
Strg-W	^-W	Nächste Warnung finden
Umschalt-Strg-W	^-⇧-W	Vorherige Warnung finden
Umschalt-Strg-Return	⌘-⇧-↵	Text in externem Editor anzeigen
-	⌘-W	Protokoll schließen
Fortgeschrittenes Editieren		
Strg-7	^-7	Einen Kommentar Knoten einfügen
Umschalt-Strg-7	^-⇧-7	Einen Kommentar Knoten oberhalb des aktuellen Knotens einfügen
Strg-A	^-A	Einen Prozeduraufruf Knoten einfügen
Strg-D	^-D	Einen Knoten zu den Lesezeichen hinzufügen
Umschalt-Strg-D	^-⇧-D	Selektierte Knoten ein-/ausschalten
Strg-I	^-I	Referenzen von Knoten finden (für Komponenten, Prozeduren, Tests und Abhängigkeiten)
Umschalt-Strg-I	^-⇧-I	Selektierte Knoten in eine If-Sequenz einpacken
Umschalt-Strg-P	^-⇧-P	Selektierte Knoten in Prozeduren konvertieren (gilt nur für Sequenzen u.ä.)
Umschalt-Strg-S	^-⇧-S	Selektierte Knoten in eine Sequenz einpacken
Umschalt-Strg-T	^-⇧-T	Selektierte Knoten in einen Testschritt einpacken
Umschalt-Strg-Y	^-⇧-Y	Selektierte Knoten in ein Try/Catch einpacken

Tabelle C.1: Tastaturkürzel für Navigation und Editieren

C.2 UI-Inspektor

Folgende Tabelle beinhaltet die Tastaturkürzel für den UI-Inspektor(108). Sie sind direkt im SUT anzuwenden. Die Standardtastaturkürzel können über Optionen geändert werden, siehe UI-Inspektor-Optionen(575).

Windows/Linux	*macOS*	*Funktion*
UI-Inspektor		
Umschalt-Strg-F11 (konfigurierbar)	^⇧-F11 (konfigurierbar)	Öffnen des UI-Inspektors
Umschalt-Strg-F12 (konfigurierbar)	^⇧-F12 (konfigurierbar)	Aktivieren der Komponentenauswahl im UI-Inspektor (wird bei Bedarf auch geöffnet)

Tabelle C.2: Tastaturkürzel für den UI-Inspektor

C.3 Aufnahme- und Wiedergabefunktionen

Folgende Tabelle beinhaltet wichtige Tastaturkürzel für Aufnahme- und Wiedergabefunktionen, von denen manche auch außerhalb von QF-Test anwendbar sind, z.B. direkt im SUT.

Windows/Linux	*macOS*	*Funktion*
Wiedergabe		
Return	⏎	Wiedergabe starten (aktuellen Knoten ausführen)
F9	F9	Pause
Alt-F12 (konfigurierbar)	⌥-F12 (konfigurierbar)	Wiedergabe unterbrechen (”Keine Panik”-Taste)
Aufnahme		
F11 (konfigurierbar)	F11 (konfigurierbar)	”Aufnahmemodus” ein-/ausschalten
Checks aufnehmen		
F12 (konfigurierbar)	F12 (konfigurierbar)	”Check mode” ein-/ausschalten
Linke Maustaste	Primär-Klick	Standard-Check aufnehmen
Rechte Maustaste	Sekundär-Klick	Liste mit Checks anzeigen
Komponenten aufnehmen		
Umschalt-F11 (konfigurierbar)	⇧-F11 (konfigurierbar)	Modus umschalten für ”Einzelne Komponente aufnehmen”
Strg-V	⌘-V	Aufgenommene QF-Test ID der Komponente aus Zwischenablage einfügen
Strg-Backspace	^-⌫	Zu letzter aufgenommener Komponente springen
Strg-F11 (konfigurierbar)	^-F11 (konfigurierbar)	Modus umschalten für ”Mehrere Komponenten aufnehmen”
Prozeduren aufnehmen		
Umschalt-F12 (konfigurierbar)	⇧-F12 (konfigurierbar)	Modus umschalten für Prozedurenaufnahme
Strg-F12 (konfigurierbar)	^-F12 (konfigurierbar)	Modus umschalten für mehrere Prozedurenaufnahmen
Debugger		
F7	F7	Einzelschritt ausführen
F8	F8	Gesamten Knoten ausführen
Strg-F7	^-F7	Bis Knotenende ausführen
Umschalt-F9	⇧-F9	Knoten überspringen
Strg-F9	^-F9	Aus Knoten herausspringen
Strg-F8	⇧-⌘-B	Breakpoint an/aus

Strg-J	^-J	In Protokoll springen
Strg-,	^-,	Ausführung bei selektierten Knoten fortsetzen

Tabelle C.3: Tastaturkürzel für Aufnahme- und Wiedergabefunktionen

C.4 Tastaturhelfer

Die folgende Grafik kann als Hilfsmittel dienen, um die Verwendung der von QF-Test belegten Funktionstasten zu erleichtern. Sie kann ausgedruckt, ausgeschnitten und über dem Bereich der Tasten F5 bis F12 Ihrer Tastatur angebracht werden.

Abbildung C.1: Tastaturhelfer

Anhang D

Glossar

API
Application Programming Interface, eine Reihe von Paket-, Klassen- und Methodendefinitionen, die der Programmierer einer Anwendung verwenden kann. Die Java-API bezieht sich auf die Schnittstelle der standardisierten Java-Klassenbibliothek, die mit jedem JDK ausgeliefert wird.

AWT
Abstract Windowing Toolkit, der Teil der Java-Bibliothek, der für die Darstellung von Fenstern und Komponenten, sowie die Verarbeitung von Events zuständig ist.

GUI
Graphical User Interface, zu deutsch *grafische Benutzeroberfläche*. Schnittstelle zwischen einem Programm und seinem Anwender, typischerweise aus Fenstern aufgebaut, die Komponenten zur Eingabe und Darstellung von Informationen enthalten.

RMI
Remote Method Invocation, Kommunikationsprotokoll/Programmierschnittstelle in Java zum Aufruf einer Methode eines entfernten Objekts.

SUT
System Under Test, die Applikation, die mit Hilfe von QF-Test einem Test unterzogen wird.

VM
Die Java *Virtual Machine* arbeitet Java-Programme in Form von sog. Bytecodes ab. Dadurch wird die Plattformunabhängigkeit und Kompatibilität von Java auf verschiedenen Rechnern und Betriebssystemen gewährleistet. Diese hängen aber, ebenso wie die Performance, stark von der Qualität der verwendeten VM ab.

Anhang E

Datenschutz

Verarbeitung von personenbezogenen und anderen Daten

E.1 Serverdaten für Versionsabfrage

4.3+ Seit QF-Test 4.3 ist eine optionale Funktion integriert, welche beim Start von QF-Test oder auf expliziten Benutzerwunsch über eine verschlüsselte HTTPS Verbindung von www.qftest.com Informationen zu aktuell verfügbaren QF-Test Versionen abfragt. Diese Informationen werden dann lokal mit der verwendeten QF-Test Version verglichen. Steht ein relevantes Update von QF-Test zur Verfügung, so wird der Benutzer über einen Dialog informiert und auf die entsprechenden QFS-Webseiten verwiesen.

Für die Abfrage der verfügbaren neuesten QF-Test Version über den Webserver von QFS werden aus technischen Gründen u.a. folgende Daten von unserem Webspace-Provider erfasst (in sogenannten Serverlogfiles):

- Ihre Internet Protokoll (IP)-Adresse

Zusätzlich werden ggf. folgende Daten bei der Abfrage übermittelt, um die korrekte Update-Version auszuwählen:

- verwendete QF-Test Version
- verwendetes Betriebssystem
- verwendete Sprachversion von QF-Test
- Typ der verwendeten QF-Test Lizenz
- Prüfsumme der verwendeten QF-Test Lizenz

Diese anonymen Daten werden getrennt von Ihren, eventuell zu einem anderen Zeitpunkt angegebenen personenbezogenen Daten gespeichert und lassen so keine Rückschlüsse auf eine bestimmte Person zu. Sie werden zu statistischen Zwecken ausgewertet, um unseren Internetauftritt und unsere Angebote optimieren zu können.

Rechtsgrundlage für die vorübergehende Speicherung der Daten und der Logfiles ist Art. 6 Abs. 1 lit. b sowie Art. 6 Abs. 1 lit. c i.V.m. Art. 32 DSGVO.

Die vorübergehende Speicherung der IP-Adresse durch das System ist notwendig, um eine Auslieferung der Versionsinformation der neuesten QF-Test Version an den Rechner des Nutzers zu ermöglichen. Hierfür muss die IP-Adresse des Nutzers für die Dauer der Sitzung gespeichert bleiben.

Die Daten werden gelöscht, sobald sie für die Erreichung des Zweckes ihrer Erhebung nicht mehr erforderlich sind. Im Falle der Erfassung der Daten zur Bereitstellung der QF-Test Versionsinformation ist dies der Fall, wenn die jeweilige Abfrage beendet ist. Um mögliche Webdienst-Ausfälle bis hin zu Sicherheitslücken aufdecken zu können, werden die Daten 42 Tage gespeichert und danach automatisch gelöscht.

Die Erfassung der Daten zur Bereitstellung der neuesten QF-Test Versionsinformation und die Speicherung der Daten in Logfiles ist für den Betrieb des Webdienstes zwingend erforderlich. Es besteht folglich seitens des Nutzers keine Widerspruchsmöglichkeit. Jedoch kann die Abfrage der neuesten QF-Test Versionsinformation und damit die Nutzung dieses Webdienstes in den Benutzereinstellungen von QF-Test deaktiviert werden (siehe Abschnitt 41.1.10(506)).

E.2 Direkter Versand von Support-Anfragen aus QF-Test heraus

4.5+ QF-Test enthält eine Hilfs-Funktion die es Kunden ermöglicht, direkt aus dem Programm heraus Anfragen an das QF-Test Supportteam zu senden. Wenn die entsprechende Funktion aus dem ”Hilfe”-Menü aufgerufen wird, so wird der Standardbrowser des Kunden gestartet und dort ein verschlüsseltes Web-Formular von www.qftest.com aufgerufen. Alle dort eingetragenen Informationen werden dann als E-Mail an unser Support-Team gesendet.

Aus technischen Gründen werden dabei u.a. folgende Daten von unserem Webspace-Provider erfasst (in sogenannten Serverlogfiles):

- Ihre Internet Protokoll (IP)-Adresse
- Browsertyp und -version
- Webseite, die Sie besuchen

- Datum und Uhrzeit Ihres Zugriffs

Zusätzlich werden ggf. folgende Daten bei der Abfrage übermittelt, um die Qualität der technischen Unterstützung zu erhöhen und die Bearbeitungszeit zu verkürzen:

- verwendete QF-Test Version
- verwendetes Betriebssystem
- verwendete Sprachversion von QF-Test
- Hinterlegter Name und Firma der verwendeten QF-Test Lizenz
- Prüfsumme der verwendeten QF-Test Lizenz

Rechtsgrundlage für die vorübergehende Speicherung der Daten und der Logfiles ist Art. 6 Abs. 1 lit. b sowie Art. 6 Abs. 1 lit. c i.V.m. Art. 32 DSGVO.

Die vorübergehende Speicherung der IP-Adresse durch das System ist notwendig, um die Übertragung von Support-Anfragen zu ermöglichen. Hierfür muss die IP-Adresse des Nutzers für die Dauer der Sitzung gespeichert bleiben.

Die Daten werden gelöscht, sobald sie für die Erreichung des Zweckes ihrer Erhebung nicht mehr erforderlich sind. Im Falle der Erfassung der Daten zur Bereitstellung von technischer Unterstützung ist dies der Fall, wenn die Geschäftbeziehung zum Kunden beendet ist. Um mögliche Webdienst-Ausfälle bis hin zu Sicherheitslücken aufdecken zu können, werden die Daten zusätzlich auf dem Web-Server 42 Tage gespeichert und danach automatisch gelöscht.

Die Erfassung der Daten zur Bereitstellung von technischer Unterstützung und die Speicherung der Daten in Logfiles ist für den Betrieb des Webdienstes zwingend erforderlich. Es besteht folglich seitens des Nutzers keine Widerspruchsmöglichkeit. Jedoch kann von Benutzer auch unabhängig vom Web-Formular direkt eine Support-Anfrage an support@qftest.com gesendet werden.

E.3 Kontext-Informationen für Online-Handbuch

7.1+ Seit QF-Test 7.1 leitet QF-Test bei der Anfrage von Handbuch- und Tutorial-Daten automatisch auf die Online-Version weiter, wenn die lokale Version nicht verfügbar ist. Zur Verknüpfung von kontextabhängiger Hilfe ist die Datei `doc/context/de/context.properties` bzw. `doc/context/en/context.properties` notwendig. Wenn diese nicht existiert (und nur dann), wird vom QFS-Server eine Online-Version abgerufen. Dabei werden aus technischen Gründen u.a. folgende Daten von unserem Webspace-Provider erfasst (in sogenannten Serverlogfiles):

- Ihre Internet Protokoll (IP)-Adresse

Weitere Informationen dazu wurden bereits im Abschnitt ”Serverdaten für Versionsabfrage” aufgeführt.

E.4 Anfragedaten beim Abruf von WebDriver-Dateien

4.5.2+ Seit QF-Test 4.5.2 wird für einige Browser (z.B. Chrome) der benötigte WebDriver automatisch aus dem Internet geladen und entpackt. Dazu werden die benötigten Dateien von den Servern `www.qftest.com`, `github.com`, `chromedriver.storage.googleapis.com` und `developer.microsoft.com` bzw. deren Datei-Server verschlüsselt abgerufen.

Aus technischen Gründen werden diese Informationen übermittelt:

- Ihre Internet Protokoll (IP)-Adresse
- Der Name der Datei, die heruntergeladen werden soll. Dieser enthält die Versionsnummer des Treibers.

Bitte entnehmen Sie die Details zur weiteren Datenverarbeitung auf den Webservern den Datenschutzerklärungen der vorgenannten Dienstleister. Um den automatischen Treiber-Download zu deaktiveren spezifizieren Sie beim entsprechenden ”Web-Engine starten”-Knoten explizit den korrekten WebDriver als Wert des Java-Properties `qftest.web.webdriver.driver`.

E.5 Client-Daten in QF-Test Protokolldateien

Hinweis Die im Folgenden aufgeführten Protokolle können durch den Benutzer dem QFS-Support z.B. mittels E-Mail oder Datei-Upload zur Supportunterstützung bereitgestellt werden. Es erfolgt keine automatische Übermittlung der Protokolle durch QF-Test an QFS oder dritte.

QF-Test Protokolle sind entscheidende Hilfsmittel für die Analyse von automatisierten Testläufen zur Auffindung von Fehlern im Testablauf oder der zu testenden Anwendung. Neben den Details zur Systemumgebung und den durchgeführten Ausführungsschritten können im Protokoll auch textuelle Ausgaben des zu testenden Systems sowie Bildschirmabbilder der am System angeschlossenen Bildschirme oder virtuellen Desktops sowie der HTML-Sourcecode der Seite, die während eines Tests im Browser angezeigt wird, gespeichert werden.

Art und Umfang der Daten, die in Protokollen gespeichert werden sollen, kann detailliert in den Protokolleinstellungen (siehe Abschnitt 41.11(578)) definiert werden. Dazu gehört auch die Anzahl der automatisch abgespeicherten Protokolle, die vorgehalten werden soll - ältere Protokolle werden automatisch gelöscht.

Das Standardverzeichnis, in dem Protokolle abgelegt werden, ist betriebssystemspezifisch. Der aktuelle Wert kann in QF-Test über das Menü Help→Info→Systeminfo unter dem Punkt `dir.runlog` ermittelt werden. Der Speicherort kann aktiv mittels des Kommandozeilenparameters `-runlogdir <Verzeichnis>`(989) definiert werden

Neben Protokollen zum Testlauf erzeugt QF-Test auch interne Protokolle zu Diagnosezwecken für QF-Test selbst und der Verbindung zum zu testenden System.

Die interne Protokollierung von QF-Test erfolgt rotierend in fünf Textdateien mit Namen `qftestN.log` mit `N = [,1,2,3,4]` und die für die Verbindung zum SUT in der Textdatei `qfconnect.log`.

Das Standardverzeichnis für interne Protokolle ist betriebssystemspezifisch. Der aktuelle Wert kann in QF-Test über das Menü Help→Info→Systeminfo unter dem Punkt `dir.log` ermittelt werden. Der Speicherort kann aktiv mittels des Kommandozeilenparameters `-logdir <Verzeichnis>`(983) definiert werden. Ein ungültiger Verzeichniswert führt zu einer Unterdrückung der internen Protokollierung.

An dieser Stelle möchten wir nochmals explizit abgrenzen, dass der Einsatz von QF-Test originär auf das automatisierte Testen von Software ausgerichtet ist. Hierbei ist immer vorausgesetzt, dass per se die Softwaretests unserer Kunden auf getrennten Testsystemen mit speziellen Testdaten erfolgen, da insbesondere die Verwendung von personenbezogenen Echtdaten aus datenschutzrechtlicher Sicht nicht zulässig ist. Dies resultiert aus den Grundsätzen der Zweckbindung, Datenminimierung sowie Anonymisierung.

Anhang F

Benutzte Software

Die folgende Software wird von QF-Test eingesetzt:

Apache Batik SVG Toolkit
Copyright © 2016 The Apache Software Foundation
Steht unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
URL: `https://xmlgraphics.apache.org/batik`

Apache Commons IO
Copyright © 2009 The Apache Software Foundation
Steht unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
URL: `http://commons.apache.org/proper/commons-io`

Apache Commons Imaging
Copyright © 2024 The Apache Software Foundation
Steht unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
URL: `https://commons.apache.org/proper/commons-imaging`

Apache PDFBox
Copyright © 2009-2017 The Apache Software Foundation
Steht unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
URL: `https://pdfbox.apache.org/`

Apache POI
Copyright © 2009 The Apache Software Foundation
Steht unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0` und `doc/licenses/poi-notice`).
URL: `http://poi.apache.org`

API Guardian
Copyright © API Guardian Team
Steht unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
URL: `https://github.com/apiguardian-team/apiguardian`

ASM
Copyright © 2000-2011 INRIA, France Telecom
Steht unter einer individuellen Lizenz (siehe `doc/licenses/asm`).
URL: `http://asm.ow2.org`

axe-core
Steht unter der Mozilla Public License 2 (siehe `doc/licenses/MPL-2.html`), seine Abhängigkeiten unter der MIT License oder der ISC License (siehe `doc/licenses/axe-core-3rd-party.txt`)
URL: `https://github.com/dequelabs/axe-core`

Bean Scripting Framework (BSF)
Copyright © 1999 International Business Machines Corporation.
Siehe Lizenz `doc/licenses/bsf`
URL: `http://www.alphaworks.ibm.com/tech/bsf`

BasE91
Copyright © 2000-2006 Joachim Henke, 2011 Benedikt Waldvogel
Siehe Lizenz `doc/licenses/base91`
URL: `https://github.com/bwaldvogel/base91`

Bouncy Castle
Copyright © 2000 - 2017 The Legion of the Bouncy Castle Inc.
Ist Bestandteil von Apache PDFBox.
Steht unter der Bouncy Castle Lizenz (analog MIT) (siehe `doc/licenses/bouncy-castle.txt`).
URL: `https://www.bouncycastle.org/java.html`

Boxicons
Stehen unter der Crative Commons Attribution 4.0 International Public License (siehe `doc/licenses/CC4.txt`).
URL: `https://boxicons.com/`

cdp4j
Copyright © 2023 cdp4j Contributors
Steht unter der MIT Lizenz (MIT) (siehe `doc/licenses/license.cdp4j`).
URL: `https://github.com/cdp4j/cdp4j`

Closure Compiler
Steht unter der Apache License, Version 2.0 (siehe

doc/licenses/apache-2.0).
URL: https://github.com/google/closure-compiler

Chromedriver
Steht unter der Modified BSD License (see doc/licenses/chromium-bsd.txt).
URL: https://sites.google.com/a/chromium.org/chromedriver

Chromium Embedded Framework
Steht unter der the Modified BSD License (see doc/licenses/chromium-bsd.txt).
URL: https://bitbucket.org/chromiumembedded/cef

ClasspathSuite
Steht unter der Apache License, Version 2.0 (siehe doc/licenses/apache-2.0).
URL: https://github.com/takari/takari-cpsuite

CommonJS Modules Support for Nashorn
Steht unter der MIT Lizenz (MIT) (siehe doc/licenses/nashorn-commonjs-modules).
URL: https://github.com/coveo/nashorn-commonjs-modules

Cryptix library
Copyright © 1995, 1996, 1997, 1998, 1999, 2000 The Cryptix Foundation Limited. All rights reserved.
Siehe Lizenz doc/licenses/cryptix
URL: http://www.cryptix.org

CSS Parser
Steht unter der Apache License, Version 2.0 (siehe doc/licenses/apache-2.0).
URL: https://sourceforge.net/projects/cssparser/

dd-plist
Steht unter der MIT Lizenz (MIT) (siehe doc/licenses/dd-plist.txt).
URL: https://github.com/3breadt/dd-plist

dom4j
Copyright © 2001-2005 MetaStuff, Ltd. All Rights Reserved
Siehe Lizenz doc/licenses/dom4j
URL: dom4j.sourceforge.net

elasticlunr.js
Copyright © 2017 Wei Song

Steht unter der MIT Lizenz (MIT) (siehe doc/licenses/elasticlunr).
URL: https://elasticlunr.com

Geckodriver
Steht unter der Mozilla Public License 2 (siehe doc/licenses/MPL-2.html).
URL: https://github.com/mozilla/geckodriver

Ghostdriver
Steht unter einer BSD License (siehe doc/licenses/ghostdriver-bsd.txt).
URL: https://github.com/detro/ghostdriver

Groovy Skriptsprache
Steht unter der Apache License, Version 2.0 (siehe doc/licenses/apache-2.0).
URL: http://groovy-lang.org/

InfluxDB 2.0 Java Client Library
Copyright © 2018 Influxdata, Inc.
Steht unter der MIT Lizenz (MIT) (siehe doc/licenses/influxdb-client-java.txt).
URL: https://github.com/influxdata/influxdb-client-java

ini4j
Steht unter der Apache License, Version 2.0 (siehe doc/licenses/apache-2.0).
URL: http://ini4j.sourceforge.net/

Jackson
Copyright © 2020 FasterXML.
Steht unter der Apache License, Version 2.0 (siehe doc/licenses/apache-2.0).
URL: https://github.com/FasterXML/jackson

JCommon
Copyright © 2007-2012 Object Refinery Limited and Contributors.
Steht unter der GNU Lesser General Public License (siehe doc/licenses/LGPL).
URL: http://www.jfree.org/jcommon/index.html

jEdit Syntax Highlighting
Public Domain
The jEdit 2.2.1 syntax highlighting package contains code that is Copyright © 1998-1999 Slava Pestov, Artur Biesiadowski, Clancy Malcolm, Jonathan Revusky, Juha Lindfors and Mike Dillon.
URL: http://syntax.jedit.org

JExcel

Copyright © 2002 Andrew Khan. All rights reserved.
Steht unter der GNU Lesser General Public License (siehe `doc/licenses/LGPL`).
Sourcecode siehe `misc/jexcelapi.tar.gz`
URL: `http://www.andykhan.com/jexcelapi/index.html`

JFreeChart

Copyright © 2005-2012 Object Refinery Limited and Contributors.
Steht unter der GNU Lesser General Public License (siehe `doc/licenses/LGPL`).
URL: `http://www.jfree.org/jfreechart/index.html`

JGoodies Looks

Copyright © 2001-2005 JGoodies Karsten Lentzsch. All rights reserved.
Siehe Lizenz `doc/licenses/jgoodies-looks`
URL: `http://www.jgoodies.com`

JIDE Common Layer

Copyright © 2002 - 2009 JIDE Software, Inc, all rights reserved.
Steht unter der GNU General Public License (siehe `doc/licenses/GPL`) mit Classpath Exception (siehe `doc/licenses/Classpath`).
URL: `http://www.jidesoft.com/products/oss.htm`

JNIWrapper

Copyright © 2000-2004 MIIK Ltd. All rights reserved.
URL: `http://www.jniwrapper.com`

JSch

Copyright © 2002-2015 Atsuhiko Yamanaka, JCraft,Inc. All Rights Reserved.
Siehe Lizenz `doc/licenses/jsch`
URL: `https://github.com/mwiede/jsch`

JUnit

Steht unter der Eclipse Public License 1.0 (siehe `doc/licenses/EPL-1.0`).
URL: `https://github.com/junit-team/junit4`

Jython Skriptsprache

Copyright © 2007 Python Software Foundation; All Rights Reserved.
Siehe Lizenz `doc/licenses/jython`
URL: `http://www.jython.org`

JZlib

Copyright © 2000-2011 ymnk, JCraft,Inc. All rights reserved.
Siehe Lizenz `doc/licenses/jzlib`
URL: `http://www.jcraft.com/jzlib/`

Hamcrest
Steht unter der BSD License (siehe `doc/licenses/hamcrest-bsd.txt`).
URL: `https://github.com/hamcrest/JavaHamcrest/`

lunr-languages
Copyright © 2017 Mihai Valentin.
Steht unter der Mozilla Public License (siehe `doc/licenses/MPL-1.1.html`).
URL: `https://github.com/MihaiValentin/lunr-languages`

map-set-polyfill
Copyright © 2018 Dmitry Vibe
Steht unter der MIT-Lizenz (MIT) (siehe `doc/licenses/mit`).
URL: `https://github.com/Riim/map-set-polyfill`

Mozilla
Copyright © 1998-2006 Contributors to the Mozilla codebase.
Steht unter der Mozilla Public License (siehe `doc/licenses/MPL-1.1.html`).
URL: `http://www.mozilla.org/projects/embedding/`

Nashorn scripting language
Copyright © 2010, 2013, Oracle and/or its affiliates.
Steht unter der GNU General Public License version 2 mit Classpath Exception (siehe `doc/licenses/gplv2ce`).
URL: `https://github.com/openjdk/nashorn`

Netty
Copyright © 2009 Red Hat, Inc.
Steht unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
URL: `http://netty.io`

Open Telemetry
Beinhaltet okhttp und kotlin-stdlib. Alle stehen unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
URL: `https://github.com/open-telemetry/opentelemetry-java`

Operadriver
Nutzung der Software in Binärform für Testzwecke gestattet.
URL: `https://github.com/operasoftware/operachromiumdriver`

PngEncoder
Copyright (c) 1999-2003 J. David Eisenberg. All rights reserved.
Steht unter der GNU Lesser General Public License (siehe `doc/licenses/LGPL`).
Sourcecode siehe `misc/PngEncoder.zip`
URL: `http://catcode.com/pngencoder/`

Python Bibliothek (Jython Beigabe)
Copyright © 2001-2008 Python Software Foundation; All rights reserved.
Siehe Lizenz `doc/licenses/python`
URL: `http://www.python.org`

Selenium and Drivers
Stehen unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
URL: `http://www.seleniumhq.org/projects/webdriver`
Indirekt verwendete Bibliotheken stehen teilweise unter anderen Lizenzen, diese sind im `selenium.jar` direkt referenziert.

SendSignal.exe
Nutzung mit Erlaubnis des Autors Louis K. Thomas.
URL:
`http://www.latenighthacking.com/projects/2003/sendSignal/`

Sixlegs PNG Bibliothek
Copyright © 1998, 1999, 2001 Chris Nokleberg
Steht unter der GNU Lesser General Public License (siehe `doc/licenses/LGPL`).
URL: `http://www.sixlegs.com`

Source Code Pro Font
Copyright © 2010-2019 Adobe
Steht unter der SIL Open Font License, Version 1.1. (siehe `doc/licenses/SIL-open-font-license.md`).
URL: `https://github.com/adobe-fonts/source-code-pro`

STAX - Streaming API for XML
Copyright © 2003-2007 The Apache Software Foundation
Steht unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
URL: `http://geronimo.apache.org`

TrueZIP Virtual File System API
Copyright © 2005-2007 Schlichtherle IT Services.
Steht unter der Apache License, Version 2.0 (siehe `doc/licenses/apache-2.0`).
Ehemals: http://truezip.dev.java.net
Neuere Version:
`https://christian-schlichtherle.bitbucket.io/truezip`

UI-Automation
Steht unter der Apache License, Version 2.0 (siehe

doc/licenses/apache-2.0).
URL: https://github.com/mmarquee/ui-automation/

Undertow and abhängige Bibliotheken
Copyright © 2017 Red Hat, Inc.
Stehen unter der Apache License, Version 2.0 (siehe doc/licenses/apache-2.0).
URL: http://undertow.io/

WebDriverAgent
Copyright © 2015-present, Facebook, Inc.
Steht unter einer BSD-Lizenz (siehe ios/ida/LICENSE).
URL: https://github.com/appium/WebDriverAgent

webp-imageio
Copyright © 2018 Luciad, © 2010, Google Inc.
Steht unter der Apache License, Version 2.0 (siehe doc/licenses/apache-2.0) und unter der WebP-Lizenz (siehe doc/licenses/webp).
URL: https://bitbucket.org/luciad/webp-imageio

wxActiveX - Library for hosting ActiveX controls within wxwidgets programs
Copyright © 2003 Lindsay Mathieson.
Steht unter der wxActiveX Library Licence (siehe doc/licenses/wxActiveX).
URL: http://sourceforge.net/projects/wxactivex

wxMozilla - Embedding Mozilla in wxWindows
Copyright © 2003-2006 Jeremiah Cornelius McCarthy.
Steht unter der wxWindows Library Licence (siehe doc/licenses/wxWidgets).
URL: http://wxmozilla.sourceforge.net/

wxWidgets GUI API
Copyright © 1998-2005 Julian Smart, Robert Roebling et al.
Steht unter der wxWindows Library Licence (siehe doc/licenses/wxWidgets).
URL: http://www.wxwidgets.org/

XML Beans
Steht unter der Apache License, Version 2.0 (siehe doc/licenses/apache-2.0 und doc/licenses/xmlbeans-notice).
URL: http://xmlbeans.apache.org

Zulu OpenJDK 8
Steht unter der GNU General Public License version 2 mit Classpath Exception

(siehe `doc/licenses/gplv2ce`).
URL: `https://www.azul.com/downloads/zulu-community/`