Tutorial

Version 9.0.0

Quality First Software GmbH[1]

Copyright © 2002-2025 Quality First Software GmbH

20. Februar 2025

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

Inhaltsverzeichnis

I Java-GUIs testen mit QF-Test **1**

1 Bearbeiten einer Beispiel-Testsuite (Java) **3**

1.1 Laden der Testsuite 3
1.2 Starten der Anwendung 5
1.3 Ein erster Testfall 7
1.4 Ein zweiter Testfall - mit Fehler 9
1.5 Das Protokoll zur Fehlerdiagnose 10
1.6 Wo finde ich Hilfe? 13
1.7 Beenden der Anwendung 14
1.8 Ein vollständiger Testlauf 15
1.9 Reportgenerierung 15

2 Erstellen einer eigenen Testsuite (Java) **19**

2.1 Starten der Anwendung 19
2.2 Aufnehmen von Aktionen 26
2.3 Aufnahme von Checks 28
2.4 Erstellen einer Testsuite 29
2.5 Beenden der Anwendung 31
2.6 Gesamte Suite ausführen 33

3 Eine Prozedur erstellen (Java) **34**

3.1 Wiederverwendbare Abschnitte identifizieren 34
3.2 Manuelle Erstellung von Prozeduren 35
3.3 Knoten in Prozedur konvertieren 40

4 Komponenten (Java) 42
4.1 Adressierung von Unterelementen von Tabellen, Bäumen und Listen . . 42
4.2 Der Bereich Fenster und Komponenten 44
4.3 SmartIDs - direkte Komponentenadressierung 50

5 Benutzen des Debuggers (Java) 54
5.1 Setzen eines Breakpoints 55
5.2 Schrittweise Ausführung 56
5.3 Knoten überspringen 58
5.4 Debug-Modus bei Fehler oder Exception aktivieren 60
5.5 Fehlerbehebung aus dem Protokoll heraus 62
5.6 Testausführung pausieren 64

6 Variablen und Prozedurparameter (Java) 66
6.1 Prozedur mit Variable 66
6.2 Die Variablendefinitionen-Tabelle 70
6.3 Fortgeschrittenes Debuggen mittels Variablendefinitionen-Tabelle 74
6.4 Variablen setzen 78
6.5 Ebenen für Variablendefinitionen 81

7 Die Standardbibliothek (Java) 83
7.1 Erforschen der Standardbibliothek 84
7.2 Ausgewählte Packages und Prozeduren 85
7.2.1 Das Checkbox Package 85
7.2.2 Das Combobox bzw. Combo Package 86
7.2.3 Das General Package 86
7.2.4 Das List Package 86
7.2.5 Das Menu Package 86
7.2.6 Das Table Package 87
7.2.7 Das Tree Package 87
7.2.8 Das Cleanup Package 87
7.2.9 Das Run-log Package 88
7.2.10 Das Run-log.Screenshots Package 89

7.2.11 Das Shellutils Package 89
7.2.12 Das Utils Package 90
7.2.13 Das Database Package 90
7.2.14 Das Check Package 91
7.2.15 Das Databinder Package 91

8 Ablaufsteuerung (Java) 92
8.1 If - else 92
8.2 Schleifen 95

9 Nun ist es Zeit, Ihre eigene Anwendung zu starten (Java) 104

II Web GUIs testen mit QF-Test 105

10 Bearbeiten einer Beispiel-Testsuite (Web) 107
10.1 Laden der Testsuite 107
10.2 Starten des Browsers 109
10.3 Ein erster Testfall 112
10.4 Ein zweiter Testfall - mit Fehler 113
10.5 Das Protokoll zur Fehlerdiagnose 115
10.6 Wo finde ich Hilfe? 118
10.7 Beenden der Anwendung 119
10.8 Ein vollständiger Testlauf 119
10.9 Reportgenerierung 120

11 Erstellen einer eigenen Testsuite (Web) 123
11.1 Erzeugen der Startsequenz 123
11.2 Aufnehmen von Aktionen 130
11.3 Aufnahme von Checks 132
11.4 Erstellen einer Testsuite 133
11.5 Beenden der Anwendung 135
11.6 Gesamte Suite ausführen 136

12 Eine Prozedur erstellen (Web) 138
12.1 Wiederverwendbare Abschnitte identifizieren 138
12.2 Manuelle Erstellung von Prozeduren 139
12.3 Knoten in Prozedur konvertieren 144

13 Komponenten (Web) 146
13.1 Adressierung von Unterelementen von Tabellen, Bäumen und Listen . . 146
13.2 Web-Komponentenerkennung 148
13.3 Der Bereich Fenster und Komponenten 150
13.4 SmartIDs - direkte Komponentenadressierung 156

14 Benutzen des Debuggers (Web) 160
14.1 Setzen eines Breakpoints 161
14.2 Schrittweise Ausführung 162
14.3 Knoten überspringen 164
14.4 Debug-Modus bei Fehler oder Exception aktivieren 166
14.5 Fehlerbehebung aus dem Protokoll heraus 168
14.6 Testausführung pausieren 170

15 Variablen und Prozedurparameter (Web) 172
15.1 Prozedur mit Variable 172
15.2 Die Variablendefinitionen-Tabelle 176
15.3 Fortgeschrittenes Debuggen mittels Variablendefinitionen-Tabelle 180
15.4 Variablen setzen 184
15.5 Ebenen für Variablendefinitionen 187

16 Die Standardbibliothek (Web) 189
16.1 Erforschen der Standardbibliothek 190
16.2 Ausgewählte Packages und Prozeduren 191
16.2.1 Das Checkbox Package 191
16.2.2 Das Select Package 191
16.2.3 Das General Package 192
16.2.4 Das Table Package 192

16.2.5 Das Cleanup Package 192
16.2.6 Das Run-log Package 193
16.2.7 Das Run-log.Screenshots Package 193
16.2.8 Das Shellutils Package 194
16.2.9 Das Utils Package 194
16.2.10 Das Database Package 195
16.2.11 Das Check Package 195
16.2.12 Das Databinder Package 196

17 Ablaufsteuerung (Web) 197
17.1 If - else 197
17.2 Schleifen 200

18 Nun ist es Zeit, Ihre eigene Anwendung zu starten (Web) 209

III Native Windows GUIs testen mit QF-Test 210

19 Bearbeiten einer Beispiel-Testsuite (Win) 212
19.1 Laden der Testsuite 212
19.2 Starten der Anwendung 214
19.3 Ein erster Testfall 216
19.4 Ein zweiter Testfall - mit Fehler 218
19.5 Das Protokoll zur Fehlerdiagnose 219
19.6 Wo finde ich Hilfe? 222
19.7 Beenden der Anwendung 223
19.8 Ein vollständiger Testlauf 224
19.9 Reportgenerierung 224

20 Erstellen einer eigenen Testsuite (Win) 228
20.1 Starten der Anwendung 228
20.2 Aufnehmen von Aktionen 234
20.3 Aufnahme von Checks 237
20.4 Erstellen einer Testsuite 238

20.5 Beenden der Anwendung 240
20.6 Gesamte Suite ausführen 241

21 Eine Prozedur erstellen (Win) 243
21.1 Wiederverwendbare Abschnitte identifizieren 243
21.2 Manuelle Erstellung von Prozeduren 244
21.3 Knoten in Prozedur konvertieren 249

22 Komponenten (Win) 251
22.1 Adressierung von Unterelementen von Tabellen, Bäumen und Listen . . 251

23 Benutzen des Debuggers (Win) 254
23.1 Setzen eines Breakpoints 255
23.2 Schrittweise Ausführung 256
23.3 Knoten überspringen 258
23.4 Debug-Modus bei Fehler oder Exception aktivieren 260
23.5 Fehlerbehebung aus dem Protokoll heraus 262
23.6 Testausführung pausieren 264

24 Variablen und Prozedurparameter (Win) 266
24.1 Prozedur mit Variable 266
24.2 Die Variablendefinitionen-Tabelle 270
24.3 Fortgeschrittenes Debuggen mittels Variablendefinitionen-Tabelle 274
24.4 Variablen setzen 278
24.5 Ebenen für Variablendefinitionen 281

25 Die Standardbibliothek (Win) 283
25.1 Erforschen der Standardbibliothek 284
25.2 Ausgewählte Packages und Prozeduren 285
25.2.1 Das Run-log Package 285
25.2.2 Das Run-log.Screenshots Package 285
25.2.3 Das Shellutils Package 286
25.2.4 Das Utils Package 286

25.2.5 Das Database Package 287
25.2.6 Das Check Package 287
25.2.7 Das Databinder Package 288

26 Ablaufsteuerung (Win) 289
26.1 If - else 289
26.2 Schleifen 292

27 Nun ist es Zeit, Ihre eigene Anwendung zu starten (Win) 301

IV Mobile-Anwendungen testen mit QF-Test 302

V Weiterführende Features von QF-Test 304

28 Datengetriebenes Testen: Einen Test case mit unterschiedlichen Testdatensätzen starten 306
28.1 Situation 306
28.2 Die traditionelle Methode für datengetriebenes Testen 307
28.3 Datentreiberkonzept 308
28.4 Zusammenfassung 314

29 Abhängigkeiten: Automatisches Sicherstellen der korrekten Vorbedingungen jedes Testfalles 316
29.1 Einführung 316
29.2 Sicherstellen von Vorbedingungen 318
29.3 Verschachtelte Abhängigkeiten 322
29.4 Fehler- und Exceptionbehandlung 328
29.4.1 Fehlerbehandlung 328
29.4.2 Exception Behandlung 329
29.4.3 Zusammenfassung 332
29.5 Mehr zu Abhängigkeiten 332

30 Automatische Erstellung von Basisprozeduren 334

30.1 Einführung 334
30.2 Automatische Erstellung von Prozeduren 337
30.3 Konfiguration der automatischen Erstellung 341
30.3.1 Einführung 341
30.3.2 Erstes Beispiel 342
30.3.3 Den aktuellen Text verwenden 346
30.3.4 Generieren von Container Prozeduren 348
30.3.5 Der aktuelle Wert der Kindkomponente 351
30.3.6 Weitere Konfigurationsmöglichkeiten 354

Vorwort

QF-Test

QF-Test ist ein professionelles Werkzeug zur Automatisierung von Tests für Java-, Web- und nativen Windows-Anwendungen mit einer grafischen Benutzeroberfläche (GUI).

QF-Test testet das System als Ganzes über das GUI. Integrationstests, die das Zusammenspiel von Einzelsystemen überprüfen, können ebenfalls mittels QF-Test implementiert werden. Das Haupteinsatzgebiet sind automatisierte Regressionstests. Aufgrund seiner Eigenschaften kann QF-Test auch zu anderen Zwecken genutzt werden, z.B. um Lasttests durchzuführen oder um Massendaten über eine grafische Oberfläche einzugeben.

QF-Test richtet sich gleichermaßen an Fachtester und Entwickler. Es verfügt über eine einfach zu bedienende Oberfläche, mit der Tests einfach über die Testaufnahmefunktion erstellt werden können. Andererseits können Tests auch ähnlich wie Programme aufgebaut und strukturiert werden. Bei Testfunktionalitäten, die nicht über Standardelemente abgebildet werden können, ist fast immer die Implementierung über Skripte möglich.

QF-Test erstellt während der Testausführung ein spezielles Protokoll, das sehr viele Informationen hinsichtlich einer möglichen Fehleranalyse enthält. Zusätzlich können konfigurierbare HTML-Reports (auch XML oder JUnit Format) erstellt werden, die das Testergebnis übersichtlich und mit Grafiken darstellen.

QF-Test ist plattformunabhängig auf Windows, Linux und macOS einsetzbar. Java-seitig können Swing, JavaFX und SWT getestet werden. Bei den Web-Browsern werden die gängigen Typen unterstützt, teils auch als headless Version. Ab QF-Test Version 5 können auch native Windows-Anwendungen getestet werden. Eine genaue Auflistung der unterstützten Betriebssystem-, Java- und Browser-Versionen finden Sie im Kapitel Systemvoraussetzungen des Handbuchs.

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 Einblick in die Funktionalität von QF-Test.

Tutorial

Dieses Tutorial soll als praktische Einführung in QF-Test dienen.

Der Basisteil stellt die Grundfunktionen von QF-Test vor und leiten Sie durch die notwendigen Schritte, um eine eigene Testsuite zu erstellen. Darüber hinaus lernen Sie Ihre Testergebnisse zu analysieren, mit Hilfe des Debuggers sich schrittweise durch Ihre Tests zu bewegen und einen Überblicksreport zu erstellen. Weitere Themen sind das Konzept der Modularisierung mit Hilfe von Prozeduren und die Komponentenerkennung, die für GUI Tests von zentraler Bedeutung ist.

Die Vorgehensweise zur Erstellung von Tests ist für alle Technologien die gleiche - sei es für Java-, Web- oder native Windows-Anwendungen. Bei Web-Anwendungen muss gegebenenfalls vor Beginn der Testerstellung ein Blick auf die Komponentenerkennung geworfen werden. Bei nativen Windows-Anwendungen benötigt man bei der Aufnahme und dem Abspielen der Tests etwas mehr Geduld. Nun, warum dann drei unterschiedliche Basisteile? Da die Demoapplikationen natürlich für jede Technologie etwas anders aussehen, unterschiedliche Vorbereitungssequenzen erzeugt werden und auch die aufgenommenen Komponenten voneinander abweichen, bezieht sich jeder Basisteil auf die entsprechende Demoapplikation. Somit entsprechen die Beispiele im Tutorial genau dem, was Sie in den Übungen selbst sehen. Bitte wählen Sie Teil I(2) für Java-, Teil II(106) für Web- und Teil III(211) für native Windows-Anwendungen.

Im weiterführenden Teil V(305) stellen wir weitere Funktionalitäten von QF-Test vor: datengetriebenes Testen, das Herstellen der Testvorbedingungen und die automatische Generierung von Basisprozeduren. Diese Techniken sind ebenfalls für alle Applikationstechnologien gleich - und da Sie sich nun bereits besser in QF-Test zurechtfinden, zeigen wir die Beispiele nur für eine Technologie.

Dieses Tutorial ist auch als HTML Online-Version verfügbar unter https://www.qftest.com/de/qftest/tutorial.html.

Als Alternative zum Selbststudium bietet QFS Schulungen für QF-Test an. Details dazu finden Sie unter https://www.qftest.com/de/qftest/training.html im Internet.

Das Tutorial 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.

Feedback

Dieses Tutorial wurde für QF-Test 4.2 von Grund auf überarbeitet. Wir hoffen, dass Sie es in dieser Form hilfreich finden und freuen uns auf Ihr Feedback - positiv wie negativ.

Alle Ihre Kommentare, Fehlerreports, Wünsche etc. senden Sie bitte an support@qftest.com.

Abbildungsverzeichnis

1.1 Das Fenster der Testsuite `ErsteJavaTests.qft` 4
1.2 Der Inhalt des Testfallsatz Knotens 5
1.3 Der Knoten ”Vorbereitung” 5
1.4 Die Sequenz zum Starten des Client 6
1.5 Das CarConfigurator Demo 7
1.6 Der ”Erste” Testfallknoten 7
1.7 Die Details des ersten Testfalls 8
1.8 Die Ergebnisanzeige in der Statusleiste 9
1.9 Der ”Zweite” Testfallknoten 9
1.10 Die Details des zweiten Testfalls 10
1.11 Fehler im zweiten Testfall 10
1.12 Protokoll des zweiten Testfalls 11
1.13 Fehlerdiagnose für den zweiten Testfall 12
1.14 Knoten mit Bildschirmabbild der Fehlersituation 13
1.15 Die Aufräumsequenz 14
1.16 Das Protokoll des gesamten Testfallsatzes 15
1.17 Auswahldialog für die Reportgenerierung 16
1.18 Ein HTML Report 17

2.1 Der Schnellstart-Assistent 20
2.2 Auswählen der SUT Art 21
2.3 Wahl des SUT Programmtyps 22
2.4 Auswahl der Programm Datei 23
2.5 Zusammenfassung 24

2.6 Generierte Startsequenz 25
2.7 Das Fenster des ”CarConfigurator” 26
2.8 Aktionen im CarConfigurator Demo aufnehmen 27
2.9 Der Baum nach Aufnahme der Sequenz 28
2.10 Die umbenannte Sequenz 28
2.11 Die aufgenommene Check-Sequenz 29
2.12 Beginn der Strukturierung 30
2.13 Der Baum nach der Neustrukturierung 31
2.14 Die einfache Aufräumsequenz 32
2.15 Der Protokollbaum der eigenen Testsuite 33

3.1 Zwei identische Testschritte 35
3.2 Prozedurknoten erstellen 36
3.3 Prozedur mit Inhalt befüllen 37
3.4 Prozeduraufruf einfügen 38
3.5 Prozedur auswählen 39
3.6 Testsuite mit Prozedur 40

4.1 Adressierung einer Tabellenzelle 43
4.2 Komponente finden 46
4.3 Komponentenbaum 47
4.4 Details eines Komponente Knoten 48

5.1 Breakpoint setzen 55
5.2 Testlauf starten 56
5.3 Breakpoint löschen 56
5.4 Einzelschritt ausführen 57
5.5 Gesamten Knoten ausführen 57
5.6 Bis Knotenende ausführen 58
5.7 Testausführung am ersten Knoten des zweiten Testfalls pausiert 59
5.8 ”Knoten überspringen” 59
5.9 ”Aus Knoten herausspringen” 60
5.10 Debugger-Optionen: Test bei Fehler anhalten 61

5.11 QF-Test pausiert bei Fehler 61
5.12 Fehlermeldung 62
5.13 Check-Knoten mit erhaltenen Daten aktualisieren 63
5.14 Korrigierter Check-Knoten 64

6.1 Zwei fast gleiche Testschritte 67
6.2 Prozedur mit hartkodiertem Wert 68
6.3 Die Details eines Prozedurknotens 69
6.4 'Check text'-Knoten 70
6.5 Prozeduraufruf von "prüfeEndPreis" in der zweiten Prozedur 71
6.6 Variablendefinitionen 72
6.7 Popup-Menü für "Parameter von Referenzen anpassen" 73
6.8 Variablendefinitionen-Tabelle zeigt den falschen Wert 75
6.9 Ausführung hier fortsetzen 77
6.10 Details des Variable setzen Knoten 79
6.11 Prozedur mit Rückgabewert 80

7.1 Die Standardbibliothek 84

8.1 Setup Sequenz mit If/Elseif Knoten 92
8.2 Warten auf Client setzt die Variable "isSUTRunning" mit dem Ergebnis . . 93
8.3 Der If Knoten wertet die Variable aus 94
8.4 Knoten konvertieren 96
8.5 Knoten einpacken 97
8.6 Details eines Schleife Knotens 99
8.7 Der neue Testfall 101
8.8 Details eines Check Elemente Knoten 102

10.1 Die erste Testsuite 108
10.2 Der Inhalt des Testfallsatz Knotens 109
10.3 Der Knoten "Vorbereitung" 109
10.4 Die Sequenz zum Starten des Browsers 110
10.5 Das CarConfigurator Webdemo 111

10.6 Der ”Erste” Testfallknoten 112
10.7 Die Details des ersten Testfalls 112
10.8 Die Ergebnisanzeige in der Statusleiste 113
10.9 Der ”Zweite” Testfallknoten 114
10.10 Die Details des zweiten Testfalls 114
10.11 Fehler im zweiten Testfall 114
10.12 Protokoll des zweiten Testfalls 116
10.13 Fehlerdiagnose für den zweiten Testfall 117
10.14 Knoten mit Bildschirmabbild der Fehlersituation 117
10.15 Die Aufräumsequenz 119
10.16 Das Protokoll des gesamten Testfallsatzes 120
10.17 Auswahldialog für die Reportgenerierung 121
10.18 Ein HTML Report 122

11.1 Der Schnellstart-Assistent 124
11.2 Auswählen der SUT Art 125
11.3 Auswahl der Programm Datei 126
11.4 Zusammenfassung 127
11.5 Generierte Startsequenz 128
11.6 Das ”CarConfigurator Web” Demo im Browser 129
11.7 Aktionen im ”CarConfigurator Web” Demo aufnehmen 130
11.8 Der Baum nach Aufnahme der Sequenz 131
11.9 Die umbenannte Sequenz 131
11.10 Die aufgenommene Check-Sequenz 132
11.11 Beginn der Strukturierung 134
11.12 Der Baum nach der Neustrukturierung 135
11.13 Die einfache Aufräumsequenz 136
11.14 Der Protokollbaum der eigenen Testsuite 137

12.1 Zwei identische Testschritte 139
12.2 Prozedurknoten erstellen 140
12.3 Prozedur mit Inhalt befüllen 141

12.4 Prozeduraufruf einfügen 142
12.5 Prozedur auswählen 143
12.6 Testsuite mit Prozedur 144

13.1 Adressierung einer Tabellenzelle 147
13.2 Web Resolver Registrierung in der ErsteWebTests.qft 149
13.3 Komponente finden 152
13.4 Komponentenbaum 153
13.5 Details eines Komponente Knoten 154

14.1 Breakpoint setzen 161
14.2 Testlauf starten 162
14.3 Breakpoint löschen 162
14.4 Einzelschritt ausführen 163
14.5 Gesamten Knoten ausführen 163
14.6 Bis Knotenende ausführen 164
14.7 Testausführung am ersten Knoten des zweiten Testfalls pausiert 165
14.8 ”Knoten überspringen” 165
14.9 ”Aus Knoten herausspringen” 166
14.10 Debugger-Optionen: Test bei Fehler anhalten 167
14.11 QF-Test pausiert bei Fehler 167
14.12 Fehlermeldung 168
14.13 Check-Knoten mit erhaltenen Daten aktualisieren 169
14.14 Korrigierter Check-Knoten 170

15.1 Zwei fast gleiche Testschritte 173
15.2 Prozedur mit hartkodiertem Wert 174
15.3 Die Details eines Prozedurknotens 175
15.4 ’Check text’-Knoten 176
15.5 Prozeduraufruf von ”prüfeEndPreis” in der zweiten Prozedur 177
15.6 Variablendefinitionen 178
15.7 Popup-Menü für ”Parameter von Referenzen anpassen” 179
15.8 Variablendefinitionen-Tabelle zeigt den falschen Wert 181

15.9 Ausführung hier fortsetzen 183
15.10 Details des Variable setzen Knoten 185
15.11 Prozedur mit Rückgabewert 186

16.1 Die Standardbibliothek 190

17.1 Setup Sequenz mit If/Elseif Knoten 197
17.2 Warten auf Client setzt die Variable "isSUTRunning" mit dem Ergebnis . . 198
17.3 Der If Knoten wertet die Variable aus 199
17.4 Knoten konvertieren 201
17.5 Knoten einpacken 202
17.6 Details eines Schleife Knotens 204
17.7 Der neue Testfall 206
17.8 Details eines Check Elemente Knoten 207

19.1 Das Fenster der Testsuite `ErsteWinTests.qft` 213
19.2 Der Inhalt des Testfallsatz Knotens 214
19.3 Der Knoten "Vorbereitung" 214
19.4 Die Sequenz zum Starten des Client 215
19.5 Das Windows CarConfigurator Demo 216
19.6 Der "Erste" Testfallknoten 216
19.7 Die Details des ersten Testfalls 217
19.8 Die Ergebnisanzeige in der Statusleiste 218
19.9 Der "Zweite" Testfallknoten 218
19.10 Die Details des zweiten Testfalls 219
19.11 Fehler im zweiten Testfall 219
19.12 Protokoll des zweiten Testfalls 220
19.13 Fehlerdiagnose für den zweiten Testfall 221
19.14 Knoten mit Bildschirmabbild der Fehlersituation 222
19.15 Die Aufräumsequenz 223
19.16 Das Protokoll des gesamten Testfallsatzes 224
19.17 Auswahldialog für die Reportgenerierung 225
19.18 Ein HTML Report 226

20.1 Der Schnellstart-Assistent 229
20.2 Auswählen der SUT Art 230
20.3 Auswahl der Programm Datei 231
20.4 Zusammenfassung 232
20.5 Generierte Startsequenz 233
20.6 Das Fenster des ”CarConfigurator” 234
20.7 Aktionen im CarConfigurator Demo aufnehmen 235
20.8 Der Baum nach Aufnahme der Sequenz 236
20.9 Die umbenannte Sequenz 236
20.10 Die aufgenommene Check-Sequenz 237
20.11 Beginn der Strukturierung 239
20.12 Der Baum nach der Neustrukturierung 240
20.13 Die einfache Aufräumsequenz 241
20.14 Der Protokollbaum der eigenen Testsuite 242

21.1 Zwei identische Testschritte 244
21.2 Prozedurknoten erstellen 245
21.3 Prozedur mit Inhalt befüllen 246
21.4 Prozeduraufruf einfügen 247
21.5 Prozedur auswählen 248
21.6 Testsuite mit Prozedur 249

22.1 Adressierung einer Tabellenzelle 252

23.1 Breakpoint setzen 255
23.2 Testlauf starten 256
23.3 Breakpoint löschen 256
23.4 Einzelschritt ausführen 257
23.5 Gesamten Knoten ausführen 257
23.6 Bis Knotenende ausführen 258
23.7 Testausführung am ersten Knoten des zweiten Testfalls pausiert 259
23.8 ”Knoten überspringen” 259
23.9 ”Aus Knoten herausspringen” 260

23.10 Debugger-Optionen: Test bei Fehler anhalten 261
23.11 QF-Test pausiert bei Fehler 261
23.12 Fehlermeldung 262
23.13 Check-Knoten mit erhaltenen Daten aktualisieren 263
23.14 Korrigierter Check-Knoten 264

24.1 Zwei fast gleiche Testschritte 267
24.2 Prozedur mit hartkodiertem Wert 268
24.3 Die Details eines Prozedurknotens 269
24.4 'Check text'-Knoten 270
24.5 Prozeduraufruf von "prüfeEndPreis" in der zweiten Prozedur 271
24.6 Variablendefinitionen 272
24.7 Popup-Menü für "Parameter von Referenzen anpassen" 273
24.8 Variablendefinitionen-Tabelle zeigt den falschen Wert 275
24.9 Ausführung hier fortsetzen 277
24.10 Details des Variable setzen Knoten 279
24.11 Prozedur mit Rückgabewert 280

25.1 Die Standardbibliothek 284

26.1 Setup Sequenz mit If/Elseif Knoten 289
26.2 Warten auf Client setzt die Variable "isSUTRunning" mit dem Ergebnis . . 290
26.3 Der If Knoten wertet die Variable aus 291
26.4 Knoten konvertieren 293
26.5 Knoten einpacken 294
26.6 Details eines Schleife Knotens 296
26.7 Der neue Testfall 298
26.8 Details eines Check Elemente Knoten 299

28.1 Konventionelle Methode für datengetriebenes Testen 307
28.2 Konventionelle Methode mit einem verschachtelten Testfallsatz 308
28.3 Dialog für eine Datentabelle 309
28.4 Die gefüllte Datentabelle 310

28.5 Testfallsatz mit Datentreiber 310
28.6 Der `$(rabatt)` Parameter 311
28.7 Vollständige Datentabelle 312
28.8 Name für Protokoll und Report Eigenschaft 313
28.9 Protokoll mit unterschiedlichen Namen für Testfälle 314

29.1 Erster Testfallsatz von dependencies_work.qft 317
29.2 Erster Testfallsatz von dependencies_work.qft 318
29.3 Beispiel Testsuite mit der ersten Abhängigkeit 319
29.4 Das Protokoll der Ausführung 319
29.5 Prozedur startStop.starteApplikation 320
29.6 Die Testsuite mit Bezug auf Abhängigkeit 321
29.7 Sicherstellen der Vorbedingungen für Testfall 'Rabattstufe 15' 322
29.8 'Fahrzeugdialog geöffnet' Abhängigkeit 325
29.9 Implementierung der Testfälle 326
29.10 Protokoll von verschachtelten Abhängigkeiten 327
29.11 Testsuite für Fehlerbehandlung 328
29.12 Abhängigkeit mit Fehlerbehandlung 328
29.13 Protokoll einer Abhängigkeit mit Fehlerbehandlung 329
29.14 Try-Catch Knoten in Testfälle 330
29.15 Testsuite mit Catch 331
29.16 Protokoll der Ausführung Abhängigkeit mit Catch 332

30.1 Bildschirmabbild der Testsuite 336
30.2 Die Testsuite automated_procedures_work.qft 337
30.3 Die aufgezeichneten Prozeduren 338
30.4 Die Testsuite mit den Prozeduren 340
30.5 Die Prozeduren für alle Panels 341
30.6 Die aktuelle Konfiguration 342
30.7 Die eigene Konfigurationsdatei 343
30.8 Die `checkText` Prozedur 344
30.9 Die `checkText` Prozedur mit Parametern 344

30.10 Der <COMPID> Platzhalter 345
30.11 Die selbst erstellten Testschritte 346
30.12 Die Konfigurationsdatei mit dem aktuellen Text 347
30.13 Die generierten Prozeduren mit dem aktuellen Text 347
30.14 Die Vorlage für die Containerprozedur 349
30.15 Die Verwendung von @FORCHILDREN 350
30.16 Die generierten Containerprozeduren 351
30.17 Konfiguration mit <CCURRENTVALUE> 352
30.18 Testsuite mit <CCURRENTVALUE> 352
30.19 Parameter für Containerprozeduren 353
30.20 Parameter für die Containerprozedur in der Testsuite 354

Teil I

Java-GUIs testen mit QF-Test

Dieser erste Teil des Tutorials soll Ihnen die Basiseigenschaften und -arbeitsabläufe von QF-Test an Hand einer Java-Anwendung erläutern.

Wenn Sie Web- oder native Windows-Anwendungen testen wollen, empfehlen wir Teil II(106) beziehungsweise Teil III(211). Alle Basisteile vermitteln die gleichen Schulungsinhalte, nutzen für die Beispiele jedoch eine jeweils passende Testanwendung.

Im Teil V(305) werden weiterführende Funktionalitäten von QF-Test erklärt, die für Tests sowohl von Java-, Web- und nativen Windows-Anwendungen genutzt werden können.

Kapitel 1

Bearbeiten einer Beispiel-Testsuite (Java)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Bearbeiten einer Beispiel-Testsuite'
https://www.qftest.com/de/yt/tutorial-1.html

In diesem ersten Kapitel werden wir uns die Struktur einer einfachen Testsuite anschauen, die wesentlichen Bestandteile erklären, sie ausführen und das Ergebnis auswerten.

1.1 Laden der Testsuite

Hinweis

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

Nach dem Starten von QF-Test laden Sie bitte unser erstes Beispiel:

Aktion

- Drücken Sie den Knopf , um den Dateiauswahl-Dialog zu öffnen.
- Wechseln Sie in das Unterverzeichnis `qftest-9.0.0/doc/tutorial` Ihrer QF-Test Installation.
- Dort wählen Sie bitte die Datei `ErsteJavaTests.qft` aus und öffnen diese.

QF-Test präsentiert Ihnen die Testsuite wie im folgenden Bild dargestellt:

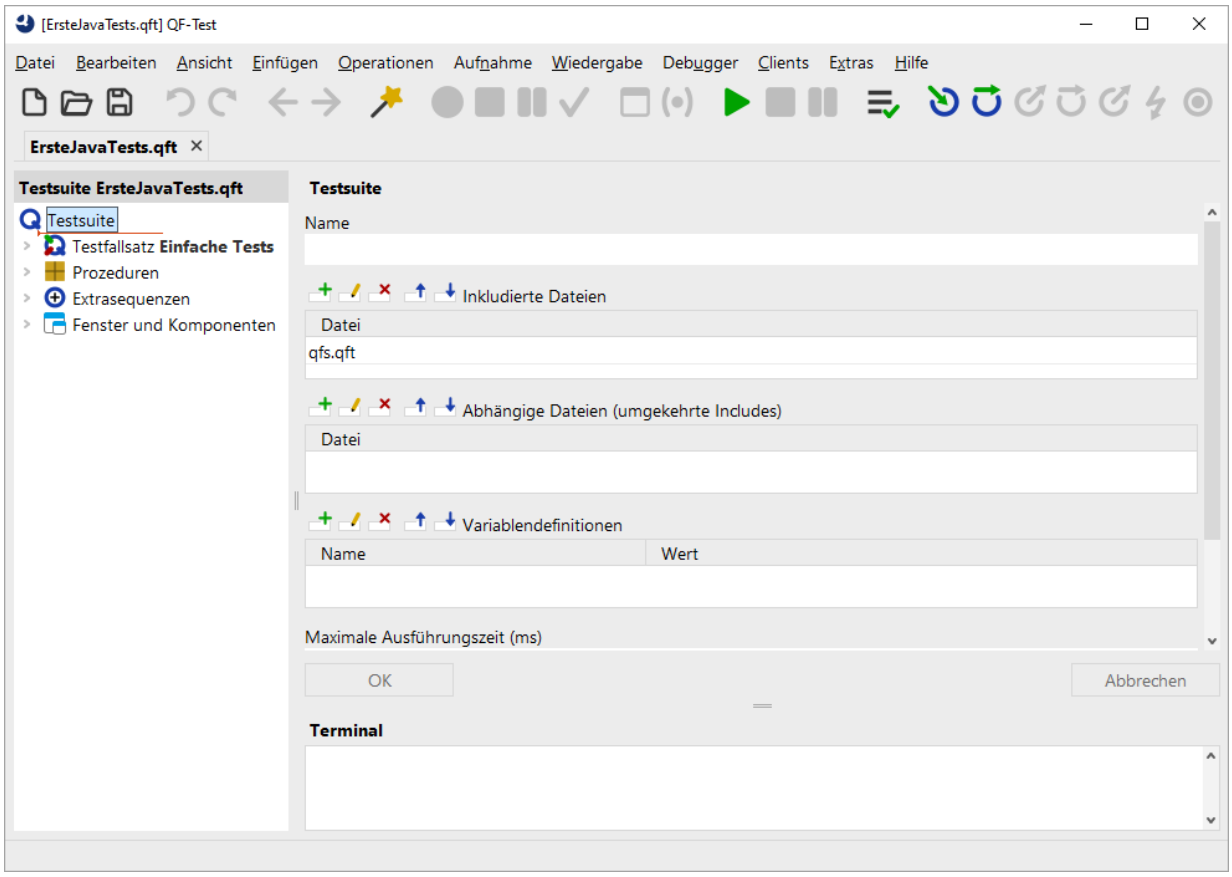

Abbildung 1.1: Das Fenster der Testsuite `ErsteJavaTests.qft`

Der **linke Bereich** des Hauptfensters enthält die Testsuite, die in einer Baumstruktur dargestellt wird.
Rechts befindet sich die Detailansicht des Knotens, der im Baum gerade markiert ist. (Falls die Detailansicht bei Ihnen nicht zu sehen sein sollte, aktivieren Sie diese bitte über das Menü Ansicht→Details anzeigen.)
Im Bereich **unten rechts** befindet sich das Terminal, welches die Ausgaben von QF-Test und dem zu testenden Client protokolliert.

Mit Hilfe des Baumes können Sie durch die Testsuite navigieren und einzelne Knoten auswählen, für die dann jeweils die Details im rechten Fensterbereich eingeblendet werden.

Aktion

- **Doppelklicken** Sie bitte den Knoten **Testfallsatz: Einfache Tests** um ihn zu expandieren und die darin liegenden Knoten sehen zu können.

Der Testfallsatz enthält primär zwei Testfälle, umgeben von einem "Vorbereitung"/"Aufräumen" Knotenpaar, das im Wesentlichen die Testanwendung

startet bzw. beendet.

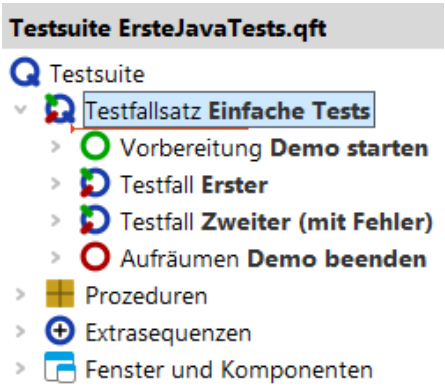

Abbildung 1.2: Der Inhalt des Testfallsatz Knotens

In den folgenden Abschnitten werden wir Funktion und Zweck der einzelnen Knoten erklären.

1.2 Starten der Anwendung

Zuerst wollen wir die Vorbereitung genauer unter die Lupe nehmen:

Aktion

- **Expandieren** Sie den Knoten **Vorbereitung: Demo starten**, wie im folgenden Bild gezeigt.

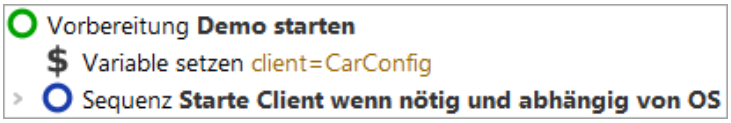

Abbildung 1.3: Der Knoten ”Vorbereitung”

Es werden zwei Kindknoten sichtbar:

1. **Variable setzen** - der Variablen ’client’, wird der Verbindungsname für das zu startende SUT zugewiesen, der für jeden Zugriff auf die Applikation benötigt wird.
2. **Sequenz: Starte Client wenn nötig und abhängig von OS** - startet das zu testende System (SUT) abhängig vom genutzten Betriebssystem, wenn es nicht schon läuft.

Lassen Sie uns noch einen kurzen Blick in die **Sequenz: "Starte Client wenn nötig abhängig von OS"** werfen:

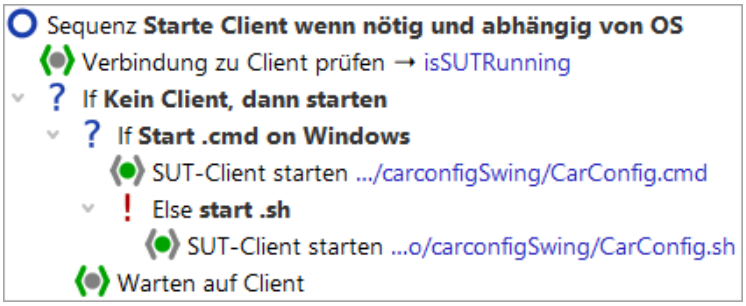

Abbildung 1.4: Die Sequenz zum Starten des Client

Mit QF-Test können Applikationen unter Microsoft Windows, macOS ebenso wie auf Linux Systemen getestet werden. Die hier beschriebene Testsuite kann auf allen drei System abgespielt werden. Die einzige Stelle, an der ein Unterschied zwischen den einzelnen Systemen gemacht werden muss, ist beim Start der Demoanwendung. Mit Hilfe einer **If - Else** Struktur wird für Windows eine `.bat`-Datei und ansonsten eine `.sh` Datei aufgerufen (für macOS und Linux).

Der **SUT-Client Starten** Knoten führt die Anwendung aus und stellt die Verbindung zwischen dem Client und QF-Test her. Um unabhängig vom absoluten Verzeichnispfad der Applikation zu sein, verwenden wir einen relativen Pfad, ausgehend vom QF-Test Installationsverzeichnis, dass über die Variable `${qftest:dir.version}` angegeben wird (siehe Handbuchkapitel Variablen).

Im Kapitel Abschnitt 8.1 (92) wird die Startsequenz ausführlich behandelt. Hier soll nur kurz erwähnt werden, dass ein Starten der Anwendung nur erfolgt, wenn sie nicht bereits läuft.

Wir wollen nun die Anwendung wirklich starten:

Aktion

- **Markieren** Sie dazu bitte den Knoten **Vorbereitung: Demo starten**, doch belassen Sie ihn aufgeklappt.
- **Klicken Sie** den Knopf **Wiedergabe**. Dies führt den aktuellen ausgewählten Knoten aus.

Während der Ausführung wird der gerade aktive Knoten durch "**->**" markiert.

Nach Abschluss der Startsequenz sollte die Demoapplikation "CarConfigurator" am Bildschirm erscheinen. Da QF-Test nach Ende der Wiedergabe den Fokus zurückerhält, kann die Demoapplikation dadurch auch wieder verdeckt worden sein.

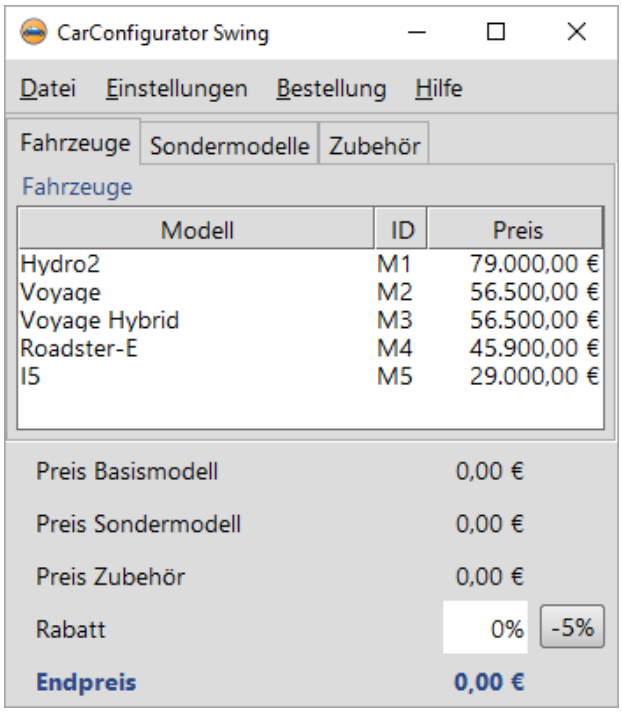

Abbildung 1.5: Das CarConfigurator Demo

1.3 Ein erster Testfall

Als nächstes wollen wir einen Blick auf den ersten Testfall werfen. Er besteht aus vier Testschritten:

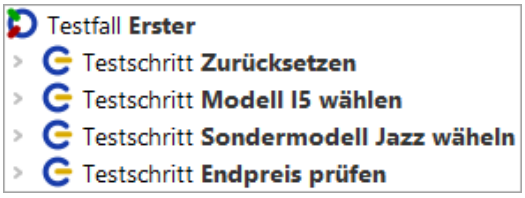

Abbildung 1.6: Der ”Erste” Testfallknoten

1. **Zurücksetzen** - stellt den Anfangszustand der Anwendung über das Menü Datei->Zurücksetzen wieder her und selektiert den Tab Fahrzeuge.
2. **Modell I5 wählen** - Wählt das letzte Modell I5 in der Fahrzeugetabelle aus.
3. **Sondermodell Jazz wählen** - Wechselt zum Tab Sondermodelle und wählt dort Jazz.

4. **Endpreis prüfen** - Überprüft, dass der berechnete Wert dem Feldes Endpreis unten rechts einem vorgegebenen Wert entspricht.

Testschritte sind oft hilfreich, um einen Testfalls zu strukturieren und dadurch lesbar und verständlicher zu gestalten. Dies erleichtert später eine eventuelle Fehlersuche oder Anpassungen des Testfalls.

Aktion

- Bitte **expandieren** Sie die vier **Testschritt Knoten**.

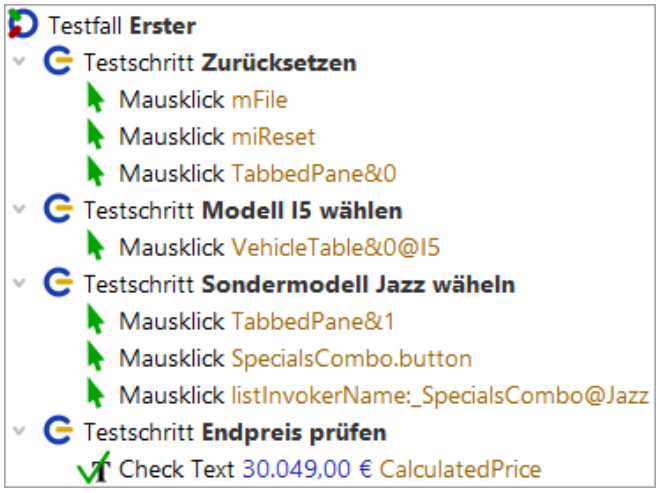

Abbildung 1.7: Die Details des ersten Testfalls

Sie sehen diverse Mausklicks sowie einen Check. Zur besseren Lesbarkeit des Testfalls wurden sie mittels Testschrittknoten strukturiert. Neben der Aktionsart (Mausklick, Check) wird angezeigt, auf welche Anwendungskomponente sich die Aktion bezieht, also wohin z.B. der Mausklick geht. Diese Knoten können direkt über die Aufnahmefunktion von QF-Test erzeugt werden. Näheres hierzu erfahren Sie im nächsten Kapitel Erstellen einer eigenen Testsuite (Java)[19].

Wir wollen uns nun die Ausführung des ersten Testfalls anschauen.

Aktion

- **Markieren** Sie dazu den **Testfall: Erster** Knoten.
- **Drücken Sie** anschließend den Wiedergabeknopf ▶ .

Die Testschritte werden nun der Reihe nach abgespielt, wobei dies typischerweise ziemlich rasch passiert.

Das aktuelle Testergebnis wird während und nach dem Testlauf in der Statuszeile am unteren rechten Rand des QF-Test Hauptfensters angezeigt und sollte ”Beendet: Keine

Fehler" lauten. Daneben zeigt QF-Test verschiedene Zähler an. Der erste Zähler bezieht sich auf die Anzahl der ausgeführten Testfälle, der zweite auf die Zahl der ausgeführten Testfälle ohne Fehler. In unserem Fall wurde ein Testfall fehlerfrei ausgeführt, was einer Erfolgsquote von 100% entspricht.

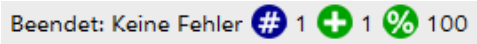

Abbildung 1.8: Die Ergebnisanzeige in der Statusleiste

Wenn Sie den Mauszeiger auf dem Symbol eines Testfallzählers ruhen lassen, wird Ihnen eine entsprechende Beschreibung angezeigt. Eine Auflistung aller Testfallzähler finden Sie im Kapitel Aufnahme und Wiedergabe des Handbuchs.

1.4 Ein zweiter Testfall - mit Fehler

Der zweite Testfall wird uns zeigen, was passiert, wenn ein Fehler bei der Testausführung auftritt.

Aktion

- Bitte **expandieren** Sie den Knoten **Testfall: Zweiter (mit Fehler)**.

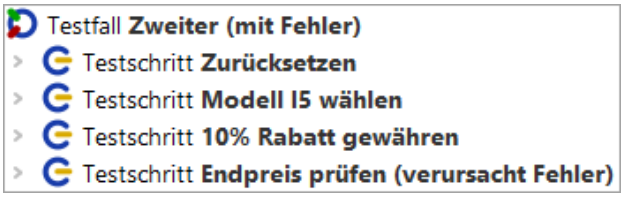

Abbildung 1.9: Der "Zweite" Testfallknoten

Bis auf den dritten Testschritt sieht es bekannt aus. Was tut der Unbekannte?

Testschritt: 10% Rabatt gewähren - Schreibt den Wert 10 in das Rabattfeld

Die Texteingabe ist eine weitere Basisaktion. Eingabe-Knoten kann man ebenfalls direkt über die Aufnahmefunktion generieren lassen. Den Wert 10 sieht man im Feld "Text" rechts und auch direkt im Text des Baumknotens.

Aktion

- **Expandieren** Sie den Knoten **Testschritt: 10% Rabatt gewähren**.

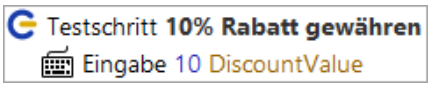

Abbildung 1.10: Die Details des zweiten Testfalls

Wir wollen uns die Ausführung des zweiten Testfalls anschauen.

Aktion

- **Markieren Sie** dazu den **Testfall: Zweiter (mit Fehler)** Knoten.
- **Drücken Sie** anschließend den Wiedergabeknopf ▶ .

Diesmal erscheint ein Dialog mit der Information, dass ein Fehler aufgetreten ist.

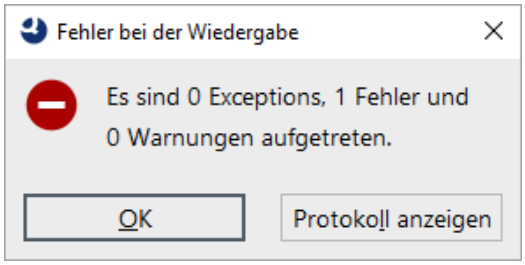

Abbildung 1.11: Fehler im zweiten Testfall

Was ist passiert? Fast immer wenn so ein Fall auftritt, ist es sinnvoll das Protokoll zu Rate zu ziehen.

Alternativ könnte man den Testfall zur Fehlersuche nochmal im Debug-Modus ausführen. Diese Vorgehensweise wird in Kapitel Benutzen des Debuggers (Java)[54] erläutert.

1.5 Das Protokoll zur Fehlerdiagnose

QF-Test protokolliert detaillierte Informationen für jede Testausführung.

Aktion

- **Öffnen** Sie nun bitte das **letzte Protokoll** über eine der folgenden Möglichkeiten:
  - den **Protokoll anzeigen** Knopf im Fehlerdialog

  oder falls Sie den Dialog bereits geschlossen haben

  - den **Button** in der Werkzeugleiste oder
  - über die Tastenkombination **Strg-L**.

Hinweis Die Protokolle der letzten Testläufe können auch über die unteren Einträge im Menü 'Wiedergabe' aufgerufen werden.

Das Protokoll öffnet sich in einem separaten Fenster und zeigt die protokollierten Aktionen des zweiten Testfalls, den Sie soeben ausgeführt haben:

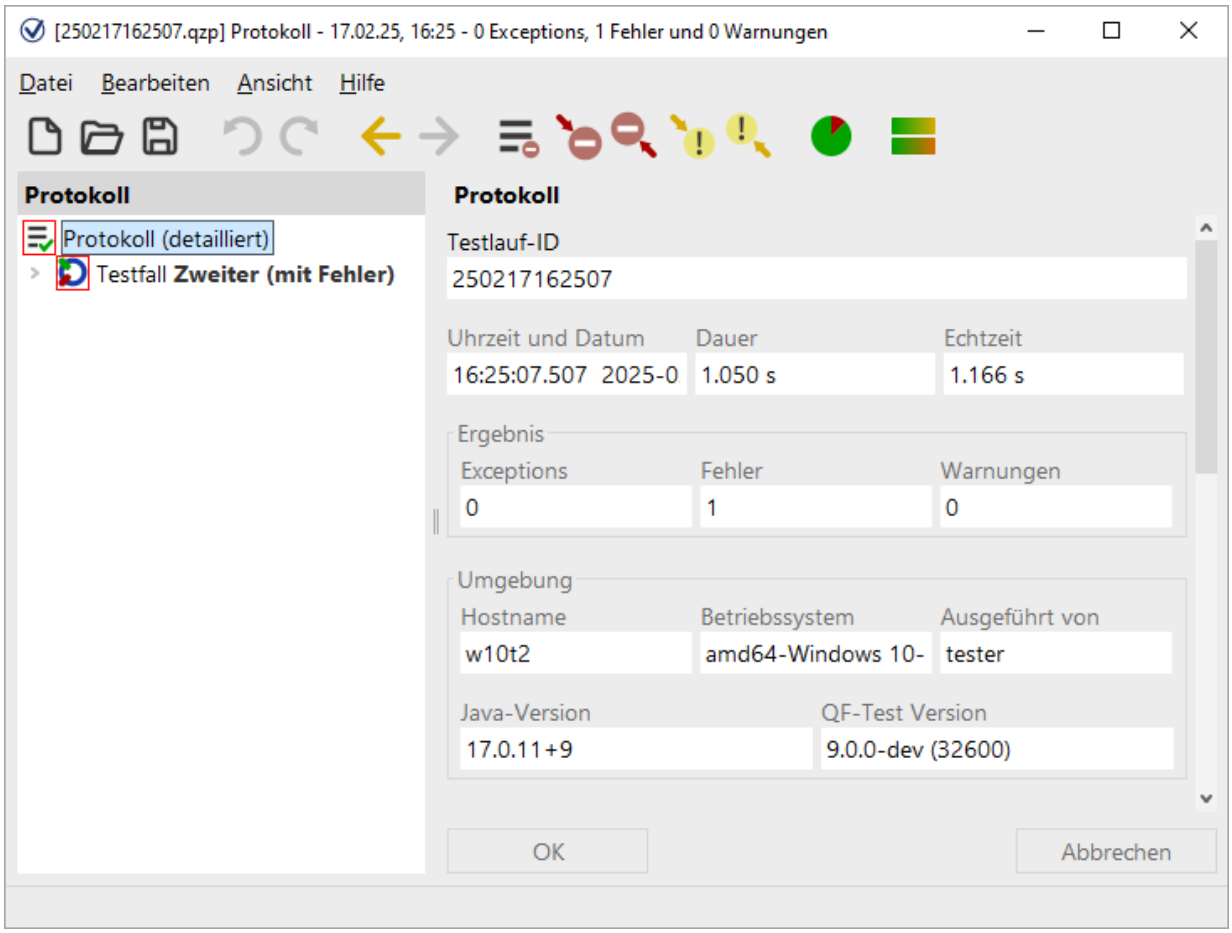

Abbildung 1.12: Protokoll des zweiten Testfalls

Das Protokoll ist in seinem Aufbau ähnlich zu dem der Testsuite. Der Baum links enthält wieder die bekannten Knoten, jedoch dieses Mal in der zeitlichen Abfolge des Testlaufs. Wenn man einen Knoten anwählt, sieht man rechts die Details inklusive Zeitstempel und Ausführungsdauer.

Im Baum links werden Ihnen **rote Rahmen** um einige Knoten auffallen. Diese zeigen an, dass sich darunter Fehler befinden. Wenn man den rot umrandeten Knoten Ebene für Ebene folgt, erreicht man irgendwann den Fehler.

Aktion

- Schneller und bequemer geht es über den Button **Nächsten Fehler finden** in

der Werkzeugleiste oder auch die Tastenkombination **[Strg-N]**.

Alle rot markierten Knoten werden expandiert und der Knoten mit dem eigentlichen Fehler wird selektiert:

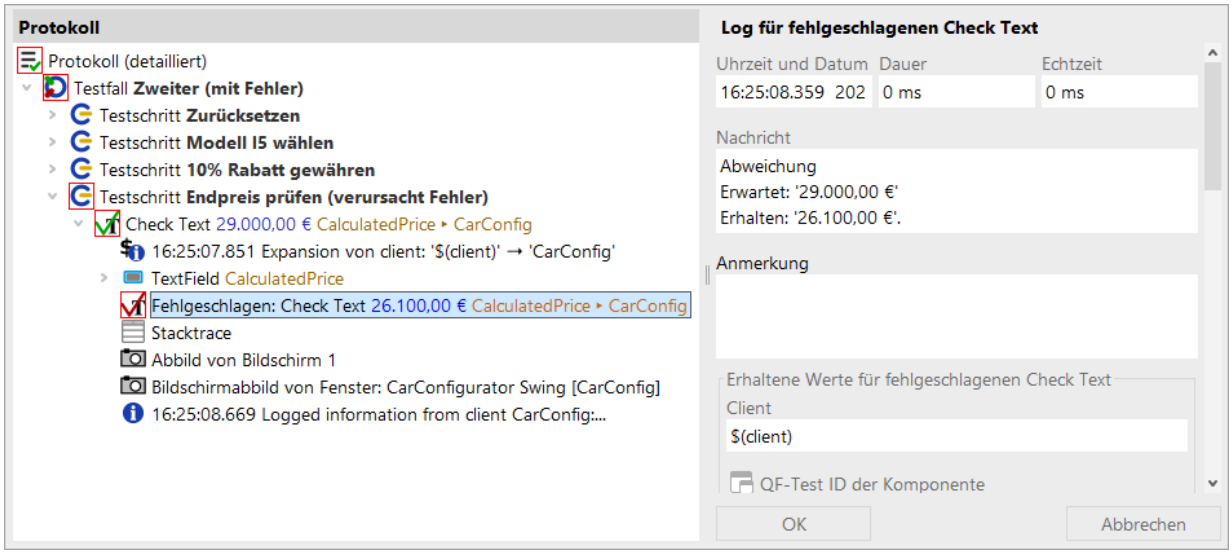

Abbildung 1.13: Fehlerdiagnose für den zweiten Testfall

Die Fehlermeldung auf der rechten Seite gibt an, dass der erhaltene Wert des Endpreis Feldes nicht dem erwarteten entspricht. Dieser Fehler wurde natürlich mit Absicht eingebaut, um zu zeigen, wie man bei der Analyse vorgehen kann.

Hilfreich bei der Fehleranalyse ist üblicherweise auch der übernächste Protokollknoten **Bildschirmabbild**. Seine Detailansicht enthält ein vollständiges Abbild des Bildschirms zum Zeitpunkt des Fehlers. Dies ist sehr nützlich, um den Zustand des SUTs zu sehen und daraus eventuell die Fehlerursache ableiten zu können. Die folgende Grafik zeigt den Knoten:

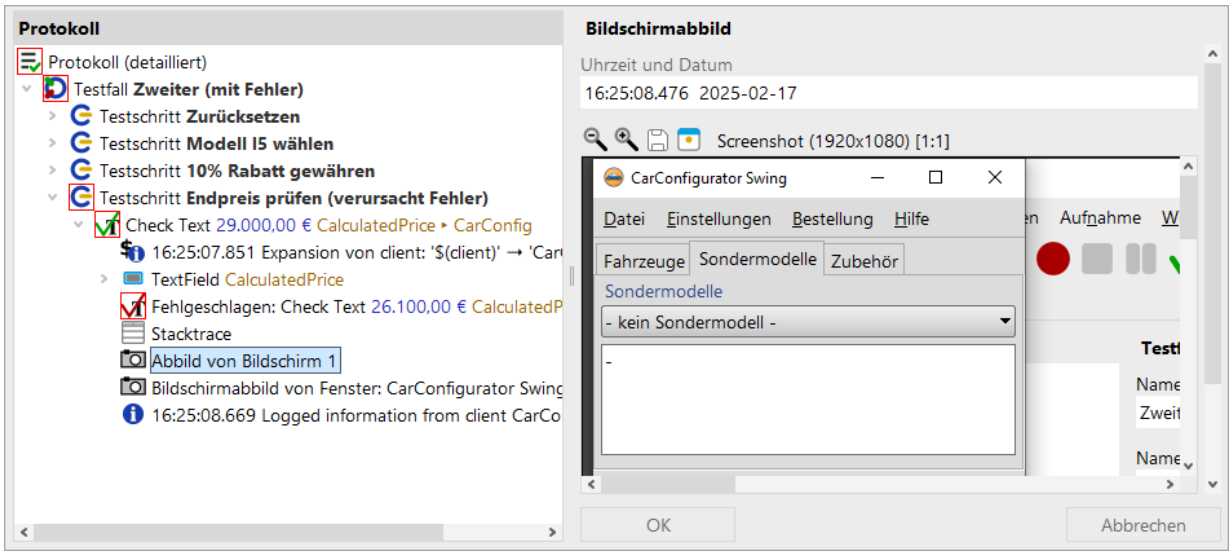

Abbildung 1.14: Knoten mit Bildschirmabbild der Fehlersituation

Neben dem Abbild aller Bildschirme speichert QF-Test auch Bilder der einzelnen Fenster des SUT zum Fehlerzeitpunkt. Dies erlaubt Ihnen deren Inhalt zu analysieren, auch wenn diese eigentlich durch andere Fenster oder Dialoge verdeckt sind.

Hinweis Die in einem längeren Testlauf im Protokoll gesammelten Informationen können große Mengen an Arbeitsspeicher verbrauchen. Deshalb ist QF-Test so voreingestellt, dass es kompakte Protokolle erstellt, wobei nur die für Fehlerdiagnose und Reportgenerierung wichtigen Informationen erhalten bleiben.
Diese Funktion ist mit der Option "Kompakte Protokolle erstellen" über Bearbeiten→Optionen...→Protokoll→Inhalt konfigurierbar. Der Typ eines Protokolls wird in seinem Wurzelknoten angezeigt. Auch die Anzahl der Bildschirmabbilder, die im Protokoll gespeichert werden, ist konfigurierbar.

1.6 Wo finde ich Hilfe?

In diesem Abschnitt machen wir eine kleine Pause, um einige allgemeine Hinweise zu geben.

Es gibt verschiedene Möglichkeiten, um Hilfe oder Antworten zu finden:

Die umfassendste Suche kann man über **Hilfe→Online Suche...** anstoßen. Dies bringt Sie auf die Suchfunktionalität unserer Homepage und erlaubt Ihnen die **Abfrage aller verfügbarer Dokumentation** (Handbuch https://www.qftest.com/qf-test-handbuch.html, Tutorial https://www.qftest.com/qf-test-tutorial.html, Standardbibliothek

https://www.qftest.com/qf-test-support/dokumentation/standardbibliothek.html, Blog https://www.qftest.com/blog.html und unsere Videos https://www.qftest.com/los-gehts-mit-qf-test/videos.html). Die angezeigten Suchergebnisse können **passend gefiltert** werden.

Wenn Sie **offline** arbeiten und nach einem Thema suchen wollen, können Sie die **PDF Versionen von Handbuch und Tutorial** nutzen, die über das **Hilfe** Menü verfügbar sind. Offline HTML Versionen haben keine übergreifende Inhaltssuche. Jedoch gibt es auf jeder HTML Seite in Kopf- und Fußzeile einen Link auf die PDF Version, so dass der Wechsel dorthin einfach möglich ist.

QF-Test bietet eine **kontextsensitive Hilfe** für alle Baumknoten und deren Detailattribute an. Um diese zu nutzen, klicken Sie einfach mit der **rechten Maustaste** auf den gewünschten Knoten oder das Attribut in der Detailansicht. Im Kontextmenü wählen Sie dann den Eintrag **Was ist das?**. Dieser bringt Sie direkt zur passenden Referenzbeschreibung ins Handbuch.

Neben der Hilfestellung in der Dokumentation haben Sie auch die Möglichkeit unser Support-Team zu kontaktieren. Während Ihrer Evaluationsphase und anschließend als Kunde mit einem gültigen Pflegevertrag können Sie Ihre Fragen direkt an unsere Support-Expert:innen richten über das Support-Formular im QF-Test Hilfe-Menü Support-Team kontaktieren oder direkt über unsere Webseite.

1.7 Beenden der Anwendung

Wir haben noch nicht die Aufräumsequenz angeschaut und wollen dies nun tun:

Aktion

- **Expandieren** Sie den **Aufräumen: Demo beenden** Knoten.

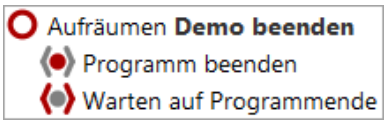

Abbildung 1.15: Die Aufräumsequenz

Unsere Aufräumsequenz stoppt "hart" den Client-Prozess und wartet anschließend, bis sich dieser vollständig beendet hat. Dies ist eine sehr einfache Variante aber für den Moment ausreichend.

Aktion

Führen Sie die **Aufräumsequenz aus** und lassen damit den CarConfigurator verschwinden.

1.8 Ein vollständiger Testlauf

Nachdem wir uns Schritt für Schritt durch den Testfallsatz gearbeitet haben, wollen wir nun alles in einem Rutsch ausführen.

Aktion

- **Schließen Sie bitte das "CarConfigurator" Demo**, falls es noch läuft.
- **Markieren** Sie den **Testfallsatz "Einfache Tests"**.
- Führen Sie ihn aus mittels ▶ .

Der Testlauf endet mit dem bekannten Fehler.

Aktion

- Wenn Sie nun bitte mittels das **Protokoll öffnen**, sehen Sie, wie QF-Test den Test abgearbeitet hat.

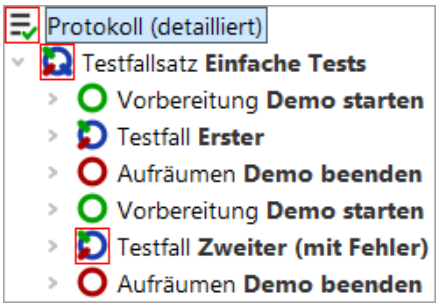

Abbildung 1.16: Das Protokoll des gesamten Testfallsatzes

Man sieht, dass die Vorbereitungs- und Aufräumenknoten vor bzw. nach **jedem Testfall** ausgeführt werden. Dies ist eine Eigenschaft, die diese im Zusammenspiel mit einem Testfallsatzknoten entwickeln. Dadurch wird für jeden Testfall immer ein sauberer Ausgangszustand hergestellt.

Hinweis

Das SUT nach jedem Testfall zu beenden ist nicht die eleganteste Art, einen sauberen Ausgangszustand zu erreichen. Elegantere Wege zur Herstellung einer definierten Testausgangssituation und Durchführung der notwendigen Aufräumarbeiten werden in Kapitel (Kapitel 29(316)) dieses Tutorials erklärt.

1.9 Reportgenerierung

Im Qualitätssicherungsprozess ist es wichtig, Testergebnisse zu dokumentieren und auch zu archivieren. QF-Test bietet die Möglichkeit, aus Protokollen Testreports zu generieren. Wir wollen dies für das gerade aufgezeichnete Protokoll beispielhaft durchführen.

Aktion

- **Öffnen** Sie bitte das **Protokoll** und
- **wählen** im Menü **Datei→Report erstellen...**.

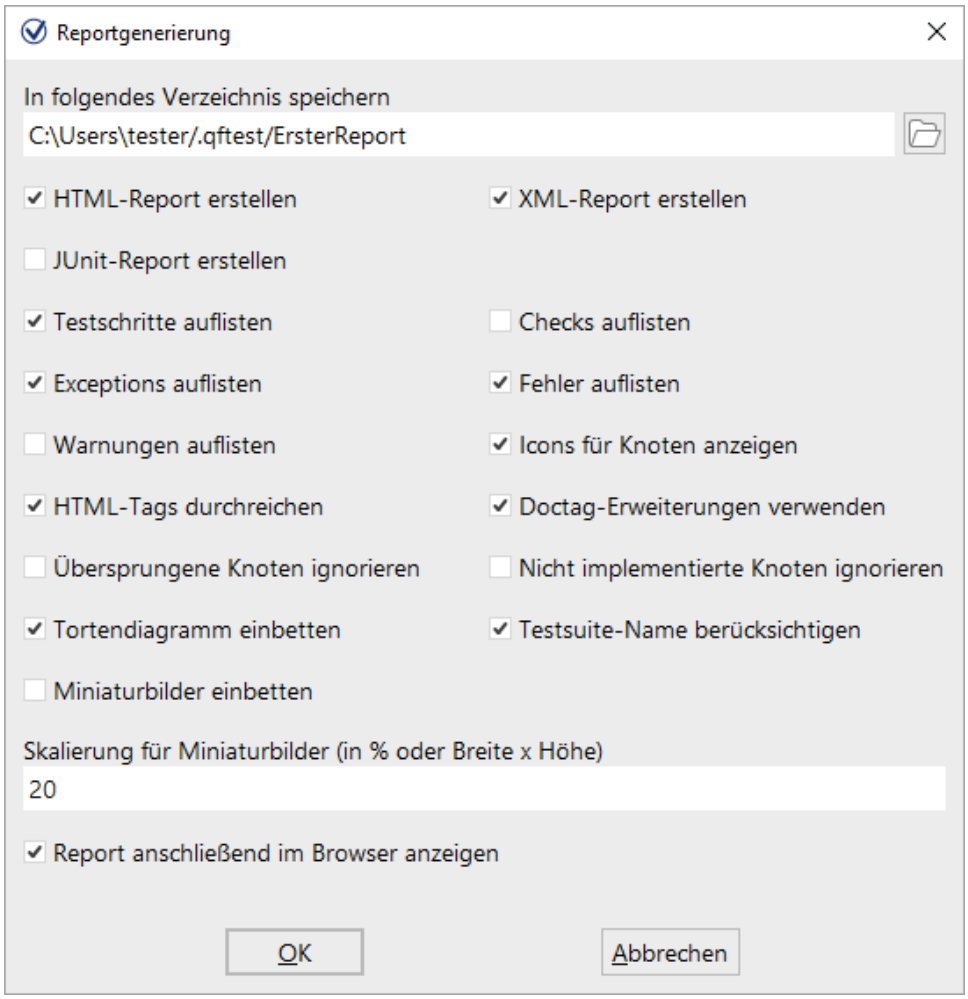

Abbildung 1.17: Auswahldialog für die Reportgenerierung

Im ersten Feld können Sie den Dateinamen des Reports festlegen. QF-Test bietet drei Arten von Reports - HTML, XML und JUnit Format. Das XML Format können Sie verwenden, wenn Sie die Reports zum Beispiel mit Hilfe eigener XSLT Stylesheets selbst gestalten wollen. JUnit-Reports erweisen sich als hilfreich, wenn es darum geht, Resultate in Build- oder Testmanagement-Tools zu importieren.

Wir wollen uns nun einen einfachen HTML-Report zu unserem letzten Testlauf erzeugen lassen.

Aktion

- Lassen Sie bitte die vorgegebenen **Optionen unverändert**.

- Bestätigen Sie den Reportdialog mit **OK**.

Anschließend sollte sich Ihr Browser automatisch mit einem Ergebnis äquivalent zum folgenden Bild öffnen:

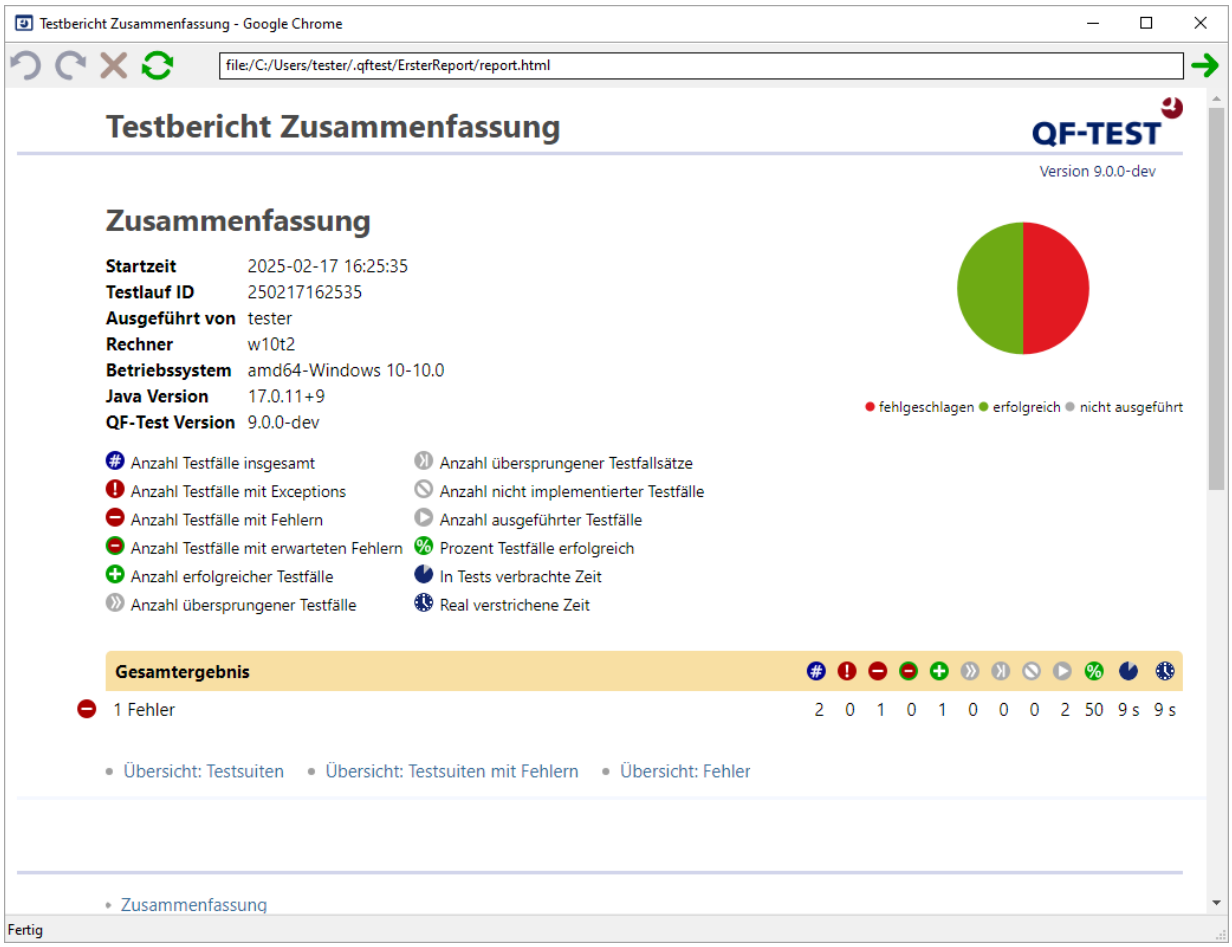

Abbildung 1.18: Ein HTML Report

Der Testbericht beginnt mit einer Zusammenfassung mit allgemeinen Systeminformationen im linken Bereich, einer Legende der verwendeten Symbole rechts, einem Überblicks-Tortendiagramm in der Mitte und dem Gesamtergebnis darunter. In unserem Fall bei einem fehlerhaften von zwei ausgeführten Testfällen eine Erfolgsquote von 50%.

Auf die Zusammenfassung folgen drei Übersichten:

1. Testsuiten, die in diesem Testlauf ausgeführt wurden.
2. Testsuiten, in denen Fehler aufgetreten sind.

3. Fehler, mit Ihrem genauen Ort und Fehlermeldung

Die Reporterstellung in QF-Test ist ein praktisches Hilfsmittel, um einen Überblick über einen Testlauf zu gewinnen und ein Dokument zu Präsentations- und Archivierungszwecken zu erstellen.

Kapitel 2

Erstellen einer eigenen Testsuite (Java)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Erstellen einer eigenen Testsuite'
https://www.qftest.com/de/yt/tutorial-2.html

In diesem zweiten Kapitel des Java-Tutorials werden wir selbst Sequenzen zum Starten und Beenden des SUT erstellen. Zusätzlich wollen wir Aktionen und Checks aufnehmen und damit einen einfachen Testfall aufbauen.

2.1 Starten der Anwendung

Zu Beginn muss die zu testende Anwendung aus QF-Test heraus gestartet werden. Es gibt einen **Schnellstart-Assistenten**, der uns hilft, eine passende Startsequenz zu erzeugen.

Aktion

- Öffnen Sie bitte eine neue, leere Testsuite mittels Datei→Neue Testsuite...
- Öffnen Sie den Schnellstart-Assistenten über das **Menü** **Extras→Schnellstart-Assistent...**.

Der Assistent startet mit einem Willkommen und allgemeinen Informationen.

Aktion

- Nach einem kurzen Hallo drücken Sie bitte den **"Weiter" Knopf**.

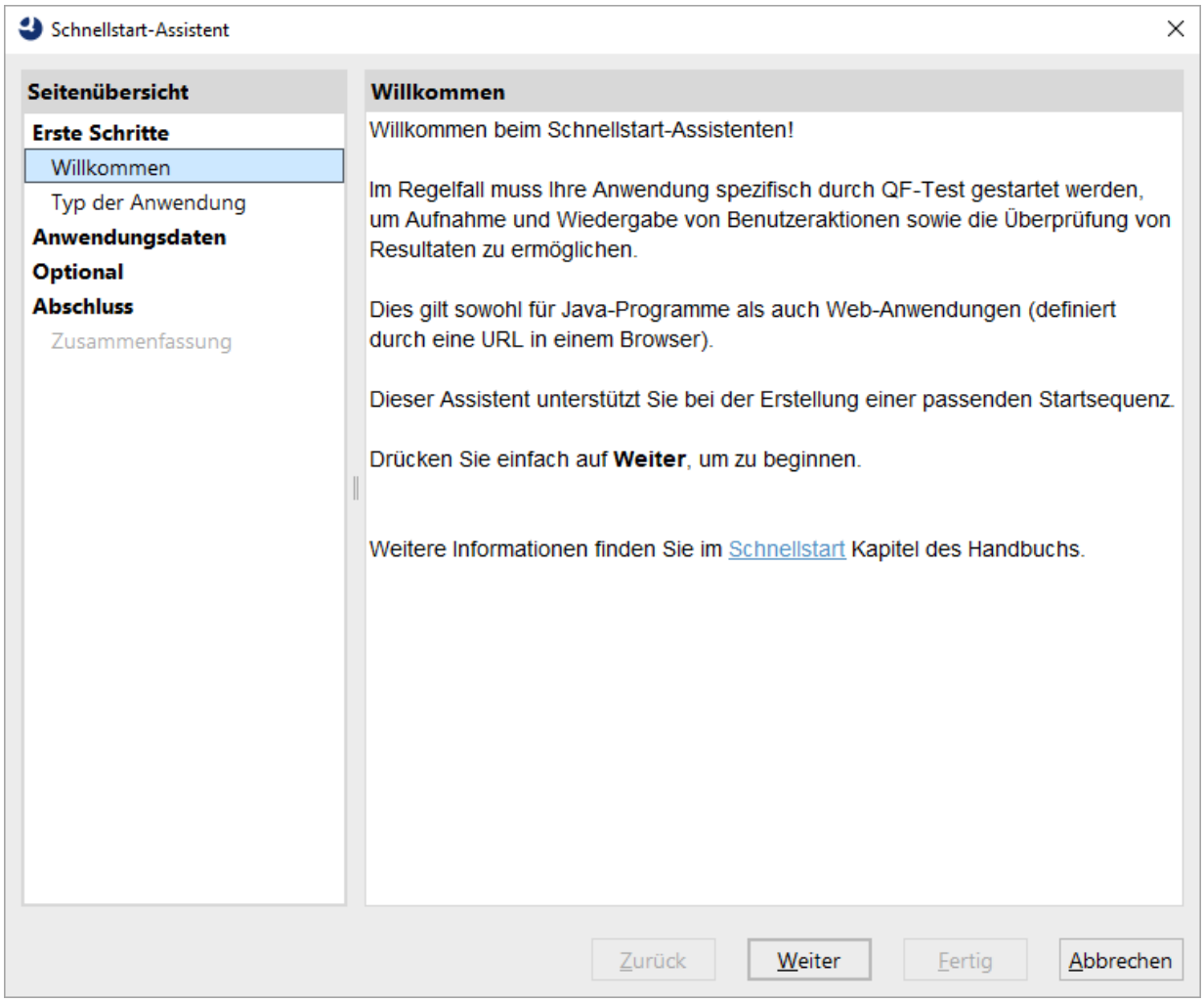

Abbildung 2.1: Der Schnellstart-Assistent

Im zweiten Schritt werden Sie dazu aufgefordert, die Art der zu testenden Applikation auszuwählen.

Aktion

- Behalten Sie bitte die erste Option **Eine Java-Anwendung**.
- Drücken Sie **Weiter**.

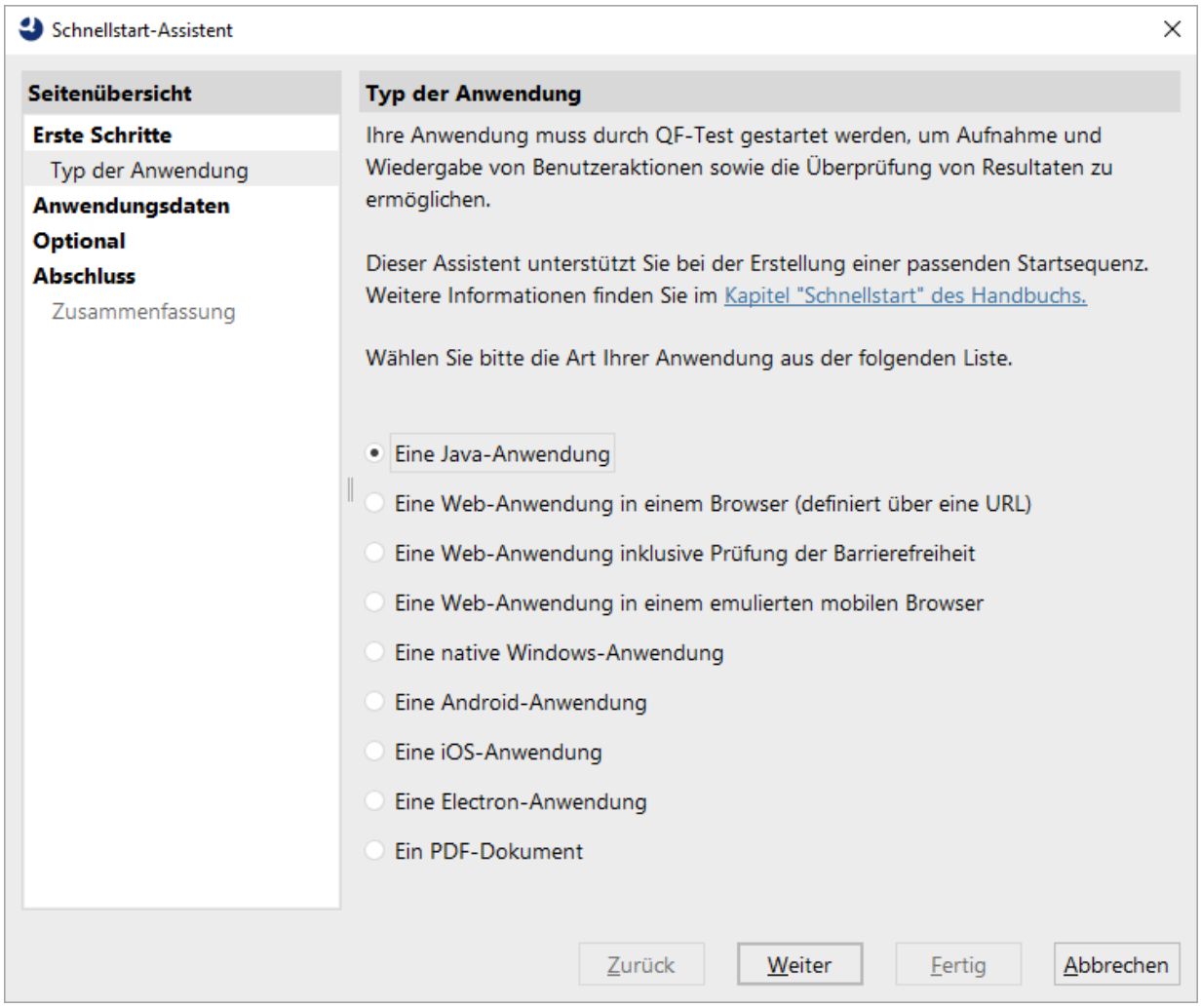

Abbildung 2.2: Auswählen der SUT Art

Im dritten Schritt kann der Typ des Anwendungsprogramms ausgewählt werden.

Aktion

- Wieder belassen wir die **erste Option** ”Ein Startskript bzw. Programm (gestartet mittels `.exe, .cmd, .bat, .app, ...`)”.
- Drücken Sie **Weiter**.

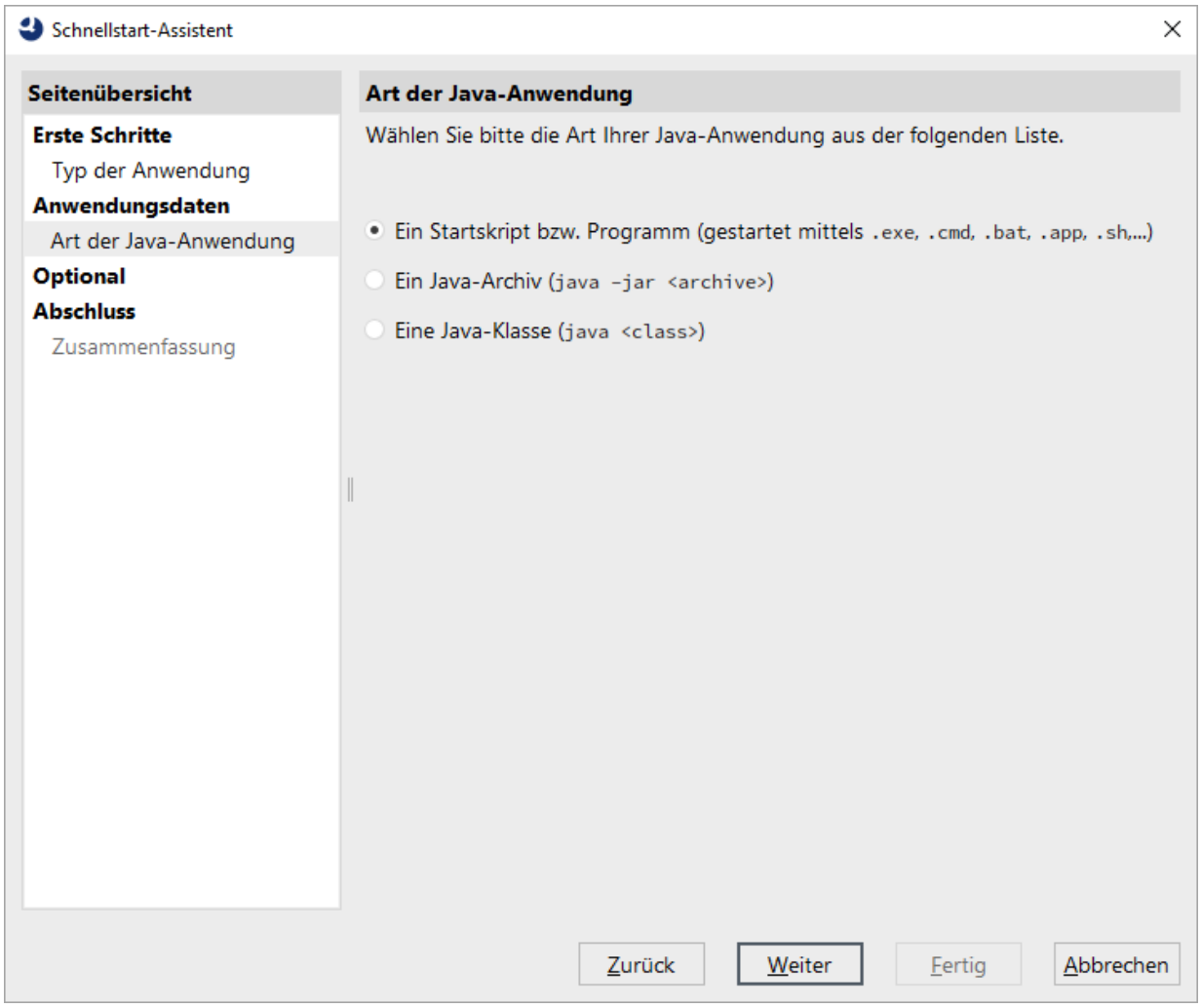

Abbildung 2.3: Wahl des SUT Programmtyps

Nun werden Sie nach dem Programm oder Startskript gefragt.

Aktion

- Nutzen Sie hierzu den **Programm auswählen** Knopf auf der rechten Seite.
- Wechseln Sie in das **Verzeichnis** `.../qftest-9.0.0/demo/carconfigSwing/` in Ihrer QF-Test Installation.
- **Wählen** Sie dort die Datei `CarConfig.cmd` (bzw. `CarConfig.sh`, wenn Sie unter macOS/Linux arbeiten).

Hinweis

Im Bild sieht man eine weitere Möglichkeit: Die Verwendung der Variablen `${qftest:dir.version}` am Beginn, die automatisch zum versionsspezifischen Installationsverzeichnis von QF-Test expandiert. Details zu speziellen QF-Test Variablen finden Sie im Handbuch Kapitel Variablen.

Aktion

- Drücken Sie den **Fertig** Knopf, da wir die weiteren optionalen Schritte für unser einfaches Demo nicht benötigen.

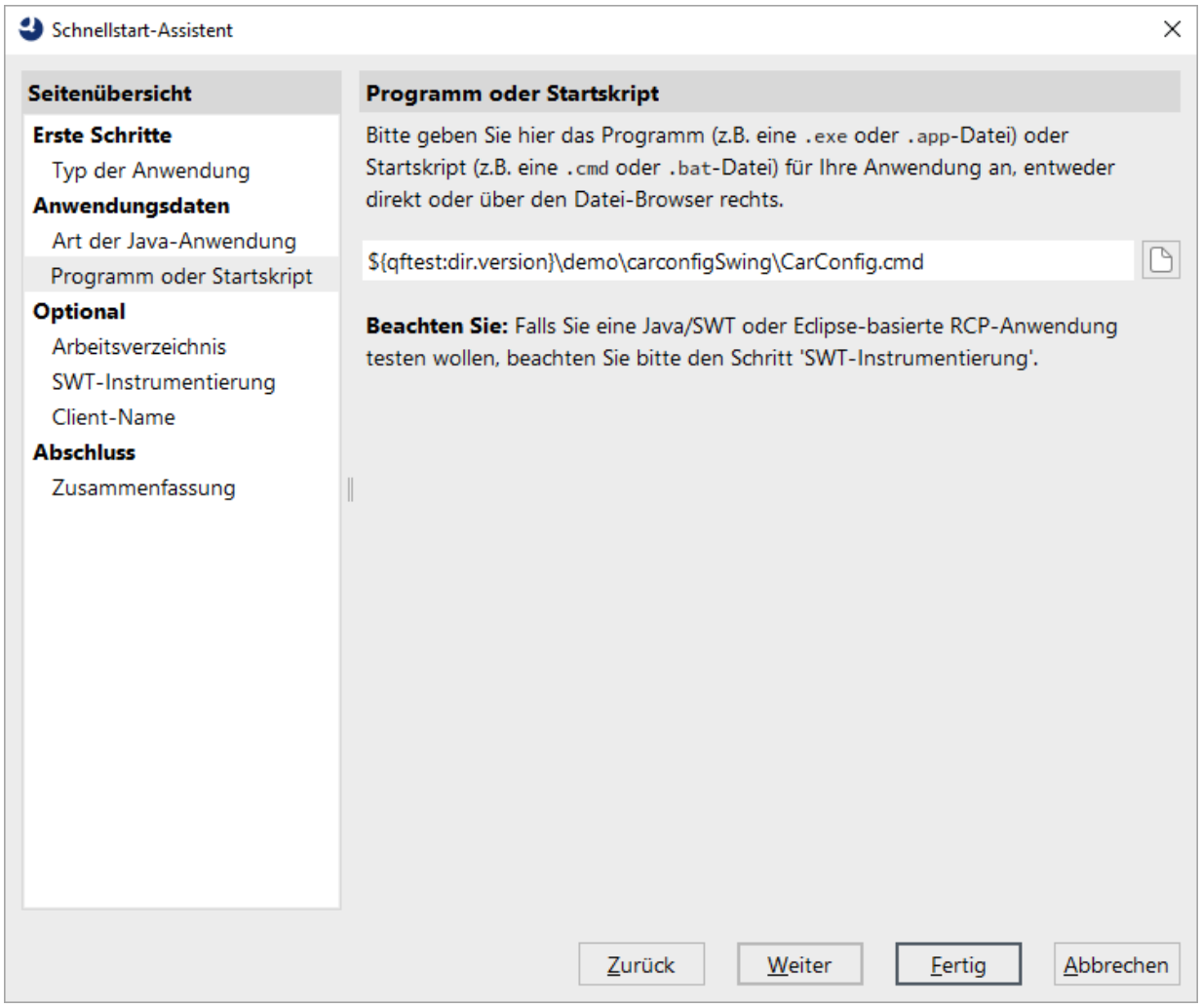

Abbildung 2.4: Auswahl der Programm Datei

Wir gelangen direkt zur Zusammenfassung, die beschreibt, wie es nach dem Beenden des Schnellstart-Assistenten weiter geht.

Aktion

- Drücken Sie den **Fertig** Knopf, um den Assistenten zu beenden.

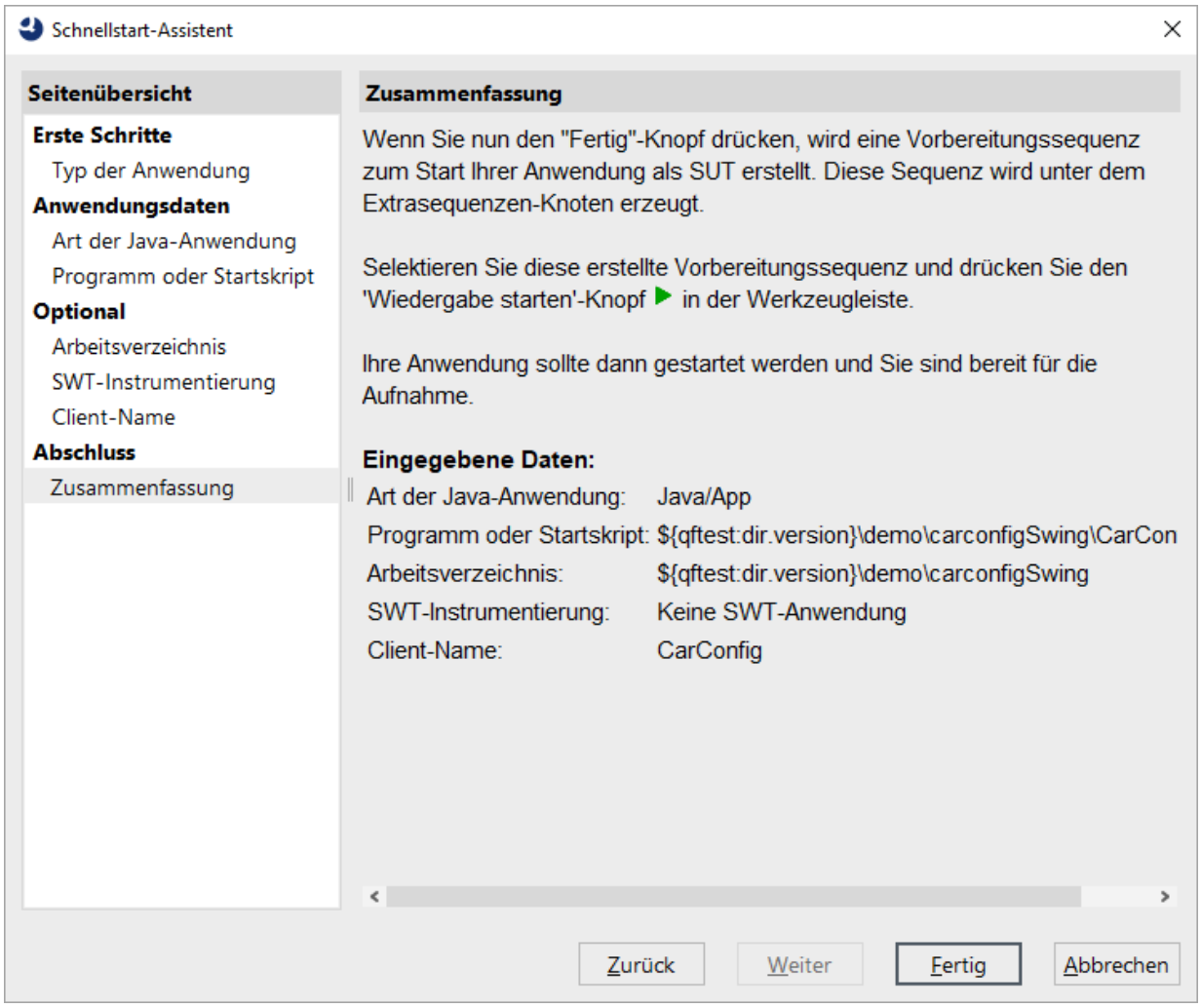

Abbildung 2.5: Zusammenfassung

Die generierte Startsequenz erscheint in den ”Extrasequenzen” der Testsuite und enthält drei Schritte:

- **Variable setzen** - definiert die globale Client Variable, die durchweg in der Testsuite benutzt wird.
- **Warten auf Client** - prüft, ob der Client bereits läuft.
- **SUT starten, wenn notwendig** - falls der Client noch nicht läuft, wird er mit Hilfe eines ”SUT-Client starten” Knotens als zu testendes System (SUT) gestartet und gewartet bis dies erfolgt ist.

Hinweis Die Information, ob der Client bereits läuft wird im ersten ”Warten auf Client” Knoten

einer Variable ”isSUTRunning” gespeichert und in der folgenden ”If” Bedingung ausgewertet. Sie können dies in den entsprechenden Knotendetails sehen. Diese Art der bedingten Ausführung wird später noch im Detail erklärt.

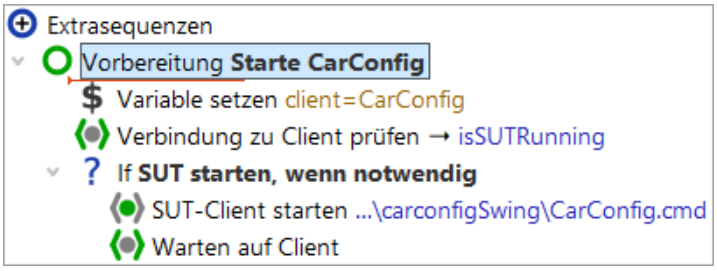

Abbildung 2.6: Generierte Startsequenz

Probieren wir sie aus:

Aktion

- Stellen Sie bitte sicher, dass der Knoten **Vorbereitung: Starte CarConfig** ausgewählt ist.
- **Drücken Sie** ▶ oder betätigen Sie einfach die Eingabe Taste.

Im folgenden Bild ist das Fenster des SUT-Client dargestellt, das nun erscheinen sollte. Da nach dem Abspielen der Fokus zurück zu QF-Test wandert, kann es sein dass das Fenster der Testsuite die Demoanwendung dann verdeckt.

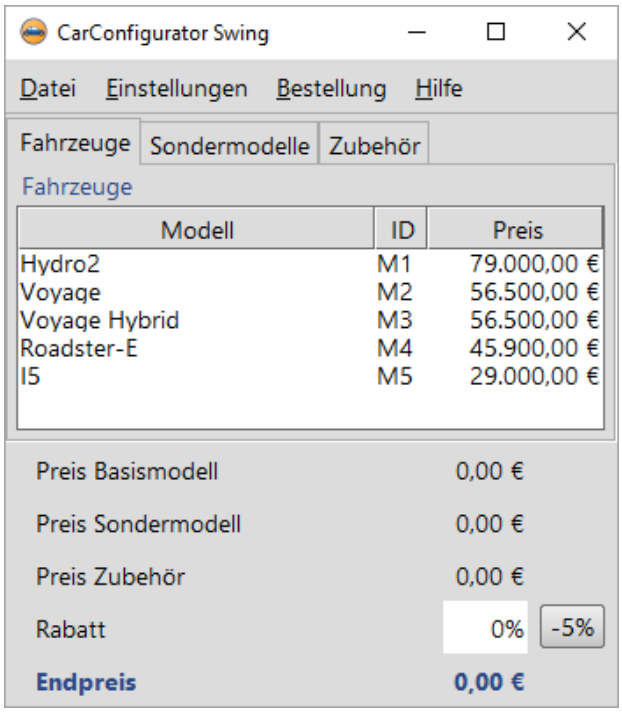

Abbildung 2.7: Das Fenster des ”CarConfigurator”

Am Ende dieses Abschnitts wollen wir unsere Testsuite speichern.

Aktion

- Drücken Sie den Knopf in der Werkzeugleiste oder nutzen Sie die Datei→Speichern Menüaktion bzw. das Tastenkürzel Strg-S.
- Im Datei-Explorer navigieren Sie in ein passendes Verzeichnis, in dem Sie Schreibrechte besitzen, z.B. `Dokumente` in Ihrem Benutzerverzeichnis.
- Geben Sie einen Namen ein z.B. `MeineErstenTests.qft`.
- Beenden Sie die Speicheraktion über den Speichern-Knopf.

2.2 Aufnehmen von Aktionen

Wir werden nun erste Aktionen in unserem Demo aufnehmen:

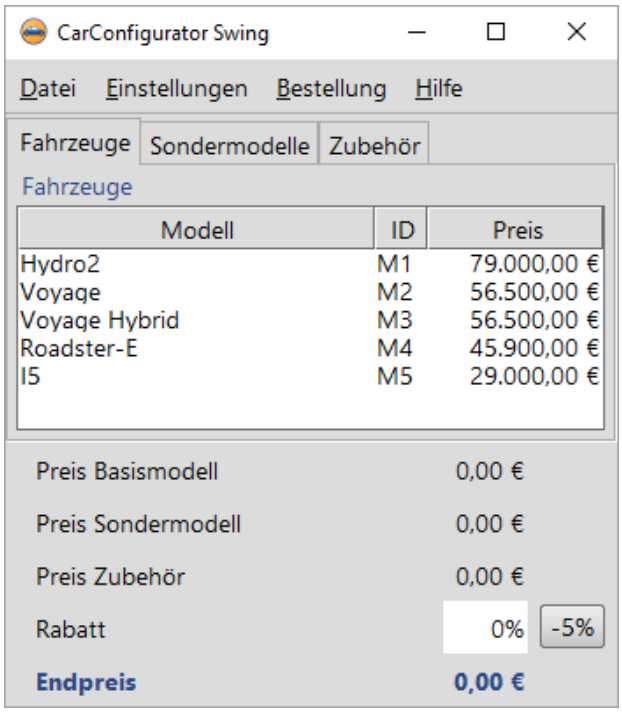

Abbildung 2.8: Aktionen im CarConfigurator Demo aufnehmen

Aktion

- Drücken Sie dazu den ● **Aufnahmeknopf**
- **Wechseln** Sie zum **CarConfigurator** Fenster. Von jetzt ab wird jede Maus- oder Tastaturaktion aufgenommen.
- Wählen Sie mit der Maus das Modell **I5** ganz unten in der Tabelle aus.
- Wechseln Sie zum Tab **Sondermodelle**.
- Wählen Sie dort das Sondermodell **Jazz** über das Dropdown-Menü.
- Zum Schluss klicken Sie wieder auf den ersten Tab **Fahrzeuge**.
- **Beenden Sie die Aufnahme**, indem Sie zurück zum QF-Test Fenster wechseln und dort den Knopf für "Aufnahme beenden" ■ drücken.

Sie finden die aufgenommene Sequenz unter dem **"Extrasequenzen"** Knoten, wie im folgenden Bild dargestellt.

Abbildung 2.9: Der Baum nach Aufnahme der Sequenz

Als Sequenzname wird standardmäßig Datum und Zeit der Erstellung verwendet. Dieser kann anschließend in den Details rechts beliebig angepasst werden.

Aktion

- **Ändern** Sie den Sequenznamen bitte ab zu ”Modell I5 Jazz wählen”
- **Öffnen** Sie die Sequenz um die enthaltenen Knoten zu sehen. Es sollten die erwarteten Mausklicks sein. Sie sollten sogar in der Lage sein, die angesprochenen Komponenten zuordnen zu können.

Abbildung 2.10: Die umbenannte Sequenz

Wir wollen nun die aufgenommene Sequenz abspielen.

Aktion

- Markieren die Sequenz **Modell I5 Jazz wählen**.
- Drücken Sie ▶ **Wiedergabe**.

Sie sollten die exakt gleichen Aktionen sehen, die Sie zuvor aufgenommen haben.

Den aufgenommenen Ablauf sollten Sie auch wiederholt ohne Fehler abspielen können. Rechts unten im Fenster der Testsuite sollte ”Beendet: Keine Fehler” zu sehen sein.

2.3 Aufnahme von Checks

Um das Verhalten des Clients zu überprüfen, verwenden wir Check-Knoten, mit denen man Zustand und Eigenschaften von Elementen abfragen kann. Auch Checks können aufgezeichnet werden.

Aktion

- Zum Aufnehmen eines Checks drücken Sie die den ✓ **"Check aufnehmen"** Knopf.
- Wechseln Sie zum Fenster des SUT. Es erscheint ein Rahmen um die Komponente, über der sich der Mauszeiger befindet.
- Klicken Sie mit der **rechten Maustaste** auf das Wertfeld des **Endpreises**. Das erscheinende Kontextmenü erlaubt Ihnen die Auswahl eines Checks. Die Liste der verfügbaren Checks hängt vom Typ der Komponente ab.
- Wählen Sie den **ersten Eintrag "Text"**, um den textuellen Wert des Feldes zu überprüfen.
- Beenden Sie die Aufnahme durch Drücken des **Stoppknopfs** ■ .

Wieder taucht die neue Aufnahme unter den "Extrasequenzen" auf.

Aktion

- **Benennen** Sie den Sequenzknoten um auf den Namen **"Endpreis prüfen"**.
- **Öffnen** Sie anschließend den Sequenzknoten, um den Checkknoten zu sehen.

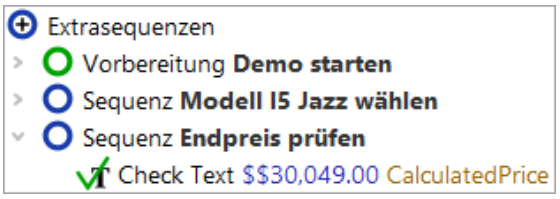

Abbildung 2.11: Die aufgenommene Check-Sequenz

In den Details des "Check Text" Knotens sieht man ebenfalls den erwarteten Wert des Endpreis Feldes.

Aktion

- Auch diese Sequenz können Sie wieder **selbst ausführen**, um die Wiedergabe zu testen.

Im nächsten Schritt wollen wir aus den beiden Sequenzen einen richtigen Testfall aufbauen.

2.4 Erstellen einer Testsuite

Die Basisstruktur unterhalb des Wurzelknotens einer Testsuite ist durch folgende Knoten festgelegt:

- Eine beliebige Anzahl von ”Testfallsatz” und ”Testfall” Knoten, um funktionale Tests zu spezifizieren und zu strukturieren.
- ”Prozeduren” - hier können wiederverwertbare Sequenzen in Prozeduren organisiert werden
- ”Extrasequenzen” - unsere Spielwiese für Aufnahmen etc.
- ”Fenster und Komponenten” - das eigentliche Herz der Testsuite. Hier sind alle aufgenommenen Fenster und Komponenten des SUT mit ihren Eigenschaften enthalten

Funktionale Testfälle werden durch ”Testfall” Knoten repräsentiert und mittels ”Testfallsatz” Knoten gruppiert bzw. strukturiert.

”Vorbereitung” und ”Aufräumen” Knoten können Aktionen enthalten, um einen wohldefinierten Zustand vor und nach einem Testfall sicherzustellen.

Aktion

- Wir beginnen mit dem **Umbenennen** des ”Testfallsatz” Knotens von ”unbenannt” in ”Demo Tests”.
- Falls ein **Dialog** bzgl. der Aktualisierung von Verweisen erscheint, können wir diesen einfach mit **”Ja”** beantworten.
- Der nächste Schritt ist, den vom Schnellstart-Assistenten erzeugten Knoten **”Vorbereitung”** in den **”Testfallsatz” zu verschieben** und zwar an die **erste Position** vor den enthaltenen Testfall. Das Verschieben kann mit Hilfe der Maus (Drag&Drop), des Kontextmenüs (rechte Maustaste Ausschneiden/Einfügen) oder der Tastenkombination Strg-X und Strg-V durchgeführt werden.

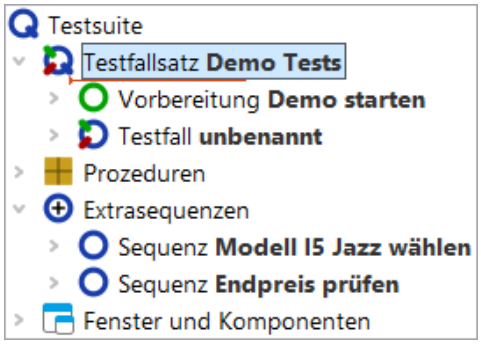

Abbildung 2.12: Beginn der Strukturierung

Als Nächstes gilt es, aus den beiden vorher aufgezeichneten Sequenzen einen Testfall zu machen.

Aktion

- **Benennen** Sie dazu den Testfall Knoten von ”unbenannt” in ”Erster” um.
- **Öffnen** Sie den Testfall Knoten durch einen Klick auf das ’>’ Symbol.
- **Verschieben** Sie die beiden Sequenzen aus den ”Extrasequenzen” in den Testfall.

Wenn Sie den Testfall Knoten nicht öffnen, versucht QF-Test die Sequenzknoten hinter dem Testfall Knoten auf der gleichen Ebene einzufügen. Dies ist jedoch für Sequenzknoten nicht zulässig.

QF-Test nimmt immer Sequenzen auf. Diese haben die gleiche Funktion wie Testschritte. Testschritte werden jedoch im Bericht aufgeführt. Man kann sie ineinander umwandeln, was wir Ihnen in den nächsten Schritten informationshalber zeigen wollen.

Aktion

- Öffnen Sie das **Kontextmenü** für den **ersten** der beiden Sequenzknoten mit der rechten Maustaste.
- Wählen Sie Knoten konvertieren in...→Testschritt
- Führen Sie dasselbe für den **zweiten** Sequenzknoten durch.

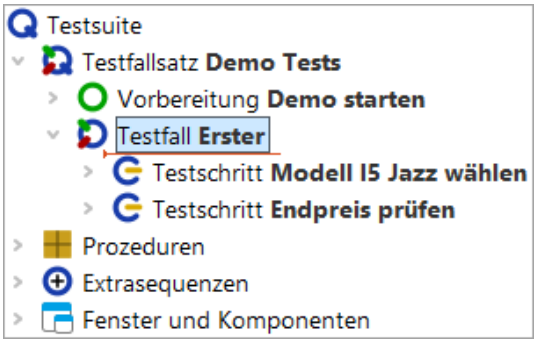

Abbildung 2.13: Der Baum nach der Neustrukturierung

Damit haben wir die wichtigsten Schritte zur Strukturierung unserer Testsuite abgeschlossen.

2.5 Beenden der Anwendung

Was uns als Basiselement noch fehlt, ist eine Aufräumsequenz, die das SUT sauber beendet.

Es gibt verschiedene Wege eine Anwendung zu beenden, z.B. über den ”Fenster schließen” Knopf rechts oben, durch Drücken von Alt-F4 oder das Menü Datei→Beenden.

Alle diese Varianten lassen sich direkt aufzeichnen. Wir werden die letzte Möglichkeit nutzen, so dass Sie die folgenden Schritte durchführen können:

Aktion

- **Aufnahme starten** ● .
- Menüaktion **Datei→Beenden** durchführen.
- Das Fenster der Demoanwendung verschwindet.
- **Aufnahme beenden** ■ .
- Benennen Sie die aufgenommene Sequenz **in "Demo beenden" um**.
- Öffnen Sie das **Kontextmenü** für den Sequenzknoten und wählen Sie den Menüpunkt **Knoten konvertieren in...→Aufräumen** .
- Zuletzt **verschieben** Sie den Aufräumenknoten nach oben, so dass er **der letzte Knoten im Testfallsatz** ist.

Hinweis

Der Aufräumknoten kann nur per Drag and Drop in den Testfallsatz verschoben werden, wenn dessen letzter Kindknoten eingeklappt ist. Um einen Knoten während einer Drag and Drop Operation ein- oder auszuklappen, verweilen Sie einen Moment mit dem Mauszeiger über dem Dreieck neben dem Knoten.

Sie sollten folgendes Resultat erhalten:

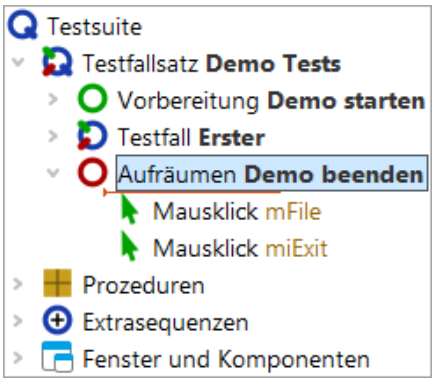

Abbildung 2.14: Die einfache Aufräumsequenz

Damit haben wir die wichtigsten Schritte zur Strukturierung unserer Testsuite abgeschlossen.

2.6 Gesamte Suite ausführen

Als Abschluss wollen wir unsere neue Suite ausführen:

Aktion

- **Beenden** Sie dazu nun bitte den SUT Client, falls er läuft.
- **Markieren** Sie den "Testsuite" **Wurzelknoten**.
- Führen Sie diesen durch Drücken von "Wiedergabe" ▶ oder der Eingabe Taste aus.

Das SUT sollte erscheinen, der Testfall aufgeführt und das SUT wieder beendet werden.

Wie wir wissen, wird das Ergebnis des Testlaufs im Protokoll festgehalten:

Aktion

- Um dieses anzuschauen, können wir den Button **"Protokoll anzeigen"** in der Werkzeugleiste oder alternative die Tastenkombination Strg-L nutzen.

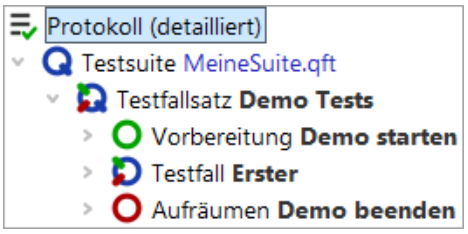

Abbildung 2.15: Der Protokollbaum der eigenen Testsuite

Wir hatten bereits im ersten Kapitel gesehen, wie das Protokoll für die Fehleranalyse genutzt werden kann.

Wir möchten damit dieses Kapitel beenden und einen Schritt weiter in Richtung Fehlerdiagnose gehen. Hierbei stellt der Debugger ein wichtiges Werkzeug innerhalb QF-Test dar. Seine Handhabung und Möglichkeiten werden im nächsten Kapitel beschrieben.

Kapitel 3

Eine Prozedur erstellen (Java)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Eine Prozedur erstellen'
https://www.qftest.com/de/yt/tutorial-3.html

In Kapitel eins und zwei haben Sie gelernt wie man eine Applikation über QF-Test startet, so dass Maus- und Tastatureingaben aufgenommen werden können, auch wie man Checks aufnimmt und wie man das Ergebnis zu einem Testfall zusammenfasst. Diese Herangehensweise ist ausreichend, solange die Tests einfach und nicht allzu viele sind. Sobald jedoch die Zahl der Tests zunimmt, ist es wichtig, sogenannte "Prozeduren" einzusetzen.

Prozeduren sind ein Mittel um Sequenzen wiederverwendbar zu machen und damit Doppelvorkommen zu vermeiden. Dies ist wichtig, um eine einfache und effiziente Wartbarkeit von Tests über die Zeit zu erreichen.

Prozeduren können in Packages gruppiert werden. Prozeduren und Packages sind die Basis für die Modularisierung der Tests.

3.1 Wiederverwendbare Abschnitte identifizieren

In diesem Abschnitt werden wir die Testsuite ErsteJavaTests.qft, die Sie bereits aus Kapitel 1 kennen, weiterentwickeln.

Aktion

- **Kopieren Sie ErsteJavaTests.qft** aus dem Unterverzeichnis `qftest-9.0.0/doc/tutorial` der QF-Test Installation in ein Arbeitsverzeichnis und
- **öffnen Sie ErsteJavaTests.qft**.

- Wenn Sie die Änderung, die wir an der Demo-Testsuite vornehmen werden, sichern wollen, so **speichern Sie diese in einem Arbeitsverzeichnis** wie am Ende von Abschnitt 2.1 (19) beschrieben.

Bitte sehen Sie sich den Testschritt "Zurücksetzen" in den beiden Testfällen an. Die beiden Testschritte sind identisch.

Abbildung 3.1: Zwei identische Testschritte

Gemäß obiger Überlegungen wäre es also sinnvoll, den Testschritt in eine Prozedur umzuwandeln.

3.2 Manuelle Erstellung von Prozeduren

Es gibt mehrere Methoden Prozeduren zu erstellen und Prozeduraufrufe einzufügen. Wir fangen mit der manuellen an, bei der ein (leerer) Prozedurknoten eingefügt wird, in den dann die entsprechenden Aktionen verschoben werden. Danach erstellen wir den zugehörigen Prozeduraufruf.

Es ist gut, wenn man diese grundlegenden Schritte kennt. Es gibt jedoch eine zweite, elegantere Methode Prozeduren zu erstellen, die wir im Anschluss zeigen werden.

Also los, fügen wir eine Prozedur von Hand ein: Wir beginnen mit dem Anlegen des Prozedurknotens, dem wir einen geeigneten Namen geben.

Aktion

- **Öffnen Sie den Prozedur Knoten** und achten Sie darauf, dass er auch selektiert (blau markiert) ist.

- Wählen Sie **Einfügen→Prozedurknoten→Prozedur**.
- Tragen Sie als Name "**zurücksetzen**" ein. Die anderen Felder brauchen nicht befüllt zu werden.
- Drücken Sie **OK** um die Erstellung der Prozedur abzuschließen.
- **Öffnen Sie** die neu erstellte "zurücksetzen" Prozedur.

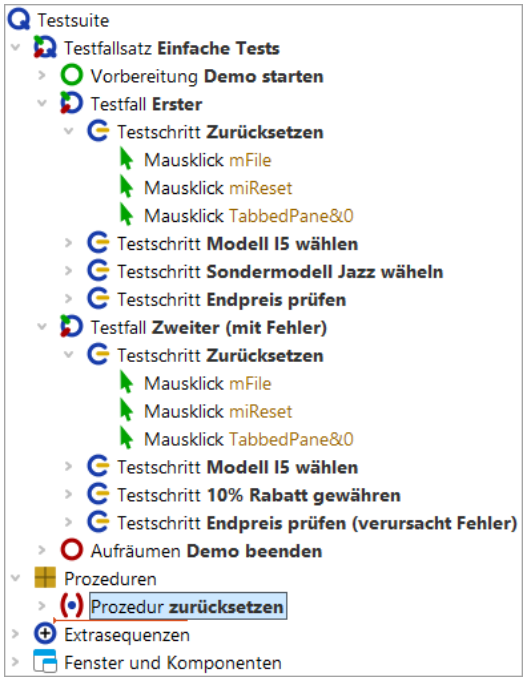

Abbildung 3.2: Prozedurknoten erstellen

Im zweiten Schritt befüllen wir die Prozedur mit den entsprechenden wiederverwendbaren Aktionen.

Aktion

- **Selektieren Sie** die drei "Mausklick" Knoten im Testschritt. Um mehr als einen Knoten zu selektieren, klicken Sie den ersten der Knoten an, dann drücken Sie die Shift Taste und klicken den letzten der zu selektierenden Knoten während Sie die Shift Taste gedrückt halten.
- **Verschieben** Sie diese nach unten in die Prozedur, z.B. mit der Maus (Drag and drop) oder über Ausschneiden/Einfügen im Menü Bearbeiten oder über das Kontextmenü.

Abbildung 3.3: Prozedur mit Inhalt befüllen

Im dritten Schritt fügen wir einen Prozeduraufruf an Stelle der drei verschobenen Mausklicks ein.

Aktion

- **Selektieren** Sie den **Testschritt "Zurücksetzen"**, der geöffnet sein sollte.
- **Wählen Sie** den Menüpunkt Einfügen→Prozedurknoten→Prozeduraufruf oder verwenden Sie das Tastaturkürzel Strg-A.

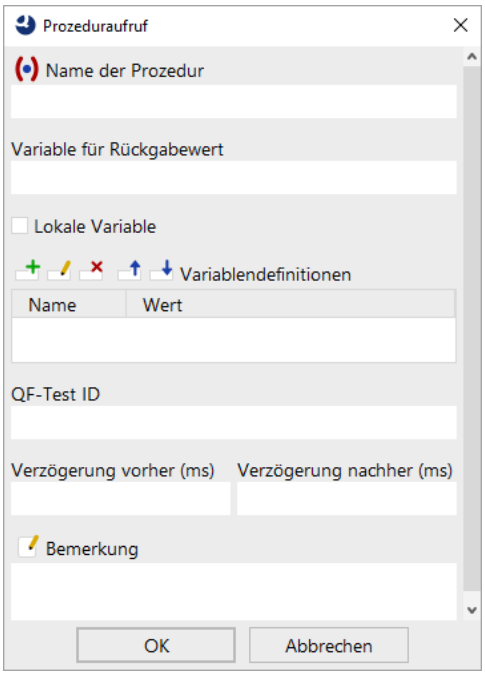

Abbildung 3.4: Prozeduraufruf einfügen

Aktion

- **Drücken Sie** den Prozedur-Auswahlknopf in dem Dialog links neben der Beschriftung "Name der Prozedur".
- **Wählen Sie "zurücksetzen"** aus der Prozedurliste. Weitere Eingaben sind nicht nötig.
- Drücken Sie **OK** in beiden Dialogen um die Erstellung des Prozeduraufrufs abzuschließen.

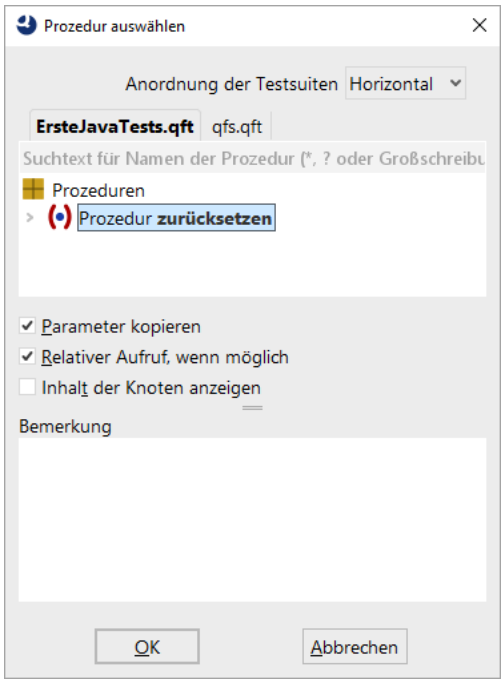

Abbildung 3.5: Prozedur auswählen

Um wirklich einen Mehrwert durch die Prozedur zu erlangen, müssen wir nun den Inhalt des Testschritts im zweiten Testfall ebenfalls durch einen Aufruf der Prozedur ”zurücksetzen” ersetzen.

Sie können dies wie oben beschrieben tun oder Sie führen folgende **alternative Schritte** zur Erstellung des Prozeduraufrufs aus:

Aktion

- **Öffnen Sie** den Testschritt ’Zurücksetzen’ des zweiten Testfalls.
- **Löschen Sie** die drei Mausklick Knoten daraus.
- **Selektieren Sie** den Prozedurknoten ”zurücksetzen”.
- **Ziehen Sie** den Prozedurknoten ”zurücksetzen” mit der Maus in den Testschrittknoten. Kopieren/einfügen kann ebenfalls verwendet werden. Dadurch wird der Prozedurknoten nicht verschoben, sondern ein entsprechender Prozeduraufruf erzeugt.

Die Testsuite sollte anschließend wie folgt aussehen:

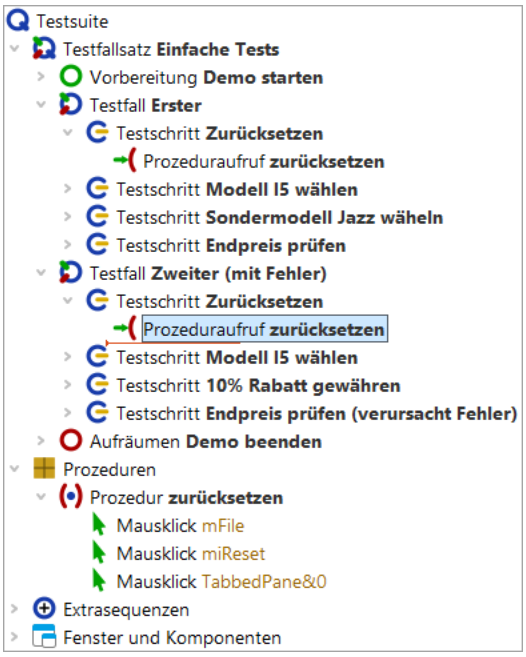

Abbildung 3.6: Testsuite mit Prozedur

Wenn Sie nun die Testfälle ausführen, sollten diese funktionieren wie zuvor. Im Protokoll sind jetzt auch die Prozeduraufrufe und deren Ausführung zu sehen.

3.3 Knoten in Prozedur konvertieren

Wie bereits am Anfang des letzten Abschnitts erwähnt, bietet QF-Test eine Alternative um Prozeduren wesentlich schneller zu erstellen.

Aktion

- **Markieren Sie den Testschritt oder Sequenz-Knoten**, der die wiederverwendbaren Schritte enthält, die zur Prozedur umgewandelt werden sollen.
- Wählen Sie den Menüpunkt **Operationen→Knoten konvertieren in→Prozedur** aus oder verwenden Sie das Tastaturkürzel Strg-Umschalt-P.

Wie Sie sehen, ist der Testschritt bzw. der Sequenzknoten verschwunden. Anstelle dessen befindet sich ein Prozeduraufruf. Außerdem wurde eine Prozedur mit dem Namen des ehemaligen Testschritts bzw. der Sequenz im Abschnitt "Prozeduren" erstellt. Sie enthält genau die gleichen Kindknoten wie zuvor der Testschritt bzw. die Sequenz.

Bei der Aufnahme einer Sequenz in QF-Test hat sich das Vorgehen bewährt, der Sequenz sofort einen Namen zu geben und sie anschließend in eine Prozedur zu konvertieren. Auch wenn man nur eine Vermutung hat, dass sich die aufgenommenen Schritte irgendwo wiederholen könnten.

Kapitel 4

Komponenten (Java)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Komponenten'
https://www.qftest.com/de/yt/tutorial-4.html

Werfen wir nun einen Blick auf den letzten verbleibenden Bereich des Testsuite Fensters, den Fenster und Komponenten Knoten. Zuvor möchten wir Ihnen jedoch zeigen, wie Unterelemente von Komponenten wie Tabellen, Bäumen und Listen adressiert werden.

4.1 Adressierung von Unterelementen von Tabellen, Bäumen und Listen

Unterelemente von Tabellen, Bäumen und Listen werden über Indizes angesprochen. Die wichtigsten beiden Indextypen sind der numerische und der Textindex. Zur Demonstration nehmen wir als nächstes einen Mausklick auf eine Tabellenzelle auf und sehen uns die aufgenommene QF-Test ID der Komponente näher an.

Aktion

- **Starten Sie das CarConfig Demo**, falls dieses nicht bereits läuft. Führen Sie dazu den Vorbereitung Knoten in der Testsuite aus.
- **Aktivieren Sie den Aufnahmemodus** über "Aufnahme starten" ● .
- **Klicken Sie auf eine Tabellenzelle**, z.B. das erste Modell.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .

Den aufgenommenen Mausklick finden Sie im Bereich Extrasequenzen.

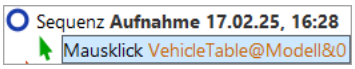

Abbildung 4.1: Adressierung einer Tabellenzelle

Die aufgenommene QF-Test ID der Komponente ist `VehicleTable@Modell&0`. Sie setzt sich aus den folgenden Teilen zusammen:

- `VehicleTable` ist die QF-Test ID der Komponente der Tabelle selbst.
- `@` und `&` trennen die einzelnen Teile voneinander. Gleichzeitig definieren sie den Typ des darauf folgenden Index: auf `@` folgt ein Textindex, auf `&` ein numerischer Index.
- `Modell` ist der Textindex für die Spalte mit der Überschrift 'Modell'.
- `0` ist der numerische Index für die erste Tabellenzeile.

Hinweis Numerische Indizes beginnen immer mit 0.

Sie können beide Indextypen für Zeilen und Spalten verwenden. Dabei ist nur wichtig, dass das Trennzeichen und der Typ des folgenden Index zusammenpassen.

Aktion

- **Ändern Sie die QF-Test ID der Komponente** so, dass das dritte Preisfeld adressiert wird. Verwenden Sie dafür numerische Indizes.

Die Lösung hierfür lautet `VehicleTable&1&2`.

Um das Modell 'I5' über Textindizes anzusprechen, tragen Sie `VehicleTable@Modell@I5` ein. Das gleiche Feld kann man numerisch mittels `VehicleTable&0&4` ansprechen oder mit gemischten Indizes mittels `VehicleTable&0@I5` oder `VehicleTable@Modell&4`.

Der dritte Indextyp von QF-Test ist ein Index mit regulärem Ausdruck. Reguläre Ausdrücke werden verwendet, um Zeichenketten durch einen Ausdruck zu ersetzen, der verschiedene Zeichenketten adressieren kann. Sozusagen eine "Sternchensuche", wobei reguläre Ausdrücke wesentlich mächtiger sind und eine eigene Syntax besitzen. Eine genauere Beschreibung regulärer Ausdrücke finden Sie im Handbuch. Beispiel: Das Modell 'I5' könnte man also auch über `VehicleTable@Modell%I.*` ansprechen.

Listen werden analog zu Tabellen adressiert, nur dass sie nur einen einzigen Index benötigen.

Bäume haben ebenfalls nur einen einzigen Index. Dieser ist der Pfad durch den Baum zu dem adressierten Baumknoten. Der Pfad setzt sich aus den einzelnen Knoten zusammen, die durch Schrägstriche ("/") voneinander getrennt werden.

Aktion

- **Starten Sie das CarConfig Demo**, falls dieses nicht bereits läuft. Führen Sie dazu den Vorbereitung Knoten in der Testsuite aus.
- **Öffnen Sie das Baum-Beispiel:** Wählen Sie im CarConfig Demo den Menüpunkt Einstellungen→Sondermodelle..., selektieren Sie ein Modell und drücken die Schaltfläche "Details".
- **Aktivieren Sie den Aufnahmemodus** über "Aufnahme starten" ● .
- **Klicken Sie auf einen Baumknoten**, z.B. "Beschreibung".
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .

Für den Baumknoten "Beschreibung" wird die folgende QF-Test ID der Komponente aufgenommen: `DetailsTree@/Information/Beschreibung`. Die einzelnen Bestandteile davon sind:

- `DetailsTree` ist die QF-Test ID der Komponente des Baums selbst.
- `@` trennt die QF-Test ID der Komponente des Baums vom Index. Die Syntax ist hierbei analog zu der der Tabellenindizes, d.h. `@` steht für einen Textindex, `&` für einen numerischen Index und `%` für einen Index mit regulärem Ausdruck.
- `/Information/Beschreibung` ist der Textindex für den Baumpfad zum Knoten "Beschreibung".

Wenn Sie den Knoten über einen numerischen Index adressieren wollen, verwenden Sie `DetailsTree&/0/1`.

4.2 Der Bereich Fenster und Komponenten

Zum Thema "Komponenten" gibt es mehrere 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:

Ein einfacher Fall wird im Video

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

erläutert.

Einen komplexeren Fall gibt es in

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

Video-Mitschnitt des Spezialwebinars

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

.

QF-Test speichert die Informationen, wie es die im GUI des SUT angezeigten Komponenten wiederfindet, im Bereich Fenster und Komponenten ab. Dabei analysiert QF-Test bei der Aufnahme die erhaltenen GUI-Element-Informationen und speichert diejenigen, mit denen der Anwender interagiert hat, in den Details der Komponente Knoten ab.

Swing, JavaFX und SWT verfügen über klare Konzepte, wie eine bestimme Komponente vom Entwickler technisch zu implementieren ist. Daher ist es bei Anwendungen, die mit diesen Sprachen entwickelt wurden, selten nötig, sich näher mit den Komponentenknoten zu befassen. In den meisten Fällen wird QF-Test die Komponente, auf die eine Aktion im GUI abgespielt werden soll, wiederfinden. Nur wenn sich die Oberfläche der Applikation jenseits des Wiedererkennungsalgorithmus von QF-Test verändert hat, müssen Sie sich mit den Komponente Knoten befassen.

Hinweis

In diesem Fall finden Sie im Handbuch, Kapitel Lösung von Problemen bei der Wiedererkennung detaillierte Anweisungen. Dort gibt es auch Links auf Videos mit entsprechenden Beispielen.

In diesem Abschnitt wollen wir Ihnen eine Vorstellung davon vermitteln, welche Informationen in Komponente Knoten abgespeichert werden und wie diese von QF-Test für die Wiedererkennung verwendet werden.

Sehen Sie sich die Details einer "TextField" Komponente an.

Aktion

- **Starten Sie das CarConfig Demo**, falls dies nicht bereits läuft. Führen Sie dazu den Vorbereitung Knoten in der Testsuite aus.
- **Öffnen Sie die Prozedur "Endpreis prüfen"**.
- **Öffnen Sie das Kontextmenü des 'Check Text'-Knotens**.
- **Springen Sie zum Knoten der "TextField" Komponente** über den Menüpunkt Komponente finden im Popup-Menü oder über das Tastaturkürzel Strg-W.

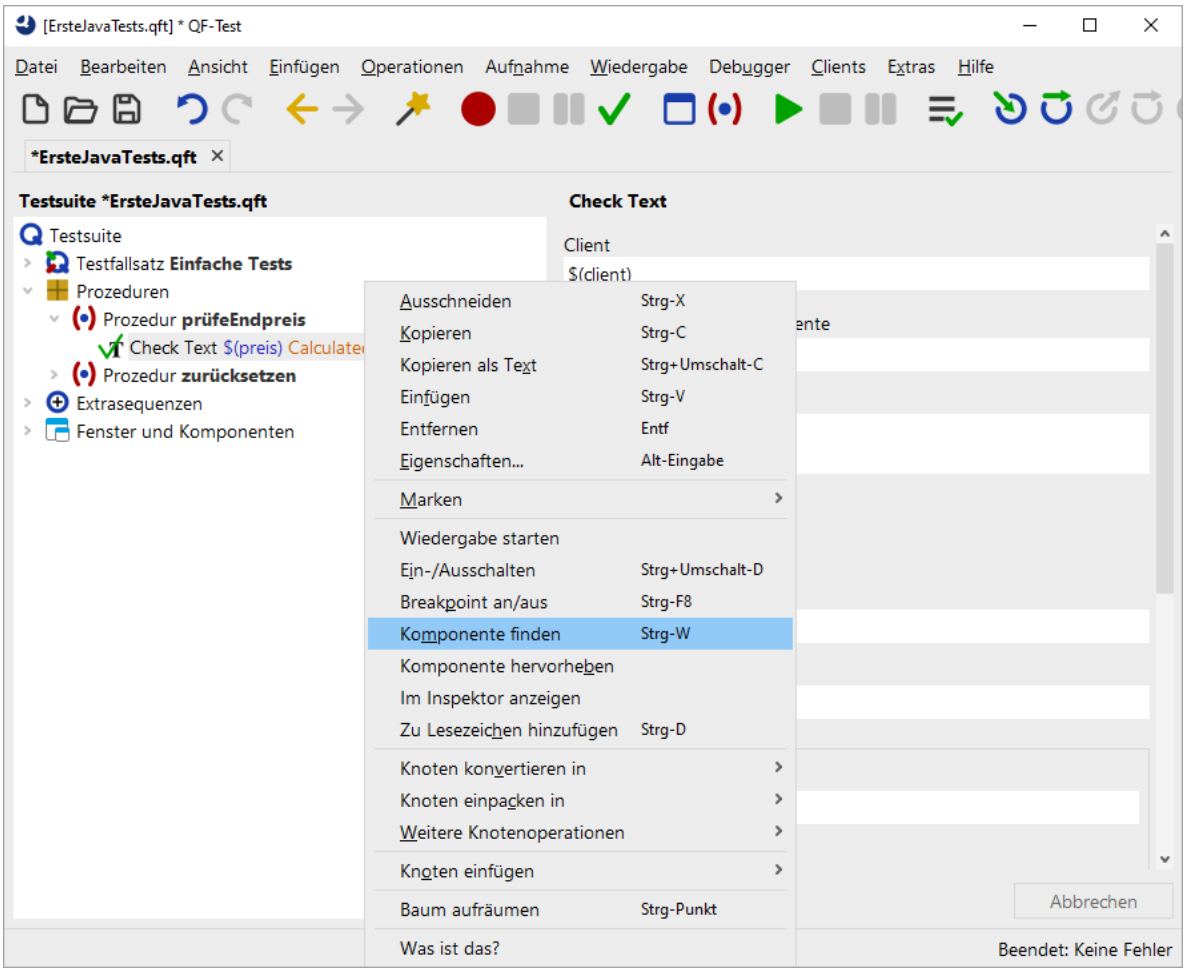

Abbildung 4.2: Komponente finden

Dadurch gelangen Sie direkt zum Knoten ”TextField CalculatedPrice” im Bereich Fenster und Komponenten.

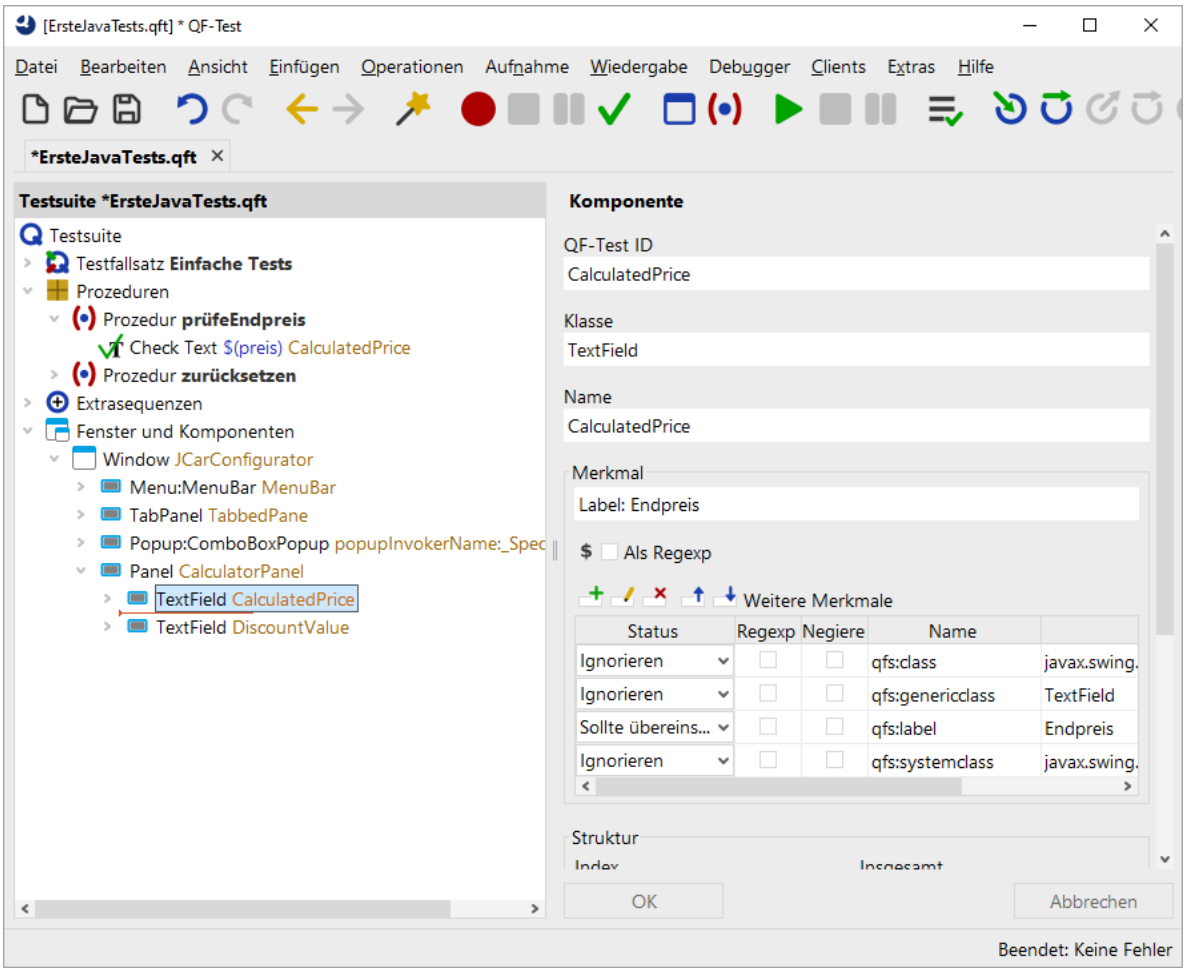

Abbildung 4.3: Komponentenbaum

Für den Komponente Knoten mit der QF-Test ID `CalculatedPrice` sehen die Details wiefolgt aus:

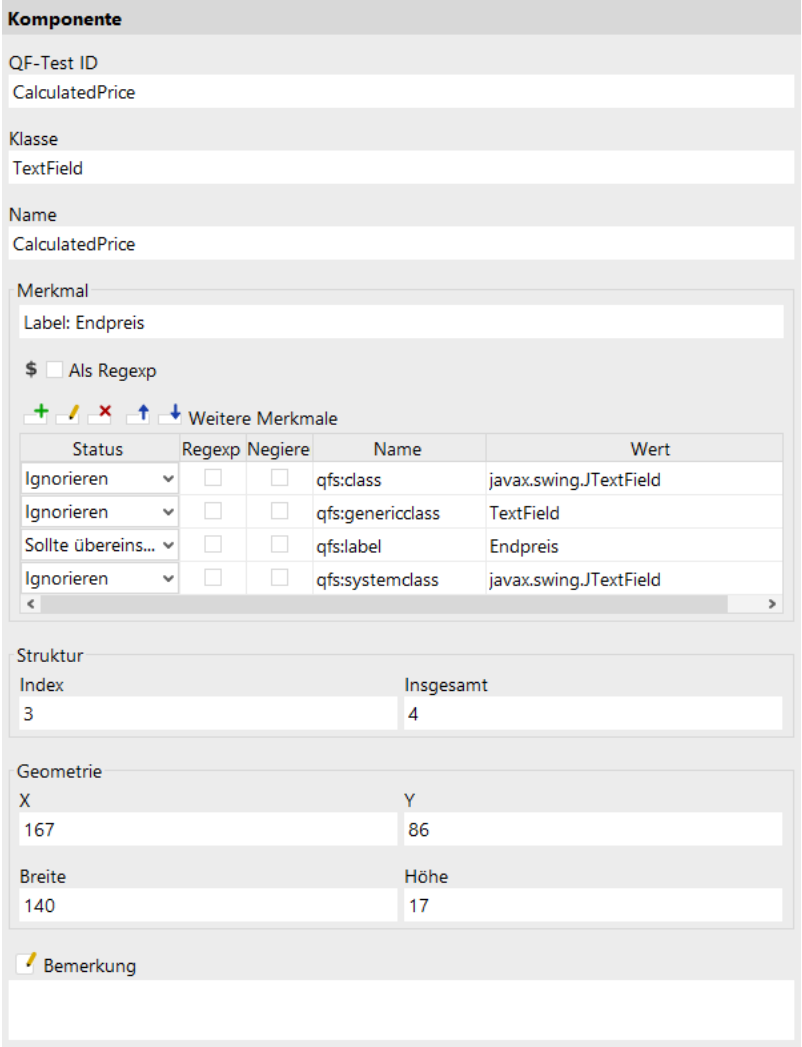

Abbildung 4.4: Details eines Komponente Knoten

Wie werden die Attribute des Komponente Knotens zur Wiedererkennung der GUI-Elemente eingesetzt?

Das erste Attribut ist **QF-Test ID**, das in den Testfällen und Prozeduren zur Ansprache der Komponente verwendet wird. Alle anderen Attribute beziehen sich auf Eigenschaften des GUI-Elements.

Das folgende Attribut ist die **Klasse**. In unserem Fall `TextField`. Für die Komponentenerkennung ist die Klasse ein eindeutiges Merkmal. Die angezeigte Klasse ist eine von QF-Test verallgemeinerte Klasse, basierend auf der Java-Klasse oder der Java-Systemklasse. Durch diese generische Klasse werden die Tests unabhängig von der konkreten Implementierung und können leicht portiert werden, z.B. von Swing auf den Nachfolger JavaFX. Die speziellen Werte werden in der Tabelle "Weitere Merkmale" mit

den Namen `qfs:class` und `qfs:systemclass` abgespeichert. Sie spielen standardmäßig für die Erkennung keine Rolle, können aber für Sonderfälle genutzt werden.

Weitere Beispiele für Klassen sind ”Panel”, ”Dialog” und ”Button”.

Dass Attribut **”Name”** enthält den Namen oder die Id, die dem GUI-Element seitens des Programmierers gegeben wurde. Falls ein Name vorhanden ist, ist dies für QF-Test zusammen mit der Klasse zur Komponentenerkennung ausreichend. Die übrigen Attribute bleiben dann unberücksichtigt.

Wenn weder ein Name noch eine Id durch den Entwickler gesetzt wurde und das Attribut ”Name” daher keinen Wert enthält, verwendet QF-Test andere Kriterien wie z.B. einen bestimmten Text, der zur Komponente gehört, Index und Geometrie.

Ein zur Komponente gehörendes Merkmal wäre bei einer Schaltfläche zum Beispiel der darauf angezeigte Text. QF-Test speichert Texte, die direkt zur Komponente gehören, im Attribut **”Merkmal”** ab. Texte in der Nähe einer Komponente, die QF-Test als mögliche Beschriftung identifiziert, werden ebenfalls im Merkmal-Attribut abgespeichert, wobei sie den Präfix `Label:` erhalten. Zusätzlich werden diese Texte in der Tabelle **”Weitere Merkmale”** unter dem Namen `qfs:label` gespeichert.

Die **Struktur** Informationen beziehen sich auf alle GUI-Elemente der jeweiligen Klasse. Die Gesamtzahl der GUI-Elemente der Klasse wird im Attribut ”Insgesamt”, der Index der Komponente selbst im Attribut ”Index” abgespeichert.

Am Schluss befinden sich die Werte für die **Geometrie**. Diese erhalten im Wiedererkennungsalgorithmus die geringste Gewichtung. In seltenen Fällen kann es jedoch vorkommen, dass sie die einzigen Kennzeichen sind, die zur Identifizierung des GUI-Elements zur Verfügung stehen.

Wenn Sie an weiteren Details zum genauen Ablauf bei der Wiedererkennung interessiert sind, können Sie diese im Kapitel Komponentenerkennung der technischen Referenz im Handbuch nachlesen.

Um ein Gefühl für die Komponentenerkennung zu erhalten, können Sie ein bisschen mit den Attributwerten herumspielen, bis QF-Test das GUI-Element nicht mehr findet oder sogar eine falsche Komponente auswählt. Sie werden feststellen, dass die Änderungen beträchtlich sein müssen, bevor QF-Test ein falsches GUI-Element identifiziert. Das heißt, dass die Komponentenerkennung von QF-Test sehr robust ist und sich bei neuen Versionen einer Anwendung ein erheblicher Anteil der Attribute eines GUI-Elements verändern kann, bevor die Komponentenerkennung fehlschlägt - selbst wenn das GUI-Element keinen Namen oder keine Id hat.

Beim Klick auf einen Komponente Knoten markiert QF-Test das erkannte GUI-Element mit einem dunkelblauen Rand.

Aktion

- **Löschen Sie den Wert `CalculatedPrice` aus dem Name-Attribut**, da QF-Test sonst die nachfolgenden Attribute nicht berücksichtigt.

- **Ändern Sie das Merkmal-Attribut** von `Label: Endpreis` auf `Label: xxx`.
- **Klicken Sie auf den "TextField" Knoten** um zu sehen, ob QF-Test immer noch das Endpreis-Feld markiert.
- **Setzen Sie das Merkmal-Attribut** zurück auf `Label: Endpreis`, entweder über den entsprechenden Button in der Werkzeugleiste oder über das Tastaturkürzel Ctrl-Z.
- **Ändern Sie den Wert von `qfs:label` in der Weitere-Merkmale-Tabelle** von `Endpreis` auf `Rabatt`.
- **Klicken Sie auf den 'TextField'-Knoten** um zu sehen, ob QF-Test immer noch das Endpreis-Feld markiert.
- **Setzen Sie den Wert von `qfs:label` in der Weitere-Merkmale-Tabelle** zurück auf `Endpreis`.
- **Setzen Sie alle Struktur- und Geometrie-Attribute** auf andere Werte und überprüfen Sie, ob QF-Test immer noch das Endpreisfeld im GUI markiert.
- **Ändern Sie das Merkmal-Attribut** von `Label: Endpreis` auf `Label: Rabatt`.

Nun markiert QF-Test das Rabattfeld.

Diese Übung ist nur eine kleine Spielerei mit der Komponentenerkennung. Konkrete Informationen, wie Sie bei Problemen mit der Komponentenerkennung umgehen, finden Sie in den oben genannten und weiteren Kapiteln im Handbuch.

4.3 SmartIDs - direkte Komponentenadressierung

Seit QF-Test 7.0 bieten SmartIDs offiziell die Möglichkeit, Komponenten ohne Aufnahme eines Komponente Knoten zu referenzieren.

Für gewisse Anwendungen kann dies die Verwaltung und Pflege der Komponenteninformationen stark vereinfachen.

Auch auf die Editier- und Lesbarkeit der Tests können SmartIDs positiven Einfluss haben.

Nicht zuletzt bietet dies die Möglichkeit, Tests auch ohne Aufnahmefunktion zu erstellen, zum Beispiel, wenn eine Komponente oder die gesamte Anwendung noch gar nicht vorhanden ist, man den Test aber bereits implementieren möchte ("Test first" Ansatz).

Als Wiedererkennungskriterien stehen die Komponentenklasse, deren Name oder Beschriftung und der Index zur Verfügung. Die Werte sind die gleichen wie beim aufgenommenen Komponente Knoten. Zudem können auch Komponentenhierarchien abgebildet werden.

Eine SmartID wird an Stelle der QF-Test ID der Komponente verwendet. Ihr Kennzeichen ist die Raute `#` als erstes Zeichen. Anschließend kann der Name oder die Beschriftung der Komponente geschrieben werden, zum Beispiel:

- `#btnOK`, mit "btnOK" als Namen der Komponente oder
- `#Vorname`, mit "Vorname" als Beschriftung der Komponente.

Der Nachteil dieser einfachen Form der SmartID kann die Performanz bei der Wiedergabe sein, da QF-Test alle Komponenten nach diesen Kriterien durchsuchen muss. Aus diesem Grund ist es sinnvoll, die Klasse der Komponente mit anzugeben. Obige SmartIDs könnten dann so aussehen:

- `#Button:btnOK`
- `#TextField:Vorname`

Aktuell ist die Aufnahme von SmartIDs nicht als Standard aktiviert, kann aber direkt über das Menü eingeschaltet werden.

Nehmen Sie nun SmartIDs auf, indem Sie

Aktion

- im Menü **Aufnahme** die Einstellung **Aufnahme von SmartIDs** anhaken.
- **Aktivieren Sie den Aufnahmemodus** über "Aufnahme starten" ● .
- **Klicken Sie auf ein Textfeld**, z.B. das Eingabefeld für den Rabatt,
- **Klicken Sie auf eine Tabellenzelle**, z.B. das erste Modell.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .

Die aufgenommenen Mausklicks finden Sie im Bereich Extrasequenzen.

Die SmartID für das Eingabefeld lautet `#TextField:name=DiscountValue`, da der Name bevorzugt für die SmartID verwendet wird, wenn Name und Beschriftung vorhanden sind.

Der zweite Mausklick zeigt die Aufnahme der Tabelle als SmartID und daran anschließend einen Textindex für die Spalte, eingeleitet mit `@` und einen numerischen Index für die Zeile, eingeleitet mit `&`, wie Sie dies im vorletzten Abschnitt kennengelernt haben: `#Table:name=VehicleTable@Modell&0`.

Aktion

- Wenn Sie nicht weiter aktiv mit SmartIDs arbeiten möchten, können Sie diese in der Aufnahme-Option wieder deaktivieren.

Sie können aber auch gerne weiter mit SmartIDs arbeiten. Dabei werden Sie bei Aufnahmen auch weitere Ausdrücke in SmartIDs finden. Einige wollen wir hier zeigen und die Hintergründe dazu erläutern.

Zum Beispiel werden Sie bei einer Beschriftung, die mit einem Doppelpunkt endet, vor diesem Doppelpunkt einen Rückstrich sehen:

- `#TextField:Vorname\:`

Der Hintergrund ist, dass ein Doppelpunkt den davor stehenden Text als Komponentenklasse kennzeichnet. Daher müssen Doppelpunkte im Namen oder der Beschriftung mit einem Rückstrich geschützt werden. Dies gilt auch für die Sonderzeichen, die Indizes einleiten, also `@`, `&` und `%`.

Vor Beschriftungen werden Sie einen Kennzeichner, hier `left=` sehen:

- `#TextField:left=Vorname`

Hintergrund: Eine SmartID mit Angabe von Klasse und Namen der Komponente erreicht eine gleich gute Performanz bei der Wiedergabe wie die Verwendung von Komponente Knoten. Bei Beschriftungen sieht dies jedoch anders aus. Hier gibt es unterschiedlichste Möglichkeiten, was die beste Beschriftung für eine Komponente darstellt. QF-Test sucht unter den für eine Komponente in Frage kommenden Beschriftungen auf Basis der Komponentenklasse, Lage und Abstand der Beschriftungen die Beste aus. Für die Performanz bei der Wiedergabe ist es daher hilfreich, wenn direkt angegeben wird, nach welcher Beschriftung gesucht werden soll.

`left=` gibt hierbei an, dass die Beschriftung links der Komponente zu finden ist. Weitere Lagebezeichnungen in einer SmartID sind `right=`, `top=`, `topleft=` und `bottom=`. Wenn die Beschriftung der Text der Komponente ist, lautet der Kennzeichner `text=`, für den Tooltip `tooltip=`.

Wenn Komponenten mit dem gleichen Namen oder der gleichen Beschriftung mehrmals auf einer Anzeige vorhanden sind, können Sie auf SmartIDs stoßen, bei denen zwei über das Verbindungszeichen `@` zu einer vereinigt werden:

- `#TitledPanel:Kundenadresse@#TextField:left=Vorname`
- `#TitledPanel:title=Rechnungsadresse@#TextField:left=Vorname`

Im Beispiel gibt es die Beschriftung ”Vorname” sowohl in der Kachel ”Kundenadresse” als auch in der Kachel ”Rechnungsadresse”. Über die geschachtelte SmartID kann die Eindeutigkeit hergestellt werden.

Das zweite Beispiel ist etwas performanter wegen `title=` bei der SmartID bei der Kachel. Dafür ist die Lesbarkeit beim ersten etwas besser. Es hängt von der Anwendung ab, wie sehr auf Performanz geachtet werden muss. Bei Web-Anwendungen mit sehr vielen geladenen GUI-Elementen ist dies typischerweise relevant. Bei Java-Anwendungen kann man häufig der Lesbarkeit den Vorrang geben.

Bei langen Beschriftungen kann ein regulärer Ausdruck zur Verkürzung verwendet werden:

- `#%Dialog:Information.*@#Button:OK`

Das Prozentzeichen direkt nach der Raute gibt an, dass entweder der Name oder die Beschriftung als regulärer Ausdruck zu interpretieren ist. Im Beispiel wird der Titel verkürzt. Außerdem gilt die SmartID für jeden OK-Button, der in einem Dialog liegt, dessen Titel mit ”Information” beginnt. Eine genauere Beschreibung regulärer Ausdrücke finden Sie im Handbuch - reguläre Ausdrücke. Weitere Informationen zu Komponenten und SmartIDs finden Sie im Handbuch - Komponenten.

Kapitel 5

Benutzen des Debuggers (Java)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Benutzen des Debuggers'
https://www.qftest.com/de/yt/tutorial-5.html

In diesem Kapitel lernen Sie, wie der in QF-Test integrierte intuitive Debugger benutzt wird. Diejenigen unter Ihnen, die bereits Erfahrungen mit anderen IDEs, wie z.B. Eclipse haben, werden hier Ähnlichkeiten in Funktion und Nutzen des Debuggers feststellen.

Wir werden uns mit den folgenden Debugger-Funktionen beschäftigen:

- Setzen eines Breakpoints[55] mittels Strg-F8 (⇧-⌘-B auf macOS).
- Testausführung pausieren[64] mittels Pausetaste oder der Tastenkombination Alt-F12.
- Schrittweise Ausführung[56] mittels "Einzelschritt ausführen" , "Gesamten Knoten ausführen" und "Bis Knotenende ausführen" .
- Knoten überspringen[58] mittels "Knoten überspringen" und "Aus Knoten herausspringen" .
- Debug-Modus bei Fehler oder Exception aktivieren[60].
- Fehlerbehebung aus dem Protokoll heraus[62].
- Den aktuellen Fehler im Protokoll direkt anspringen über Strg-J. (Ins Protokoll springen in Abschnitt 5.5[62]).

Hinweis

Anstatt über die Schaltflächen können die Befehle auch über die Menüzeile oder Tastaturkürzel abgesetzt werden. Die Kürzel stehen neben den Optionen in den QF-Test Menüs, sofern vorhanden. Eine vollständige Übersicht der von QF-Test verwendeten

Tastaturkürzel finden Sie im Anhang Tastaturkürzel im Handbuch. Dort findet sich auch ein kleiner Helfer für die Funktionstastenbelegung von QF-Test zum Befestigen an der Tastatur.

Es gibt noch einige weitere Debugger-Funktionen wie

- "Aktuellen Knoten finden" (Aktuellen Knoten finden in Abschnitt 6.3[74]),
- "Ausführung hier fortsetzen" über das Popup-Menü des entsprechenden Knotens (Abbildung 6.9[77]),
- die "Exception erneut werfen" ,
- die Tabelle der Variablendefinitionen (Abschnitt 6.3[74]),

auf die wir in späteren Kapiteln eingehen werden.

5.1 Setzen eines Breakpoints

Zunächst einmal soll der Debugger aktiviert werden. Dies kann auf mehrere Arten erfolgen, zum Beispiel durch das Setzen eines Haltepunktes (Breakpoint) vor Start des Testlaufs. Der Zweck eines Haltepunktes ist es, den Testlauf an einer Stelle, die man näher untersuchen möchte, zu unterbrechen. Sobald QF-Test auf den Breakpoint trifft, wird die Testausführung pausiert und der Debugger-Modus aktiviert. Der Pauseknopf ist nun gedrückt.

Aktion

- Selektieren Sie einen Knoten und **drücken Sie Strg-F8 (⇧-⌘-B auf macOS)**. Der Haltepunkt wird durch das Symbol B kenntlich gemacht.

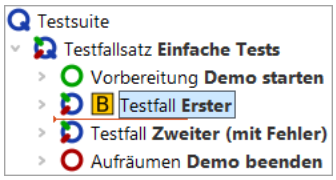

Abbildung 5.1: Breakpoint setzen

Aktion

- Selektieren Sie den Testsuite Knoten und starten Sie den Testlauf über die Taste Enter.

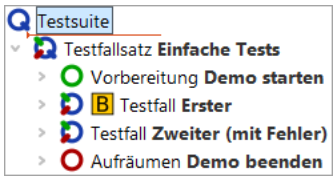

Abbildung 5.2: Testlauf starten

Aktion

- Löschen Sie den Breakpoint wieder, indem Sie nochmals **Strg-F8** (**⇧-⌘-B** **auf macOS) drücken**.

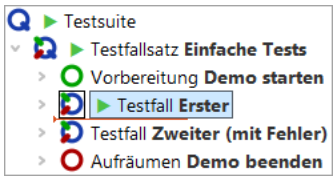

Abbildung 5.3: Breakpoint löschen

Man kann einen Breakpoint nicht nur über das Tastaturkürzel Strg-F8 (⇧-⌘-B auf macOS) sondern auch über den Menüpunkt Debugger→Breakpoint an/aus oder alternativ durch Rechtsklick auf den Knoten und Auswahl von Breakpoint an/aus im Popup-Menü setzen bzw. löschen. Für die weiteren Debugger-Funktionen werden wir hauptsächlich die jeweiligen Schaltflächen nennen, aber auch hier stehen die anderen Varianten zur Verfügung.

Wieder sehen Sie den kleinen Pfeil, der diesmal anzeigt, welcher Knoten als nächster ausgeführt wird. Dieser Knoten wird **aktueller Knoten** genannt. Bei der Aktivierung des Debug-Modus navigiert QF-Test zum aktuellen Knoten, falls dieser nicht bereits sichtbar ist, und selektiert diesen, d.h. die Zeile wird blau hinterlegt.

Das Kommando Debugger→Alle Breakpoints löschen ist ebenfalls nützlich, wenn alle Haltepunkte in allen geöffneten Testsuiten gelöscht werden sollen.

Es gibt keine Beschränkung für die Anzahl an Breakpoints, die Sie in Ihrer Testsuite setzen können. Haltepunkte werden beim Schließen der Testsuite nicht mit abgespeichert.

5.2 Schrittweise Ausführung

Nun wollen wir die Testfälle schrittweise ausführen.

Aktion

- Bitte experimentieren Sie ein wenig mit **"Einzelschritt ausführen"** , **"Gesamten Knoten ausführen"** und **"Bis Knotenende ausführen"** .

Wie Sie sicher festgestellt haben werden, öffnet **"Einzelschritt ausführen"** einen Knoten mit Kindern und macht den ersten Kindknoten zum aktiven Knoten. Dies ist wie immer an der Pfeilmarkierung des Knotens erkennbar.

Wenn Sie an dem Punkt weitergemacht haben, an dem die Ausführung der Testsuite im letzten Abschnitt pausiert war, d.h. vom Knoten "Testfall: Erster" aus, so würde nun der Testfall geöffnet werden:

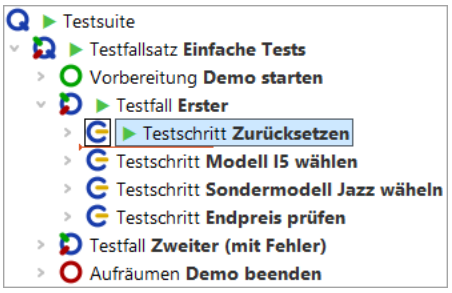

Abbildung 5.4: Einzelschritt ausführen

Im Falle eines Blattknotens, d.h. eines Knotens, der keine Kinder hat, ist die Wirkung die gleiche wie die der folgenden Funktion.

Mittels der Schaltfläche **"Gesamten Knoten ausführen"** wird ein Knoten inklusive aller Kindknoten ausgeführt. Der als nächstes auszuführende Knoten auf der gleichen Ebene wird dann der aktive und erhält den Pfeil.

Abbildung 5.5: Gesamten Knoten ausführen

"Bis Knotenende ausführen" führt die verbleibenden Geschwisterknoten aus und stoppt beim nächsten auszuführenden Knoten der übergeordneten Hierarchieebene.

Abbildung 5.6: Bis Knotenende ausführen

Im Beispiel ist dies der Aufräumen Knoten. Wie bereits im ersten Kapitel Ein vollständiger Testlauf[15] erläutert, haben Vorbereitung und Aufräumen Knoten die besondere Eigenschaft, dass sie vor und nach **jedem Testfall** ausgeführt werden, um so einen definierten Anfangszustand für jeden Testfall herzustellen.

Hinweis Dieses Verhalten tritt nur auf, wenn Sie die komplette Testsuite oder den Testfallsatz gestartet hatten und sich im Debug-Modus befinden. Wenn keine Testausführung aktiv war und Sie nur den Testfall selektiert hatten, so bewirkt die Funktion "Gesamten Knoten ausführen", dass der Testfall ausgeführt wird und dann der nächste Testfall selektiert wird.

Aktion

- Führen Sie die Aufräumen und Vorbereitung Knoten aus, indem Sie mit Hilfe der Schaltfläche die **gesamten Knoten ausführen** und dann über **den zweiten Testfall öffnen**. Dies ist eine Vorbereitung für das nächste Kapitel, in dem es um das Überspringen von Knoten geht.

Hinweis Bitte beachten Sie bei der interaktiven Testerstellung bei Menüs und Comboboxen, dass diese häufig zuklappen, wenn die Applikation den Fokus verliert, was beim Wechsel in den Debug-Modus der Fall ist. In diesem Fall empfiehlt es sich, den Knoten, der ein Menü oder eine Combobox öffnet, und den Knoten, der die Auswahlaktion durchführt, gemeinsam auszuführen, also nicht zwischendurch in den Debug-Modus zu gehen. Dies kann man zum Beispiel dadurch erreichen, dass man nach dem Knoten, der die Auswahlaktion durchführt, einen Haltepunkt B setzt und bei Erreichen des Knotens, der das Menü oder die Combobox öffnet, die Testausführung durch Lösen der Pausetaste freigibt.

5.3 Knoten überspringen

Die "Überspringen" Funktionen erweitern die Fähigkeiten des Debuggers von QF-Test in einer Weise, die über den Funktionsumfang von Standardprogrammierumgebungen

hinausgeht. Wie der Name andeutet, erlauben die ”Überspringen” Operationen einen oder mehrere Knoten während des Testlaufs auszulassen, d.h. weiter zu springen ohne diese auszuführen. Dies kann aus verschiedensten Gründen sinnvoll sein. Sei es um schnell an eine gewisse Position in Ihrem Testablauf zu gelangen oder um einen aktuell zu einem Fehler führenden Knoten zu überspringen.

Am Schluss des letzten Abschnitts haben wir den ersten Testschritt im zweiten Testfall zum aktiven Knoten gemacht. Dies ist er Ausgangspunkt für unsere nächste Aktion:

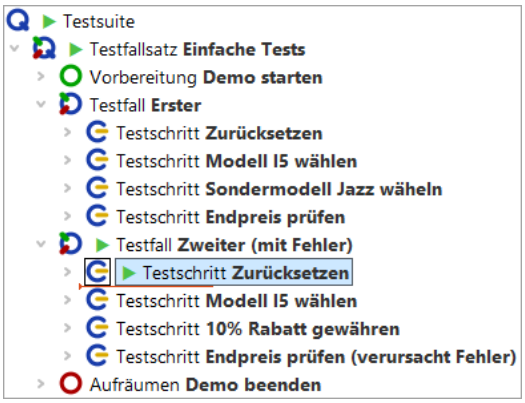

Abbildung 5.7: Testausführung am ersten Knoten des zweiten Testfalls pausiert

- Drücken Sie nun die Schaltfläche **”Knoten überspringen”** . QF-Test springt einfach über den aktiven Knoten ohne ihn oder seine Kindknoten auszuführen. Anschließend pausiert QF-Test beim nächsten auszuführenden Knoten auf der gleichen Ebene.

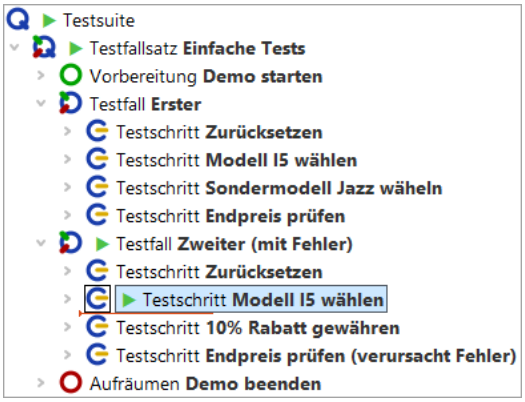

Abbildung 5.8: ”Knoten überspringen”

Aktion

- Und zuletzt die Schaltfläche **”Aus Knoten herausspringen”** . Sie sehen so-

fort, dass QF-Test aus dem Knoten, in dem Sie sich befinden, herausspringt ohne weitere Kindknoten auszuführen.

Abbildung 5.9: ”Aus Knoten herausspringen”

Hinweis Noch eine Bemerkung zu ”Knoten überspringen” und ”Aus Knoten herausspringen”: Benutzen Sie diese mit Vorsicht! Aus einer Sequenz herauszuspringen, bevor diese zu Ende gelaufen ist, kann dazu führen, dass Ihr SUT in einem Status belassen wird, auf dem andere Sequenzen oder Tests in der Suite nicht aufsetzen können.

5.4 Debug-Modus bei Fehler oder Exception aktivieren

Beim Debuggen eines Tests ist es oft hilfreich, wenn die Testausführung genau dann stoppt und in den Debug-Modus gewechselt wird, wenn ein Fehler, eine Exception oder manchmal auch nur eine Warnung auftritt.

Mittels dieser Technik werden wir in diesem Abschnitt und dem nächsten den zweiten Testfall debuggen.

Aktion

- Bitte **öffnen Sie das Debugger-Menü** und ändern Sie die Standardeinstellungen wie folgt:
- **Klicken Sie** auf den Menüpunkt **Debugger→Debugger aktivieren** um ihn zu aktivieren.
- **Klicken Sie** auf den Untermenüpunkt **Debugger→Optionen→Unterbrechen bei Fehler** um auch diese Funktion zu aktivieren.

Wenn Sie nun das Debugger-Menü und das Optionen-Untermenü wieder öffnen sollte es wie folgt aussehen:

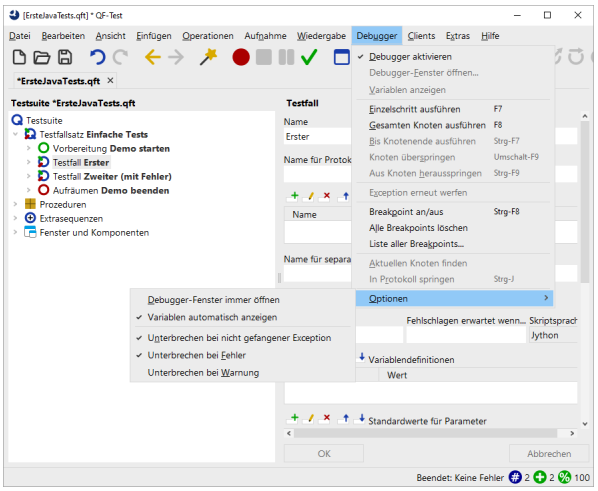

Abbildung 5.10: Debugger-Optionen: Test bei Fehler anhalten

Wir müssen die Debugger-Optionen ändern, da sonst der Test einfach durchlaufen würde, analog zu den vorherigen Beispielen aus Kapitel eins und zwei.

Aktion

- **Selektieren Sie den Testsuite Knoten** und starten Sie anschließend den Test mittels "Wiedergabe starten" ▶ .

QF-Test hält bei dem fehlerhaften Knoten an und wechselt in den Debug-Modus:

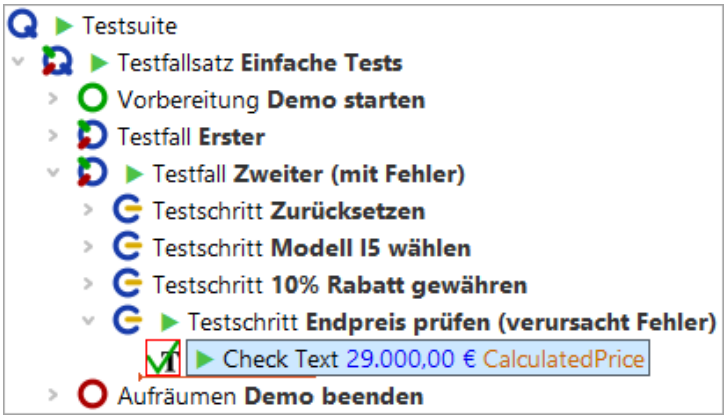

Abbildung 5.11: QF-Test pausiert bei Fehler

Der Knoten, der den Fehler verursacht hat, wird durch ein rotes Quadrat markiert. Außerdem erscheint ein Fehlerdialog, der uns Näheres zur Fehlerursache mitteilt. Über

diesen wechseln wir in das Protokoll, das wie so oft der Schlüssel zur Fehlerbehebung ist.

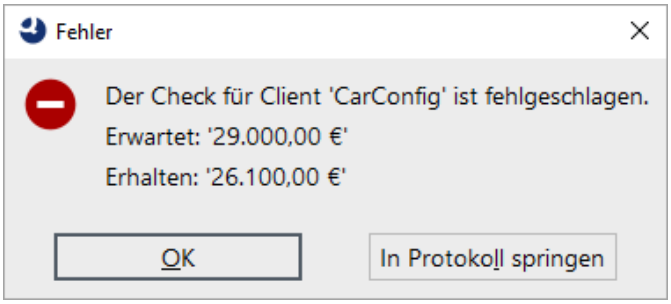

Abbildung 5.12: Fehlermeldung

Aktion

- **Klicken Sie** auf die Schaltfläche **In Protokoll springen** der Fehlermeldung.

5.5 Fehlerbehebung aus dem Protokoll heraus

Über die Schaltfläche **In Protokoll springen** (siehe Fehlermeldung in Abbildung Abbildung 5.12[62]) wird das Protokoll direkt bei dem entsprechenden Knoten geöffnet.

Neben der eigentlichen Fehlermeldung wurden etliche weitere Informationen zur Testumgebung zum Zeitpunkt des Fehlers protokolliert. Neben Bildschirmabbildern zum Fehlerzeitpunkt enthält das Protokoll unter dem Knoten, der den Fehler verursachte, eine Liste der gebundenen Variablen (Stacktrace). Auf die Nützlichkeit des Stacktrace werden wir zu einem späteren Zeitpunkt eingehen (Die Variablendefinitionen-Tabelle[70]).

Beim vorliegenden Fehler wird der falsche Wert im Check Text Knoten der Testsuite erwartet. Zur Fehlerbehebung muss dieser durch den tatsächlich angezeigten ersetzt werden. Dies geht bei einem Check mit festem Wert, um den es sich hier handelt, am einfachsten, indem Sie

Aktion

- auf den rot umrandeten Fehler-Knoten **"Fehlgeschlagen: Check Text: default ..." rechtsklicken** und
- im Kontextmenü **Check-Knoten mit erhaltenen Daten aktualisieren** **auswählen**.

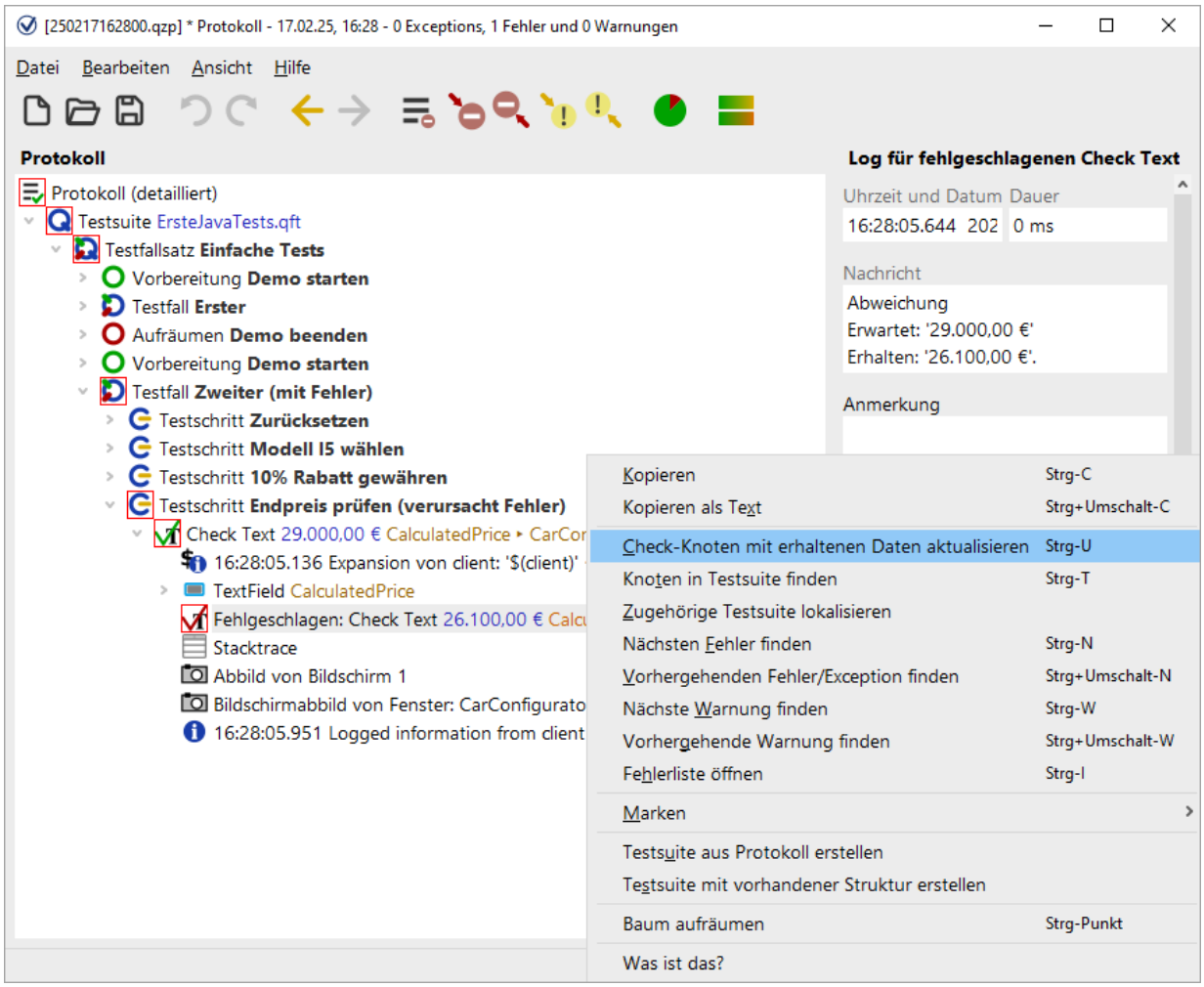

Abbildung 5.13: Check-Knoten mit erhaltenen Daten aktualisieren

QF-Test navigiert zum betroffenen Check Text Knoten in der Testsuite und aktualisiert den Wert des Attributs Text anhand der aus dem SUT ausgelesenen Daten.

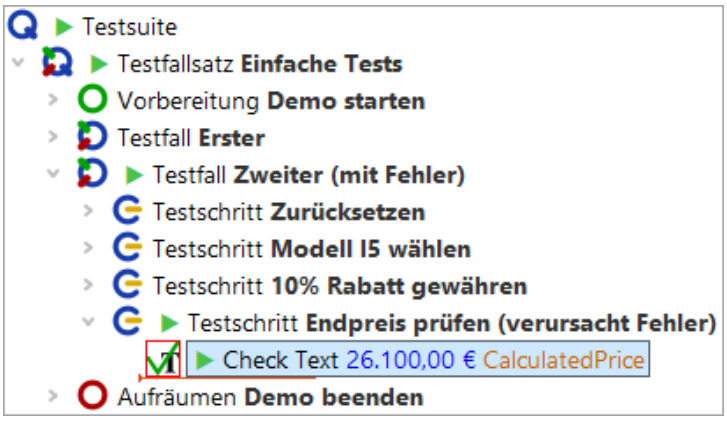

Abbildung 5.14: Korrigierter Check-Knoten

Der Knoten enthält nun zwar den korrekten Wert, ist aber immer noch rot umrandet, da er noch nicht wieder ausgeführt wurde. Dies wollen wir nun tun.

Aktion

- führen Sie den Test also fort, indem Sie auf den Pauseknopf ⏸ drücken und so **die Pause lösen**.

QF-Test führt den Rest der Testsuite aus. In unserem Fall sind das der Check Text und der Aufräumen Knoten. Anschließend informiert Sie QF-Test, dass ein Fehler aufgetreten ist. Diesen haben wir allerdings bereits während des Testlaufs behoben.

Ins Protokoll springen: Falls Sie das Protokoll an der Stelle, an der sich die Testausführung gerade befindet, öffnen wollen, brauchen Sie aus dem Debug-Modus heraus nur den Menüpunkt **Debugger→In Protokoll springen** anzuklicken oder das Tastaturkürzel Strg-J zu drücken. Wenn Sie einfach nur das Protokoll öffnen wollen, ohne an die aktuelle Stelle zu springen, steht Ihnen Strg-L zur Verfügung, was auch nach Ende des Testlaufs weiterhin funktioniert.

5.6 Testausführung pausieren

Wenn ein Test gerade ausgeführt wird und Sie den Debug-Modus aktivieren wollen, so können Sie entweder schnell einen Breakpoint auf einen noch nicht ausgeführten Knoten setzen oder Sie drücken einfach die Schaltfläche "Pause" ⏸ um den Debug-Modus sofort zu aktivieren.

Um die Ausführung fortzusetzen lösen Sie den Pauseknopf ⏸ , völlig unabhängig von der Art und Weise wie der Debug-Modus aktiviert wurde.

Wir wollen die vorhandenen Testfälle im Verlauf dieses Tutorials weiter verwenden. Allerdings wurde der Fehler in dem zweiten Testfall jetzt behoben. Insofern macht es Sinn ”(mit Fehler)” aus dem Namen des zweiten Testfalls zu löschen, genauso wie ”(verursacht Fehler)” aus dem Namen des Testschritts.

Es gibt manchmal Situationen, in denen das SUT ständig den Fokus für sich beansprucht. Dann kann es schwierig sein, das QF-Test Fenster lange genug im Vordergrund zu halten, um die Pausetaste drücken zu können. In einem solchen Fall steht Ihnen die ”Keine Panik”-Taste Alt-F12 zur Verfügung. Sie unterbricht alle laufenden Tests sofort. Zur Weiterführung des Tests können Sie diese Tastenkombination erneut drücken.

Kapitel 6

Variablen und Prozedurparameter (Java)

In diesem Kapitel lernen Sie, wie man eine Prozedur einsetzt um die gleichen Schritte auf unterschiedlichen Daten auszuführen. Außerdem sehen Sie, wie man Variablen einsetzt. Ebenso wird die Fehleranalyse in Bezug auf Variablen behandelt.

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Variablen und Prozedurparameter'
https://www.qftest.com/de/yt/tutorial-6.html

6.1 Prozedur mit Variable

Sehen Sie sich den letzten Testschritt "Endpreis prüfen" in unseren beiden Testfällen an.

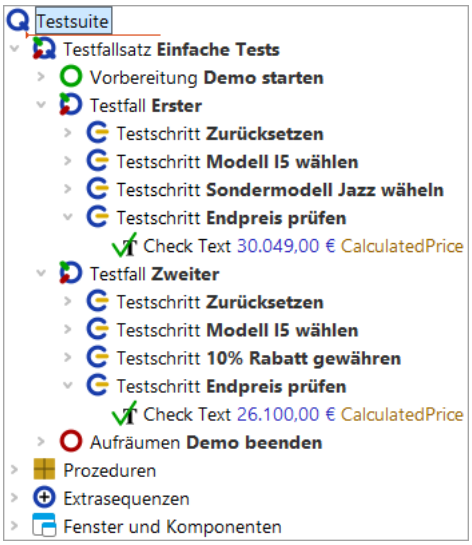

Abbildung 6.1: Zwei fast gleiche Testschritte

Es wird der gleiche Schritt ausgeführt, jedoch mit unterschiedlichen Daten. Auch wenn es sich nur um einen Schritt handelt, macht es Sinn eine Prozedur daraus zu machen. Vielleicht kommen wir später auf die Idee, die hartkodierten Werte `30.049,00 €` und `26.100,00 €` in ein anderes Format zu bringen, so dass der Check auf das Feld "Endpreis" auch für andere Währungen funktioniert. Diesen Algorithmus zweimal zu implementieren wäre auf jeden Fall nicht sinnvoll.

Aktion

- **Selektieren Sie den "Check text" Knoten im ersten Testfall.**
- Wählen Sie den Menüpunkt **Operationen→Knoten einpacken in→Sequenz** aus oder verwenden Sie das Tastaturkürzel Strg-Umschalt-S um ihn in eine Sequenz einzupacken.
- **Nennen Sie die Sequenz 'prüfeEndpreis'.** Dieser Name entspricht der Java-Konvention die Wörter zusammenzuschreiben. Andererseits erlaubt QF-Test auch Leerzeichen in Prozedurnamen, so dass Sie der Java-Konvention nicht zu folgen brauchen.
- Drücken Sie **Strg-Umschalt-P** um auf kürzestem Weg den Sequenzknoten in eine Prozedur zu konvertieren (wie aus dem letzten Kapitel bekannt). Wie Sie sehen, wurde die Sequenz durch einen Prozeduraufruf von "prüfeEndpreis" ersetzt.
- **Klicken Sie doppelt auf den Prozeduraufruf**, um zur Prozedur im Prozeduren Knoten zu springen.
- **Öffnen Sie** den neu erstellten Prozedurknoten um den Inhalt zu sehen.

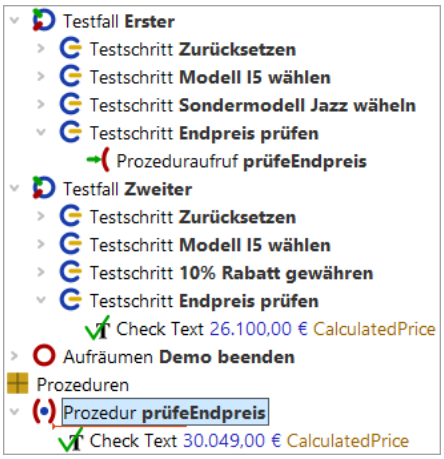

Abbildung 6.2: Prozedur mit hartkodiertem Wert

Wie erwartet befindet sich der "Check text" Knoten in der Prozedur. Er ist jedoch nur für einen einzigen Preis gültig, nämlich `30.049,00 €`. Da wir die gleiche Prozedur auch für den zweiten Testfall verwenden möchten, müssen wir den Preis durch eine Variable ersetzen. Der Wert dieser Variable sollte dann beim Prozeduraufruf mit übergeben werden.

Im nächsten Beispiel werden wir einen Parameter mit Standardwert im Prozedurknoten einfügen. Standardwerte werden häufig verwendet, wenn der entsprechende Parameter bei den meisten Prozeduraufrufen den Standardwert erhalten würde. Dann braucht man den Standardwert nicht jedes Mal zu spezifizieren, sondern kann auf den im Prozedurknoten definierten Wert zurückgreifen. Obwohl das auf den Preis als Parameter nicht zutrifft, können wir ihn gerade deshalb verwenden um zu zeigen, wie ein Standardwert funktioniert und wie man ihn bei Bedarf mit einem anderen Wert überschreiben kann.

Als erstes fügen wir also eine Variable mit Standardwert ein:

Aktion

- **Selektieren Sie die Prozedur 'prüfeEndpreis'**
- **Drücken Sie den** "Zeile einfügen" Knopf über der Tabelle "Standardwerte für Parameter".
- **Tragen Sie `preis`** als Namen des Parameter ein.
- **Tragen Sie `30.049,00 €`** als Wert ein.
- Drücken Sie **OK**.

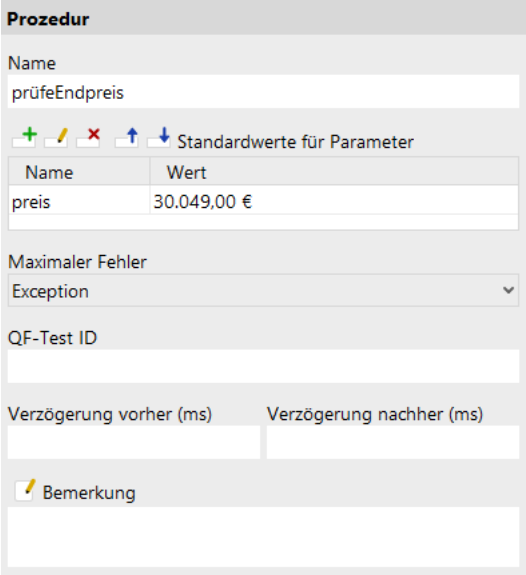

Abbildung 6.3: Die Details eines Prozedurknotens

Im nächsten Schritt ersetzen wir den Wert des Text Attributs des Check Text Knotens durch eine Referenz auf die Variable.

Hinweis

Variablensyntax: Wenn Sie mit Variablen arbeiten, ist es wichtig, sich bewusst zu sein, dass Sie an bestimmten Stellen QF-Test mitteilen wie, eine Variable heißt und an anderen, dass QF-Test auf den Wert einer Variablen zugreifen soll.

In obigem Beispiel wird QF-Test in der Namensspalte für die Standardwerte der Variablenname mitgeteilt. In diesem Fall brauchten Sie nur `preis` einzutragen.

Im Text Attribute des Check Text Knotens soll der Wert der Variablen verwendet werden. Bei QF-Test geschieht dies dadurch, dass Sie den Variablenname in `$()` setzen, hier `$(preis)`. Falls Sie den Variablennamen nicht in `$()` setzen, würde QF-Test den Preis mit der Zeichenfolge `preis` vergleichen.

Aktion

- **Selektieren Sie den Check Text Knoten** in der Prozedur ”prüfeEndpreis”.
- **Tragen Sie `$(preis)`** im Text Attribut der Check Text Knotendetails ein.
- **Drücken Sie ’OK’** in den Knotendetails.

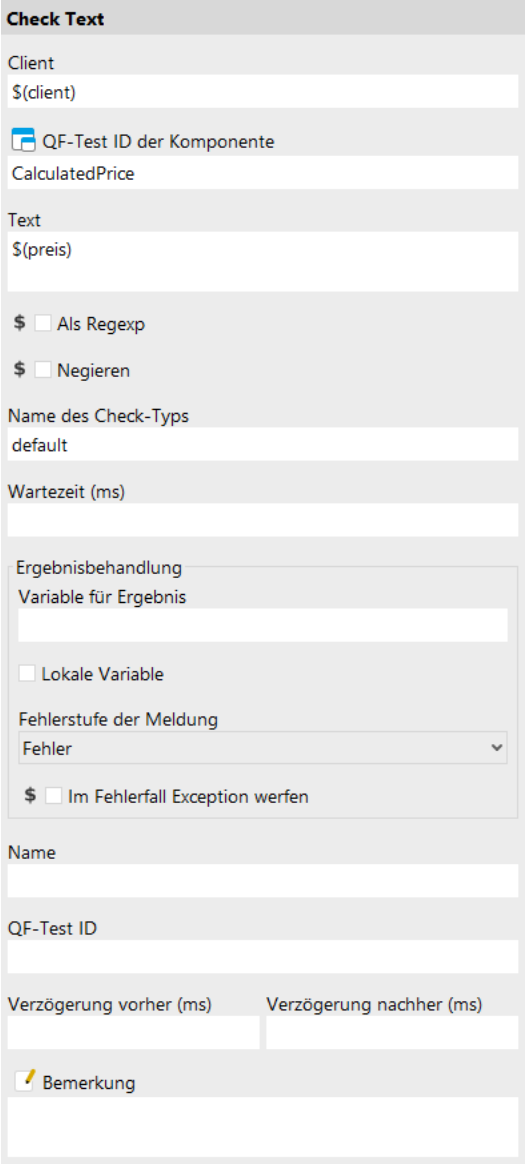

Abbildung 6.4: 'Check text'-Knoten

Aktion

- **Führen Sie den ersten Testfall aus.**

Der Testfall sollte fehlerfrei durchlaufen.

6.2 Die Variablendefinitionen-Tabelle

Im nächsten Schritt fügen wir einen Prozeduraufruf im zweiten Testfall ein.

Aktion

- **Ersetzen Sie den Check Text Knoten** des zweiten Testfalls **durch einen Prozeduraufruf von "prüfeEndPreis"**. Sie können einfach den Prozeduraufruf aus dem ersten Testfall kopieren oder den Prozeduraufruf wie oben beschrieben einfügen.

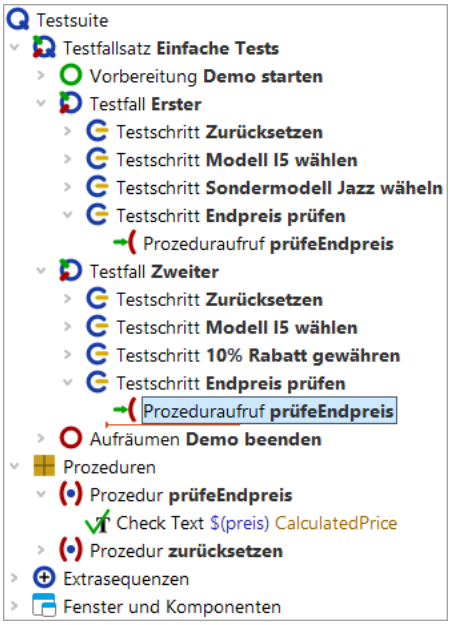

Abbildung 6.5: Prozeduraufruf von "prüfeEndPreis" in der zweiten Prozedur

Hinweis

Wenn im Prozeduraufruf bereits der Preis mit Standardwert eingetragen ist, rührt das daher, dass der Prozeduraufruf mit Hilfe der Prozedur selbst erzeugt wurde. Entweder durch Kopieren der Prozedur oder durch eine Drag-and-Drop Aktion unter Verwendung des Prozedurknoten oder über direktes Einfügen der Prozedur wie weiter oben erläutert. Aktuell geht es jedoch um den Standardwert. Daher bitten wir Sie, den evtl. vorhandenen Preis-Parameter zu löschen, wenn Sie dem Tutorial Schritt für Schritt folgen wollen. Dazu klicken Sie das rote X über der Variablendefinitionen-Tabelle.

Aktion

- Überprüfen Sie, ob der **Debugger so eingestellt ist, dass er bei Fehlern unterbricht** (siehe Debugger-Optionen: Test bei Fehler anhalten[61]).
- **Selektieren Sie den Knoten "Testfall: Zweiter"**.
- **Führen Sie ihn aus**, entweder über den Knopf ▶ oder durch Drücken der Eingabe Taste.

Eine Fehlermeldung zeigt an, dass ein anderer als der angezeigte Preis erwartet wurde. Was lief schief? Lassen Sie uns nach dem Fehler forschen. Üblicherweise würden wir ins Protokoll schauen, aber es gibt noch eine andere wichtige Informationsquelle.

Aktion

- **Klicken Sie OK**, um die Fehlermeldung zu schließen.

Im Debugging-Modus sehen Sie rechts unten im QF-Test Fenster eine Knotenliste mit Variablen, die von diesen gebunden sind.

Aktion

- Eventuell müssen Sie die Variablendefinitionen-Tabelle vergrößern, um alle Einträge sehen zu können.

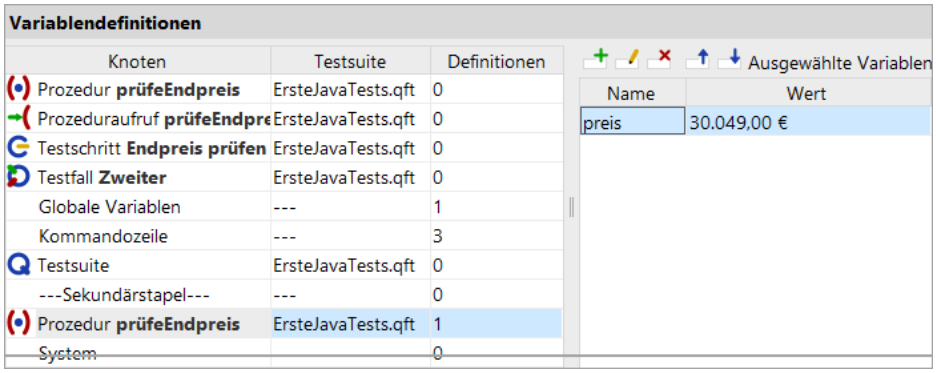

Abbildung 6.6: Variablendefinitionen

Die Variablendefinitionen-Tabelle ist beim Debuggen sehr hilfreich, da sie die aktuellen Werte der Variablen anzeigt. Sie unterstützt beim Arbeiten mit Prozeduren als auch beim Verständnis, wie QF-Test den richtigen Variablenwert ermittelt.

Hinweis

QF-Test geht die Variablendefinitionen-Tabelle immer von oben nach unten durch.

Sie sehen, dass in den ersten Zeilen keine Variablen gebunden sind. Auf der Ebene "Globale Variablen" ist eine Variable gebunden und auf dem Sekundärstapel für "Prozedur: prüfeEndpreis" eine weitere. Die globale Variable wird für die Verbindung zur SUT Applikation verwendet und wurde vor dem Anwendungsstart gesetzt. (vgl. Starten der Anwendung[5]). Die andere Variable interessiert uns im Moment mehr - sie hat jedoch den falschen Wert.

Dieser Wert auf dem Sekundärstapel ist der Standardwert, da er dann verwendet wird, wenn nirgendwo sonst einer Variablen mit dem gleichen Namen ein Wert zugewiesen wurde.

Um es richtig zu machen, müssen wir den korrekten Wert beim Prozeduraufruf an die Prozedur übergeben. Wieder gibt es mehrere Arten, dies zu tun. Ein Weg wäre, eine neue Zeile in der Variablendefinitionen-Tabelle in den Details des Prozeduraufrufs einzufügen, ähnlich wie beim Prozedurknoten im vorigen Abschnitt.

Wenn es jedoch bereits mehrere Prozeduraufrufe gibt, ist folgendes einfacher:

Aktion

- **Beenden Sie die laufende Testausführung** mittels ■ .

- Führen Sie einen **Rechtsklick auf den Prozedurknoten** aus und wählen **Weitere Knotenoperationen→Parameter von Referenzen anpassen** im Popup-Menü.

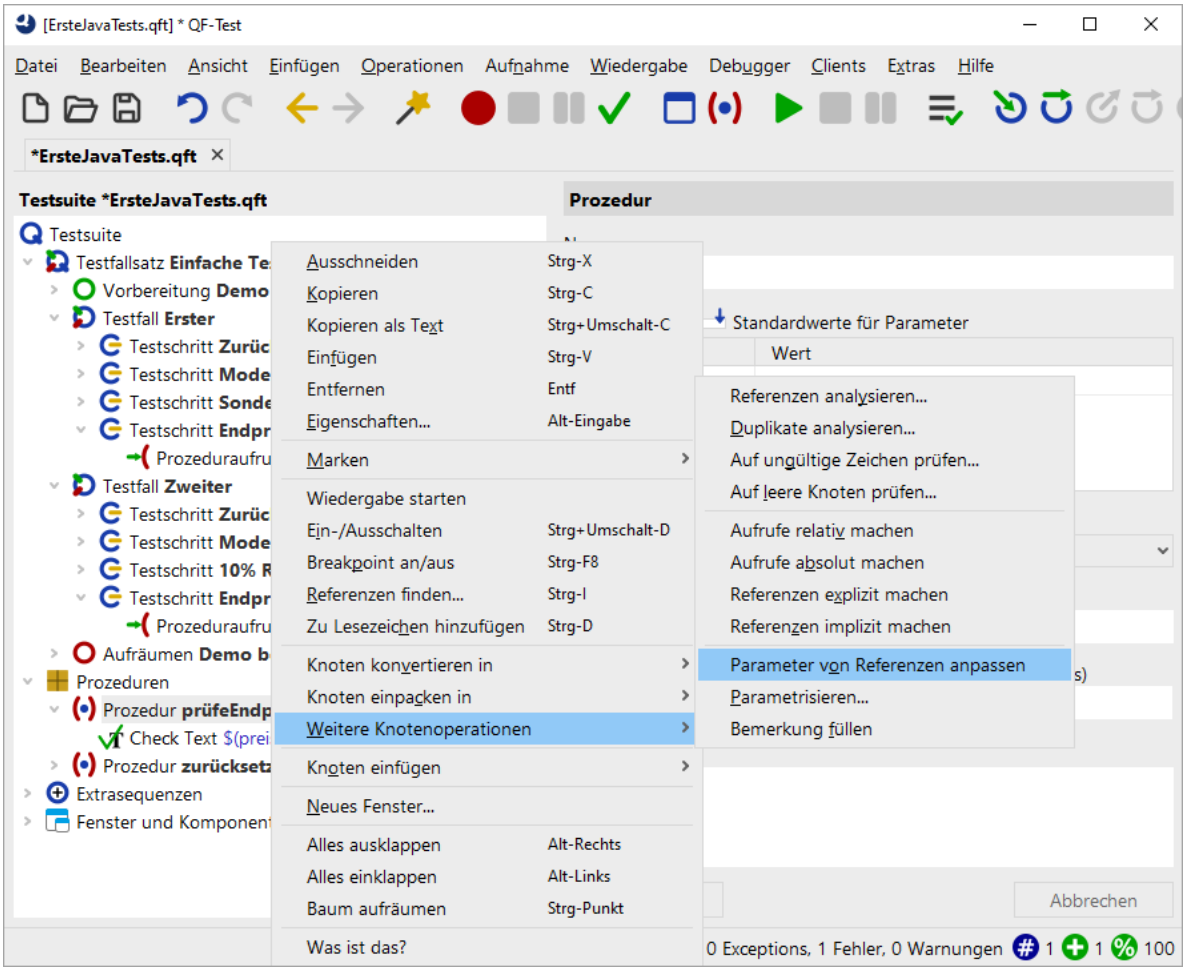

Abbildung 6.7: Popup-Menü für "Parameter von Referenzen anpassen"

- Prüfen Sie im folgenden Dialog, dass ein Häkchen bei **Fehlende Parameter beim Aufrufer hinzufügen** gesetzt ist, und **bestätigen Sie mit OK**.

Im Prozeduraufruf erzeugt QF-Test für jeden Standardwert eine Zeile in der Variablen Definitionen Tabelle. In unserem Fall wurde eine Zeile für den Parameter mit dem Namen `preis` und dem Wert `30.049,00` € hinzugefügt.

Auch damit wird es im zweiten Testfall noch nicht funktionieren, auch wenn der Wert direkt übergeben wird, weil es sich immer noch um den Standardwert handelt, der hier

nicht korrekt ist. Bitte verändern Sie den Wert noch nicht, damit wir Ihnen mittels des entstehenden Fehlers eine weitere Möglichkeit des Debuggens zeigen können.

Aktion

- **Schließen Sie den Dialog "Angepasste Knoten"**, den QF-Test anzeigt, um Sie über die vervollständigten Knoten zu informieren.

6.3 Fortgeschrittenes Debuggen mittels Variablendefinitionen-Tabelle

Als nächstes wollen wir die Variablendefinitionen-Tabelle unter die Lupe nehmen und herausfinden, wie man sie für Debugging-Zwecke einsetzen kann. Daher belassen Sie bitte den fehlerhaften Wert, der im vorigen Abschnitt im Prozeduraufruf eingefügt wurde.

Dazu soll die Ausführung des Testfalls beim Prozeduraufruf unterbrochen werden um dann mittels Einzelschritten in die Prozedur zu gehen. Dabei werden wir uns ansehen, was in der Variablendefinitionen-Tabelle passiert. Anschließend wollen wir direkt aus der Variablendefinitionen-Tabelle zum fehlerhaften Prozeduraufruf springen und dort den Parameterwert korrigieren.

Aktion

- **Setzen Sie einen Breakpoint** bei "Prozeduraufruf: prüfeEndpreis" im zweiten Testfall.
- **Führen Sie den zweiten Testfall aus**.
- Wenn QF-Test am Breakpoint anhält, **führen Sie zwei Einzelschritte in die Prozedur** mittels aus und **beobachten dabei die Variablendefinitionen-Tabelle**.

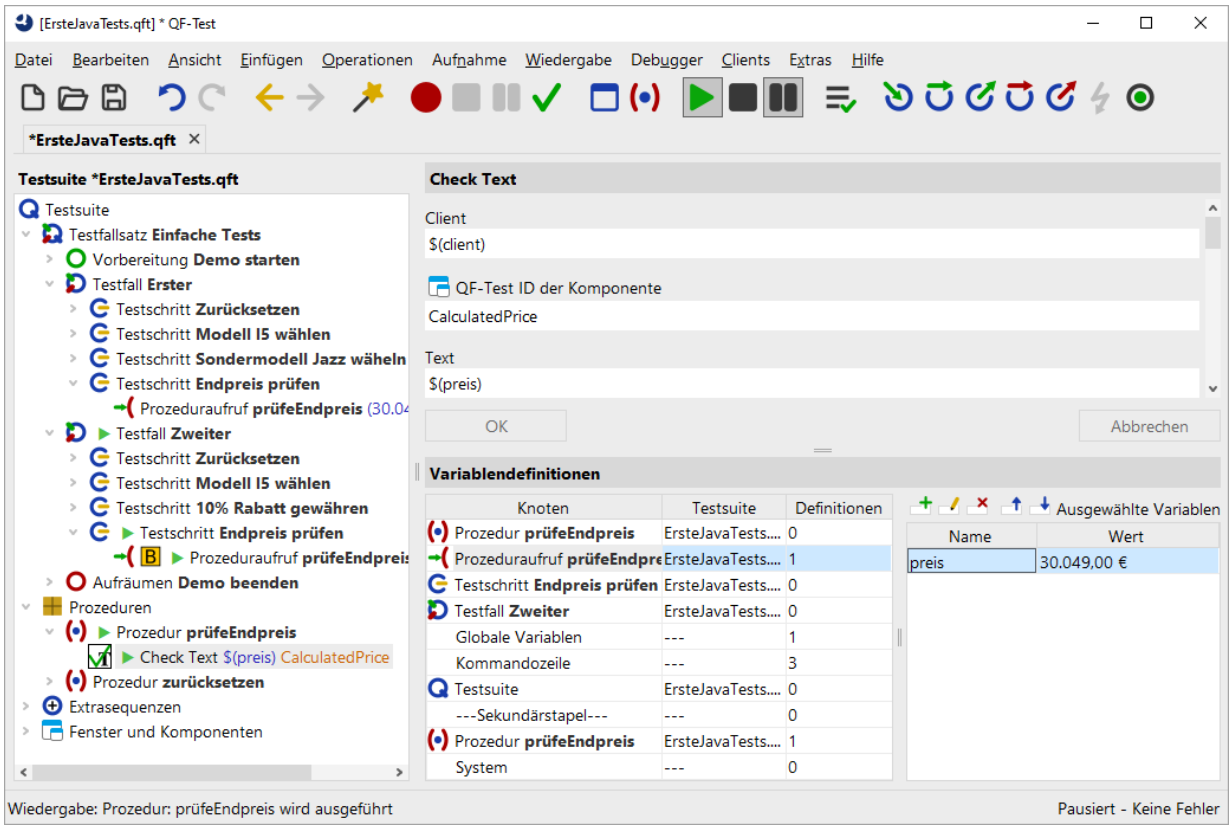

Abbildung 6.8: Variablendefinitionen-Tabelle zeigt den falschen Wert

Wenn Sie mit Einzelschritten in die Prozedur gehen, wird beim ersten eine Zeile für "Prozeduraufruf: prüfeEndpreis" oben in der Tabelle erzeugt und beim zweiten die Zeile "Prozedur: prüfeEndpreis".

Jetzt gibt es die Variable `preis` auf zwei verschiedenen Ebenen in der Variablen Definitionen Tabelle: in der Zeile für "Prozeduraufruf: prüfeEndpreis" und in der Zeile "Prozedur: prüfeEndpreis" auf dem Sekundärstapel, wobei keiner der beiden Variablenwerte der richtige ist.

In QF-Test können Sie interaktiv die Werte von Variablen in der Variablendefinitionen-Tabelle verändern, wenn Sie sich im Debugging-Modus befinden. Sie können sogar neue Variablen hinzufügen oder vorhandene löschen. Damit können Sie arbeiten, solange sich die Variablen auf dem Variablenstapel befinden, in unserem Fall solange wie die Prozedur ausgeführt wird.

Änderungen des aktuellen Variablenwertes in der Variablendefinitionen-Tabelle bewirken keine Anpassung des im Prozeduraufrufknoten eingetragenen Parameterwerts. Der Wert muss explizit im Prozeduraufruf geändert werden.

Die schnellste Methode um zum Prozeduraufruf zu gelangen ist ein Doppelklick auf die

Prozeduraufrufzeile (zweite Zeile) in der Variablendefinitionen-Tabelle. Diese Methode ist besonders hilfreich, wenn Sie umfangreiche Testsuiten debuggen und der Knoten, zu dem Sie springen wollen, nicht im Testsuite-Fenster angezeigt wird. Statt eines Doppelklicks können Sie auch einen Rechtsklick auf die Zeile ausführen und den Menüpunkt Zu Knoten in Testsuite springen wählen.

Aktion

- **Führen Sie einen Doppelklick auf die zweite Zeile mit dem Prozeduraufruf** in der Variablendefinitionen-Tabelle aus.
- **Setzen Sie den Parameterwert auf den richtigen Wert**, d.h. `26.100,00 €`.

Umgekehrt wird auch der aktuelle Wert in der Variablendefinitionen-Tabelle dadurch nicht verändert. Um dies zu erreichen müssen wir den Prozeduraufruf erneut ausführen. Allerdings ist die Testausführung über diesen Punkt bereits hinaus.

Hinweis

Daher wollen wir hier eine weitere nützliche Funktion des Debuggers zeigen, mit der man den QF-Test anweisen kann, den nächsten auszuführenden Knoten zu verändern. Dazu selektieren Sie den entsprechenden Knoten und wählen den Menüpunkt **Ausführung hier fortsetzen** oder verwenden das Tastaturkürzel Strg-,.

Also, um den neu gesetzten Wert auszuprobieren:

Aktion

- **Führen Sie einen Rechtsklick auf den Knoten "Prozeduraufruf: prüfeEndpreis"** in der zweiten Prozedur aus.
- **Wählen Sie "Ausführung hier fortsetzen"** im Popup-Menü.

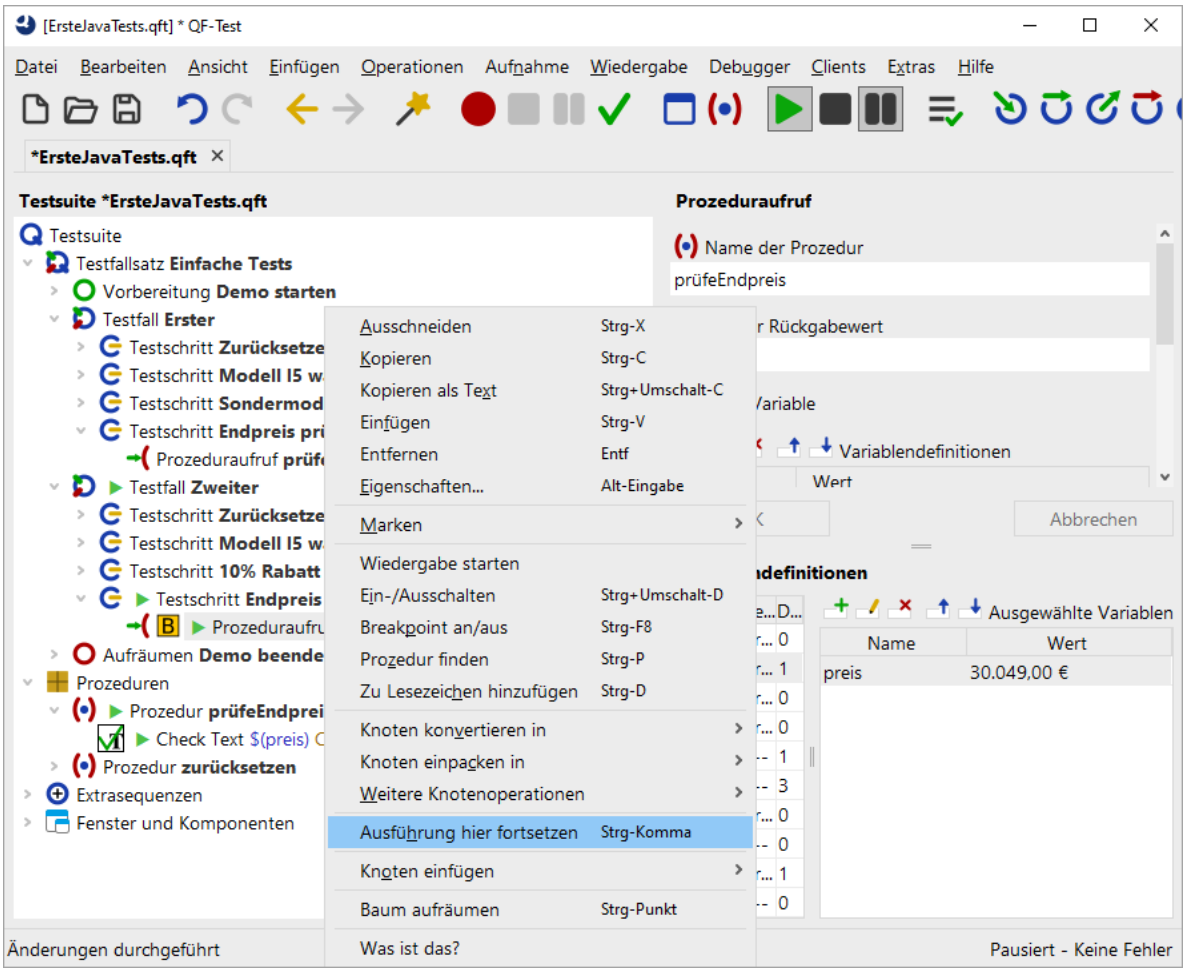

Abbildung 6.9: Ausführung hier fortsetzen

In der Variablendefinitionen-Tabelle sind die zwei obersten Zeilen verschwunden. Der Grund ist, dass Sie die Prozedur verlassen haben (wenn auch "rückwärts") und dass dadurch der Prozeduraufruf mit den daran gebundenen Variablen vom Aufrufstack genommen wurde.

Aktion

- Lösen Sie den Pauseknopf ⏸ .

Nun sollte kein Fehler mehr auftauchen.

Hinweis

Da die Variablendefinitionen-Tabelle äußerst hilfreich ist, wenn Sie nach fehlerhaften Variablenwerten fahnden, wird eine Kopie davon auch unter dem Knoten "Stacktrace" im Protokoll abgespeichert, in dem die Variablenwerte genau zum Zeitpunkt des Fehlers zu sehen sind.

Aktuellen Knoten finden: Manchmal entfernt man sich beim Debuggen ziemlich weit vom aktuellen Knoten und möchte anschließend wieder zu diesem Knoten zurückfinden.

Das geht am einfachsten indem man in der Toolbar "Aktuellen Knoten finden" drückt oder den Menüpunkt Debugger→Aktuellen Knoten finden wählt.

6.4 Variablen setzen

Zusätzlich zu den oben beschriebenen Wegen können die Variablen auch wie folgt gesetzt werden:

- Mittels Variable setzen Knoten,
- als Rückgabewert einer Prozedur,
- als Ergebnis eines QF-Test Knotens wie Text auslesen, Geometrie auslesen, Index auslesen und Check,
- in der 'Variablendefinitionen' Tabelle von Testsuite, Testfall, Testschritt, Sequenz und weiteren Knoten wie dem If oder Schleife Knoten,
- über Kommandozeilenparameter.

Informationen dazu, an welcher Stelle eine Variable am besten definiert wird, finden Sie im nächsten Abschnitt.

Ein Variable setzen Knoten kann über den Menüpunkt Einfügen→Diverse Knoten→Variable setzen eingefügt werden. In den Knotendetails können Sie angeben, ob es sich um eine lokale (Häkchen bei "Lokale Variable" setzen) oder eine globale Variable handeln soll.

Die folgende Abbildung zeigt die Details eines Variable setzen Knotens, den Sie als ersten Knoten im Vorbereitung Knoten finden. Es wird eine Variable mit dem Namen `client` definiert. Dass es sich um eine globale Variable handelt, erkennen Sie daran, dass das Attribut 'Lokale Variable' nicht gesetzt ist.

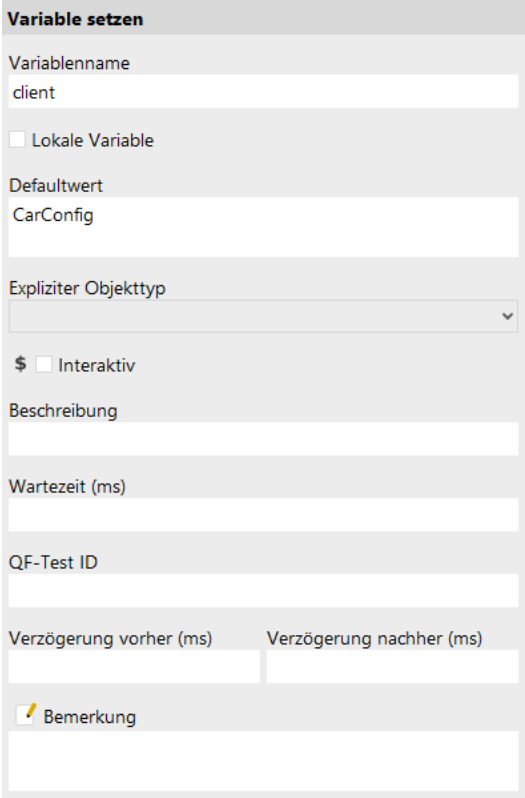

Abbildung 6.10: Details des Variable setzen Knoten

Wenn eine Variable mit dem Rückgabewert einer Prozedur gesetzt werden soll, geben Sie den Variablennamen im Attribut "Variable für Rückgabewert" des Prozeduraufrufs an. In der Prozedur selbst müssen Sie als letzten auszuführenden Knoten einen Return Knoten einfügen, der den betreffenden Wert zurückgibt.

Die Prozedur in der folgenden Abbildung liest den Rabattwert aus dem SUT und gibt den Wert an den aufrufenden Testfall zurück. Dort heißt die empfangende Variable `Rabatt` und ist als lokale Variable deklariert. Dieses Beispiel ist nicht in der Übungstestsuite enthalten.

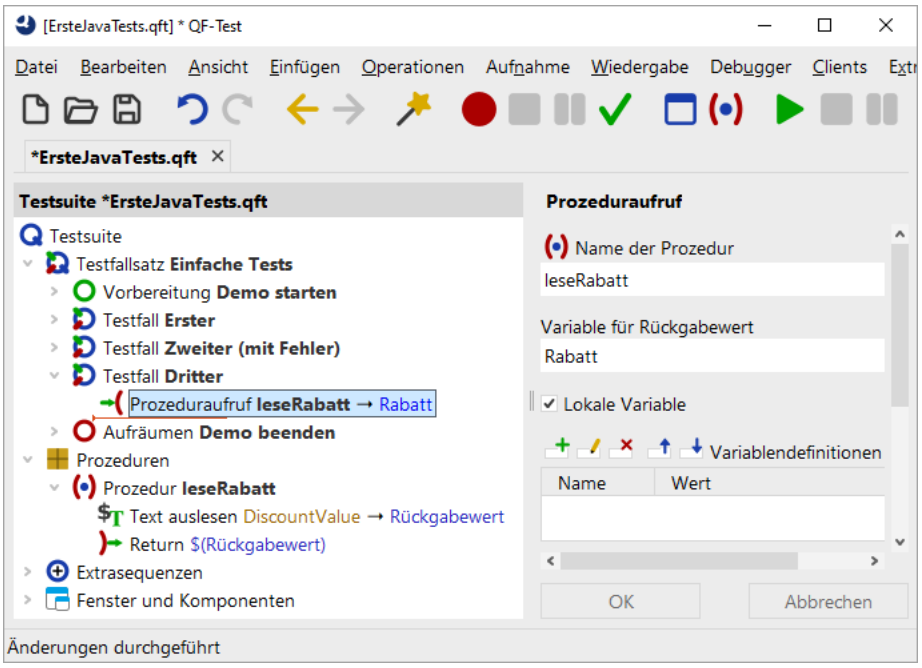

Abbildung 6.11: Prozedur mit Rückgabewert

Der Text auslesen Knoten in der obigen Abbildung ist einer der QF-Test Knoten, die direkt den Wert einer Variablen setzen. Dabei wird der Variablenname in dem entsprechend benannten Attribut eingetragen. Sie haben wiederum die Wahl, ob es eine lokale oder globale Variable werden soll.

Es gibt eine Reihe von Knoten, die eine Variablendefinitionen-Tabelle besitzen. Dort können Sie lokale Variablen setzen. Falls sich der betreffende Knoten in einer Prozedur befindet, wird die Variable als lokale Variable an die Prozedur gebunden, ansonsten als lokale Variable an den Testfall. Variablen, die mittels dieser Tabelle an den Testsuite Knoten gebunden sind, können von allen Knoten der Testsuite referenziert werden.

Alle Knoten, an die Variablen gebunden werden können, werden im Debugger-Modus im Variablen Definitionen Fenster rechts unten angezeigt, wenn sie gerade ausgeführt werden.

Variablen können auch über die Kommandozeile spezifiziert werden. Hierzu verwenden Sie den Kommandozeilenparameter `-variable`. Beispiel: `qftest -batch -variable "browser"="ie" test.qft`. Weitere Informationen hierzu finden Sie im Handbuch, Kapitel 'Kommandozeilenargumente'.

6.5 Ebenen für Variablendefinitionen

Hinweis Dieser Abschnitt gibt Antworten auf die Frage, auf welcher Ebene eine Variable definiert werden sollte. Wenn Sie diese Frage momentan nicht interessiert, können Sie direkt zum nächsten Kapitel springen.

Variablen können auf unterschiedlichen Ebenen gebunden werden:

- Im Testsuite Knoten,
- in Testfällen und Prozeduren als Standard- oder als lokale Variablen,
- als Parameter in einem Prozeduraufruf,
- als globale Variable und
- als Kommandozeilenparameter.

Die Ebene, auf welcher eine Variable am sinnvollsten definiert wird, hängt vom Verwendungszweck der Variablen ab:

Prozedurparameter
Übergeben Sie einen Wert als Parameter an eine Prozedur, wenn die gleiche Prozedur mehr als einmal und mit unterschiedlichen Werten ausgeführt werden soll. Prozedurparameter werden in der Variable Definitionen Tabelle eines Prozeduraufruf Knoten angegeben.

Lokale Variablen in einer Prozedur
Lokale Variablen werden innerhalb der Prozedur definiert und sind nur dort gültig. Wenn die Prozedur beendet wird, werden sie gelöscht. Verwenden Sie eine lokale Variable, wenn diese nicht außerhalb der Prozedur benötigt wird. Sie sind das Mittel der Wahl für Zwischenergebnisse.

Lokale Variablen in einem Testfall
In einem Testfall können lokale Variable entweder während der Durchführung des Testfall angelegt werden oder über die entsprechende Tabelle in den Details des Testfall Knotens. Wenn Sie in einem Testfall mehrfach den gleichen Wert verwenden, ist es sinnvoll, diesen einmalig einer Variablen zuzuweisen und dann die Variable zu verwenden. Dies erhöht die Wartbarkeit. Auch für Zwischenergebnisse sollte man lokale Variablen verwenden.

Globale Variablen
Wenn globale Variablen einmal erstellt wurden, existieren sie, bis sie entweder explizit gelöscht werden oder bis QF-Test beendet wird. Auch Stopps und die erneute Ausführung von Tests ”überleben” sie. Verwenden Sie sie für

Werte, die in mehreren Testfällen genutzt werden. Ein Beispiel ist die Variable `client`, die im Vorbereitung Knoten beim Start der Applikation angelegt wird. Um sie wieder loszuwerden, wählen Sie den Menüpunkt Wiedergabe→Globale Variablen löschen. Auch beim Beenden von QF-Test werden sie gelöscht.

Kommandozeilenparameter

Variablen, die über Kommandozeile gesetzt werden, sind im Batch-Modus sinnvoll, wenn Sie mehrere Batch-Läufe mit unterschiedlichen Werten durchführen wollen. Kommandozeilenparameter gelten während der gesamten Laufzeit des Batch-Laufs. Ein typisches Beispiel ist die variablengesteuerte Ausführung auf verschiedenen Browsern. Variablen können über den Kommandozeilen-Parameter `-variable` spezifiziert (vgl. Kapitel 'Kommandozeilenargumente' im Handbuch).

Testsuite-Variablen

Testsuite-Variablen können von allen Testfällen verwendet werden. Ihr Verwendungszweck entspricht dem von globalen Variablen, nur dass sie im Batch-Modus durch Variablen in der Kommandozeile überschrieben werden können.

Standardwerte (Sekundärstapel)

Sie können Standardwerte für die Variablen von Prozeduren, Testfällen und Testfallsätzen definieren. Diese kommen zum Zug, wenn keine Variable mit dem gleichen Namen auf einer höheren Ebene definiert wurde.

Kapitel 7

Die Standardbibliothek (Java)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Die Standardbibliothek'
https://www.qftest.com/de/yt/tutorial-7.html

QF-Test stellt eine gewisse Anzahl an Knotentypen bereit, die für die Testerstellung genutzt werden können. Wenn Sie Funktionalität benötigen, die darüber hinausgeht, können Sie diese mittels Skript-Knoten implementieren. Um Ihnen die Arbeit zu erleichtern, wurden viele Funktionen, die häufig benötigt werden, bereits in Prozeduren implementiert und werden in einer Standard-Prozeduren-Bibliothek mit QF-Test ausgeliefert.

Wenn Sie also eine Aufgabenstellung haben, die nicht über die bereitgestellten Knoten gelöst werden kann, sollten Sie zunächst einmal in der Standardbibliothek forschen, ob Sie dort eine passende oder ähnliche Lösung finden. Wenn Sie eine ähnliche Lösung finden, kopieren Sie einfach die vorhandene Standardprozedur und passen sie Sie gemäß Ihrer Bedürfnisse an. Informationen zum Arbeiten mit Skripten erhalten Sie im Handbuch, Kapitel 12 "Skripting".

Die Bibliothek ist in der Datei `qfs.qft` enthalten und ist Teil der QF-Test Distribution. Da sie mit jeder QF-Test Version weiterentwickelt wird, ist es nicht ratsam, Änderungen in der ausgelieferten Standardbibliothek vorzunehmen, sondern bei Bedarf eine Prozedur in eine eigene Testsuite zu kopieren und dort anzupassen.

Um die Bibliothek `qfs.qft` verwenden zu können, muss sie im "Testsuite" Wurzelknoten Ihrer Suite in den "Inkludierte Dateien" eingebunden werden. Bei neuen Testsuiten ist dies automatisch der Fall.

Aktion

- Wählen Sie den 'Testsuite'-Wurzelknoten Ihrer Testsuite aus.
- Überprüfen Sie in den Details des 'Testsuite'-Wurzelknoten, dass `qfs.qft` im Attribut "Inkludierte Dateien" aufgeführt ist.
- Fügen Sie `qfs.qft` zu dieser Liste dazu, falls es noch nicht enthalten ist.

Hinweis Eine Pfadangabe ist nicht notwendig, da das `include` Verzeichnis von QF-Test automatisch im Bibliothekspfad (siehe auch Referenzteil des Handbuchs) enthalten ist.

Im Folgenden beschreiben wir eine Auswahl der am häufigsten benötigten Standardprozeduren. Eine vollständige HTML-Dokumentation der Standardbibliothek finden Sie unter dem Menüpunkt Hilfe→Standardbibliothek qfs.qft...

7.1 Erforschen der Standardbibliothek

Zusätzlich zum Einfügen von Prozeduraufrufen aus der Standardbibliothek ist es hilfreich, einen Blick darauf zu werfen, wie Funktionen implementiert und organisiert sind.

Aktion

- Öffnen Sie die Bibliothek selbst, also die Suite `qfs.qft`, die sich im Verzeichnis `qftest-9.0.0/include` Ihrer QF-Test Installation befindet.

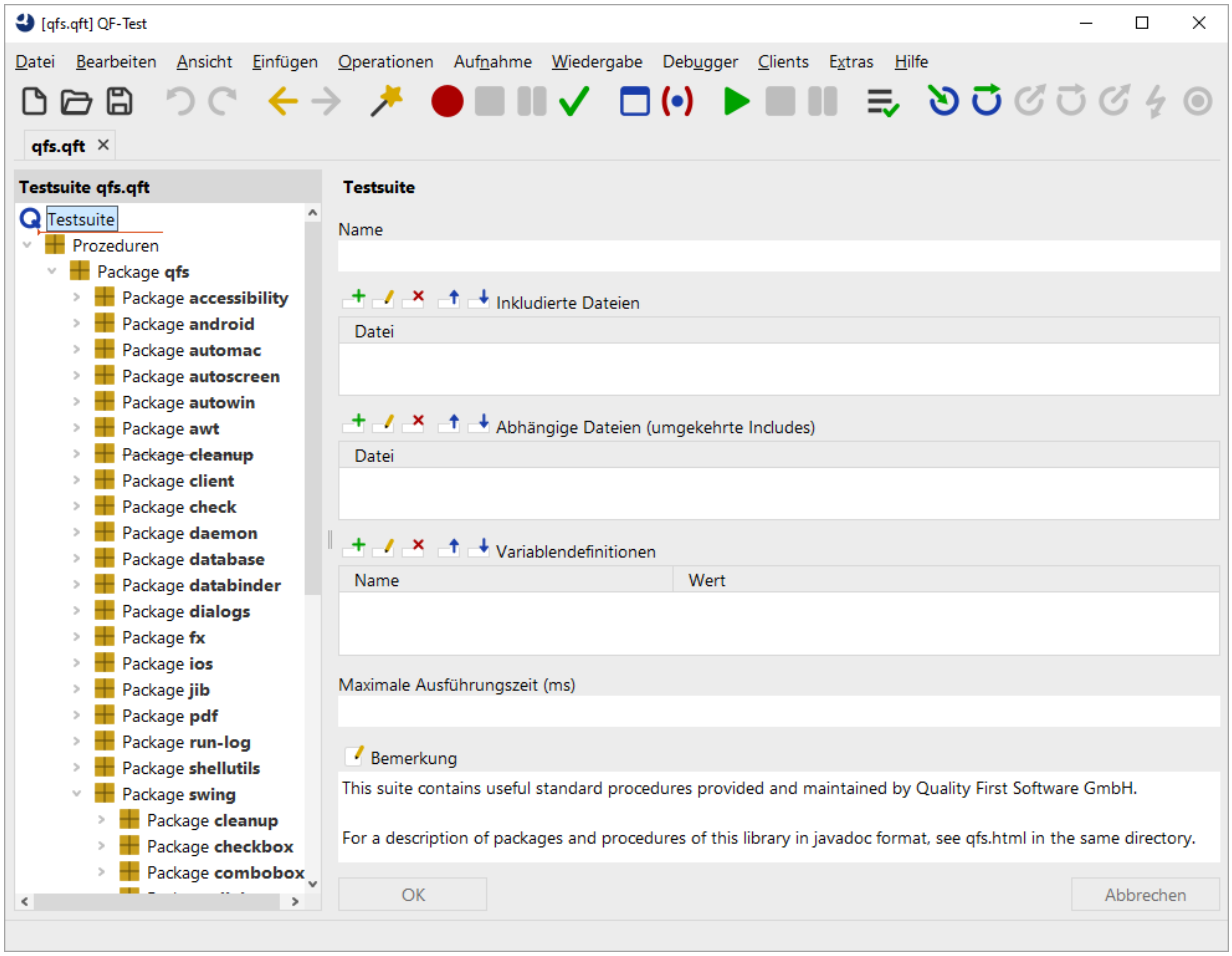

Abbildung 7.1: Die Standardbibliothek

Sie sehen ein Haupt-Package `qfs`, das die spezifischen Packages umschließt. Das `qfs` Package hilft dabei, die Prozeduren leicht als solche der Standardbibliothek zu identifizieren.

In fast allen Prozeduren unserer Bibliothek werden Sie die Verwendung der Variable `$(client)` bemerken. Dies ist ein Standardmechanismus, um Testsuiten unabhängig von einem spezifischen SUT zu gestalten. Für die Benutzung der Standardbibliothek wird vorausgesetzt, dass ein gültiger Wert für `$(client)` gesetzt wird, bevor eine ihrer Prozeduren verwendet werden kann.

7.2 Ausgewählte Packages und Prozeduren

Wir werfen nun einen genaueren Blick auf ein paar ausgewählte Packages und Prozeduren der Standardbibliothek.

Wir werden mit Packages beginnen, die den Zugriff auf Komponenten abhängig von der verwendeten GUI Technologie ermöglichen, also JavaFX, Swing, Eclipse/SWT und Web Komponenten.

7.2.1 Das Checkbox Package

Wir beginnen nun mit der genaueren Betrachtung der Packages `qfs.fx.checkbox` `qfs.swing.checkbox`, `qfs.swt.checkbox` oder `qfs.web.checkbox`.

Hier sind einige Prozeduren aus diesen Packages:

- **select** Selektiert ein Kontrollkästchen. Wenn sich das Kontrollkästchen bereits im ausgewählten Zustand befindet, wird keine Aktion ausgeführt.
- **deselect** Deselektiert ein Kontrollkästchen. Wenn sich das Kontrollkästchen bereits im nicht-ausgewählten Zustand befindet, wird keine Aktion ausgeführt.
- **set** Setzt ein Kontrollkästchen auf den angegebenen Zustand (true oder false).

Für jede dieser Prozeduren wird die QF-Test ID der Kontrollkästchenkomponente als variables Argument übergeben. Die Bibliothek kümmert sich um die Überprüfung, dass der Zustand des Kontrollkästchens wie erwartet gesetzt wurde.

Die anderen Prozeduren in diesem Package folgen demselben Muster.

7.2.2 Das Combobox bzw. Combo Package

Die Packages `qfs.fx.combobox`, `qfs.swing.combobox`, `qfs.swt.combo` und `qfs.web.select` enthalten Prozeduren, um Werte in einer Combobox zu selektieren.

Die wichtigsten Prozeduren sind:

- **setValue** Selektiert einen Wert in der Liste der Combobox.
- **getItemCount** Liefert die Anzahl der Einträge zurück.

7.2.3 Das General Package

Die Packages `qfs.fx.general`, `qfs.swing.general`, `qfs.swt.general` und `qfs.web.general` enthalten allgemeine Prozeduren für GUI-Elemente.

Die wichtigsten Prozeduren sind:

- **setLocation** Setzt die Position der Komponente mittels angegebenen Koordinaten.
- **setSize** Setzt die Größe der Komponente.

7.2.4 Das List Package

Die Packages `qfs.fx.list`, `qfs.swing.list`, `qfs.swt.list` und `qfs.web.list` enthalten allgemeine Prozeduren für Listen.

Die wichtigsten Prozeduren sind:

- **getItemCount** Zählt die Einträge einer Liste.

7.2.5 Das Menu Package

Die Packages `qfs.fx.menu`, `qfs.swing.menu` und `qfs.swt.menu` erlauben es Ihnen, in einfacher Weise Einträge und auch Kontrollkästchen in Menüs oder Untermenüs auszuwählen bzw. zu setzen. Die wichtigsten Prozeduren sind:

- **selectItem** Wählt einen Eintrag in einem Menü aus.
- **selectSubItem** Wählt einen Eintrag in einem Untermenü aus.

An alle diese Prozeduren muss die QF-Test ID der Menükomponente wie auch des Eintrags bzw. Kontrollkästchens übergeben werden. Die Benutzung variiert leicht, abhängig von der jeweiligen Art der Prozedur.

7.2.6 Das Table Package

Die Packages `qfs.fx.table`, `qfs.swing.table`, `qfs.swt.table` und `qfs.web.table` enthalten Hilfsprozeduren für Tabellen.

- **getRowCount** Liefert die aktuelle Zeilenanzahl einer Tabelle zurück. Diese Prozedur verwendet technologiespezifische Methoden um an die Anzahl zu kommen.
- **getColumnCount** Liefert die aktuelle Spaltenanzahl einer Tabelle zurück. Diese Prozedur verwendet technologiespezifische Methoden um an die Anzahl zu kommen.
- **selectCell** Selektiert eine angegebene Tabellenzelle.

7.2.7 Das Tree Package

Die Bibliothek stellt in den Packages `qfs.fx.tree`, `qfs.swing.tree`, `qfs.swt.tree` und `qfs.web.tree` einige einfache Prozeduren zur Manipulation von Bäumen zur Verfügung. Das sind:

- **collapseNode** Klappt einen Baumknoten ein. Ist der Knoten bereits eingeklappt, wird keine Aktion ausgeführt. Diese Prozedur besitzt drei einzelne Parameter für Baum und Knoten
- **expandNode** Klappt einen Baumknoten aus. Ist der Knoten bereits expandiert, wird keine weitere Aktion ausgeführt. Diese Prozedur besitzt drei einzelne Parameter für Baum und Knoten.
- **selectNode** Selektiert einen angegebenen Baumknoten.

Jede der Prozeduren benötigt die QF-Test ID des Baumknotens als Argument.

7.2.8 Das Cleanup Package

Die Packages `qfs.fx.cleanup`, `qfs.swing.cleanup` und `qfs.swt.cleanup` bieten eine gute Unterstützung für das Aufräumen der SUT Umgebung, wenn unerwartet eine Exception auftritt. Stellen Sie sich zum Beispiel vor, dass eine Exception geworfen

wird, während auf ein Menü des SUTs zugegriffen wird. Die Exception bewirkt, dass der Ausführungspfad innerhalb Ihrer Testsuite zu einem Exception Handler umgeleitet wird - oder zu einem "impliziten" Exception Handler. Das bedeutet, dass der normale Ausführungspfad, der das geöffnete Menü in der Regel wieder ordnungsgemäß geschlossen hätte, unterbrochen wurde. Ohne eine entsprechende Aktion kann dieses Menü geöffnet bleiben und somit andere Ereignisse an das SUT blockieren.

Hier sehen Sie die wichtigsten Prozeduren innerhalb des Packages:

- **closeAllModalDialogs** bewirkt, dass modale Dialoge des SUTs geschlossen werden. **Nur für Swing und FX verfügbar**!
- **closeAllDialogsAndModalShells** bewirkt, dass alle Dialoge und modalen Shells geschlossen werden. **Nur für Eclipse/SWT verfügbar**!
- **closeAllMenus** Schließt alle offenen Menüs des SUT.

Das Konzept zur Behandlung von impliziten Exceptions ist von großer Bedeutung, denn eine Exception in einem einzigen Testfall soll nicht zum Beenden des gesamten Testlaufs führen. Lediglich der aktuelle Testfall soll abgebrochen werden, dann sollte es mit dem nächsten Testfall weitergehen.

Aus diesem Grund wird eine Exception innerhalb eines Testfalls auf dieser Ebene gefangen und nicht nach oben propagiert. Dies verhindert den Abbruch des gesamten Testlaufs. Der Fehlerstatus wird jedoch stets korrekt im Protokoll und Report festgehalten.

Benutzt der Testfall eine Abhängigkeit, wird die Exception an den Catch Knoten derselben übergeben, falls ein solcher vorhanden ist. Diese Art der Behandlung von Exceptions (und Fehlern) wird im Kapitel Abhängigkeiten des Handbuchs erklärt.

7.2.9 Das Run-log Package

Das Package `qfs.run-log` enthält Prozeduren, um Meldungen in das Protokoll zu schreiben.

Hier sehen Sie die Liste von verfügbaren Prozeduren innerhalb des Packages:

- **logError** Schreibt eine Fehlermeldung ins Protokoll.
- **logWarning** Schreibt eine Warnung ins Protokoll.
- **logMessage** Schreibt eine Meldung ins Protokoll.

7.2.10 Das Run-log.Screenshots Package

Das `qfs.run-log.screenshots` Package enthält Prozeduren, die Bildschirmabbilder ins Protokoll schreiben und einige Hilfsprozeduren.

Hier sehen Sie die Liste von verfügbaren Prozeduren innerhalb des Packages:

- **getMonitorCount** Liefert die Anzahl der an den Computer angeschlossenen Monitore.
- **logScreenshot** Schreibt ein Bildschirmabbild des aktuellen Monitors ins Protokoll.
- **logImageOfComponent** Schreibt ein Bildschirmabbild einer Komponente ins Protokoll.
- **logScreenshotOfMonitor** Schreibt ein Bildschirmabbild eines angegebenen Monitors ins Protokoll.

7.2.11 Das Shellutils Package

Das `qfs.shellutils` Package beinhaltet Prozeduren für die wichtigsten Shell-Kommandos.

Hier sehen Sie die Liste von verfügbaren Prozeduren innerhalb des Packages:

- **copy** Kopiert eine angegebene Datei oder ein Verzeichnis an eine neue Stelle.
- **deleteFile** Löscht eine angegebene Datei.
- **exists** Prüft, ob eine angegebene Datei oder ein Verzeichnis existiert.
- **getBasename** Gibt den Dateinamen einer Datei zurück.
- **getParentDirectory** Gibt die Verzeichnisstruktur einer Datei zurück.
- **mkdir** Erzeugt ein Verzeichnis. Noch nicht existierende Verzeichnisse werden angelegt.
- **move** Verschiebt eine angegebene Datei oder ein Verzeichnis.
- **touch** Erzeugt eine Datei.
- **removeDirectory** Löscht ein angegebenes Verzeichnis.

7.2.12 Das Utils Package

Das Package `qfs.utils` enthält nützliche Prozeduren für häufig auftretende Anforderungen der Testentwicklung.

Hier sehen Sie einige Prozeduren des Packages:

- **getDate** Gibt einen String zurück, der ein Datum enthält. Standardmäßig wird das aktuelle Datum zurückgegeben. (Andere Daten sind konfigurierbar.)
- **getTime** Gibt einen String zurück, der eine Zeit enthält. Standardmäßig wird die aktuelle Zeit zurückgegeben. (Andere Zeiten sind konfigurierbar.)
- **logMemory** Schreibt den aktuellen Speicherverbrauch ins Protokoll.
- **printVariable** Gibt den Inhalt einer spezifizierten Variable auf der Konsole aus.
- **printMessage** Gibt den Inhalt einer angegebenen Nachricht auf der Konsole aus.
- **writeMessageIntoFile** Schreibt einen angegebenen String in eine angegebene Datei.

7.2.13 Das Database Package

Das Package `qfs.database` enthält nützliche Prozeduren, um mit Datenbanken zu interagieren.

Bitte beachten Sie, dass die jar-Datei mit dem Datenbanktreiber vor dem Start von QF-Test ins `qftest` Pluginverzeichnis kopiert werden muss.

Für weitere Informationen über den Aufbau einer Datenbankverbindung kontaktieren Sie bitte einen Entwickler oder werfen Sie einen Blick auf www.connectionstrings.com.

Die wichtigsten Prozeduren sind:

- **executeSelectStatement** Führt einen angegebenen SQL-Select-Befehl aus. Das Ergebnis wird zum einen in die globale Variable ”resultRows” des Jython Variablenstacks geschrieben und ist somit in jedem Jython Skript verfügbar. Zum anderen wird das Ergebnis auch in eine Gruppenvariable mit dem Standardnamen ”resultGroup” geschrieben und ist somit direkt von QF-Test Knoten aus ansprechbar.
- **executeStatement** Führt einen angegebenen SQL Befehl aus. Hier kann jedes beliebige SQL Kommando ausgeführt werden.

7.2.14 Das Check Package

Das `qfs.check` Package enthält Prozeduren, die Checks ausführen.

Die wichtigsten Prozeduren sind:

- **checkEnabledStatus** Überprüft, ob eine Komponente en- bzw. disabled ist. Im Fehlerfall wird von der Prozedur ein entsprechender Fehler ins Protokoll geloggt.
- **checkSelectedStatus** Überprüft, ob eine Komponente selektiert bzw. nicht selektiert ist. Im Fehlerfall wird von der Prozedur ein entsprechender Fehler ins Protokoll geloggt.
- **checkText** Überprüft den Text einer Komponente. Im Fehlerfall wird von der Prozedur ein entsprechender Fehler ins Protokoll geloggt.

7.2.15 Das Databinder Package

Das Package `qfs.databinder` enthält Prozeduren zur Ausführung innerhalb eines Datentreiber Knotens, um Daten für datengetriebenes Testen zu binden.

Die wichtigsten Prozeduren sind:

- **bindList** Bindet eine Liste von Werten an eine Variable. Die Werte sind durch Leerzeichen oder das als Parameter übergebene Trennzeichen getrennt.
- **bindSets** Bindet Sätze von Werten an einen Satz von Variablen. Die Sätze von Werten sind durch Zeilenumbrüche getrennt, die Werte innerhalb eines Satzes durch Leerzeichen oder das als Parameter übergebene Trennzeichen.

Kapitel 8

Ablaufsteuerung (Java)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Ablaufsteuerung'
https://www.qftest.com/de/yt/tutorial-8.html

Die zwei wichtigsten Kontrollstrukturen von QF-Test sind Schleifen und die bedingte Ausführung von Knoten. Schleifen können über zwei verschieden Knoten implementiert werden: While und Schleife Knoten. If, Elseif und Else Knoten stehen für die bedingte Ausführung von Knoten zur Verfügung.

8.1 If - else

If und Else Knoten kennen Sie bereits aus der Vorbereitung Sequenz im Kapitel Starten der Anwendung[5]. Sehen wir uns diese nun etwas genauer an.

Abbildung 8.1: Setup Sequenz mit If/Elseif Knoten

Über einen If Knoten können Sie steuern, ob bestimmte Knoten ausgeführt werden oder nicht. In unserem Fall geht es um den Start des SUT. Zunächst müssen wir herausfinden, ob die Applikation bereits läuft. Dies geschieht über den Warten auf Client Knoten, der als Ergebnis entweder `true` (wahr) oder `false` (falsch) in die Variable `isSUTRunning` schreibt.

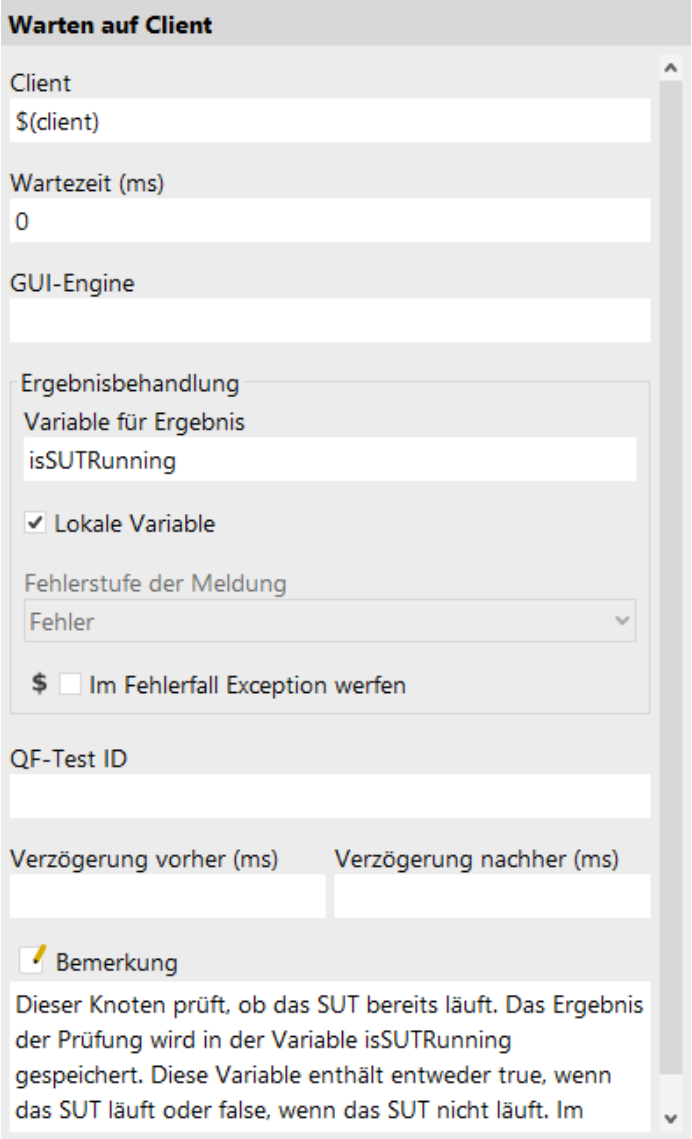

Abbildung 8.2: Warten auf Client setzt die Variable "isSUTRunning" mit dem Ergebnis

Der If Knoten wertet die Ergebnisvariable `isSUTRunning` im 'Bedingung' Attribut aus. Da auf den Wert der Variablen zugegriffen werden soll, wird die Syntax `$()` verwendet (vgl. Variablensyntax in Kapitel Abschnitt 6.1[66]).

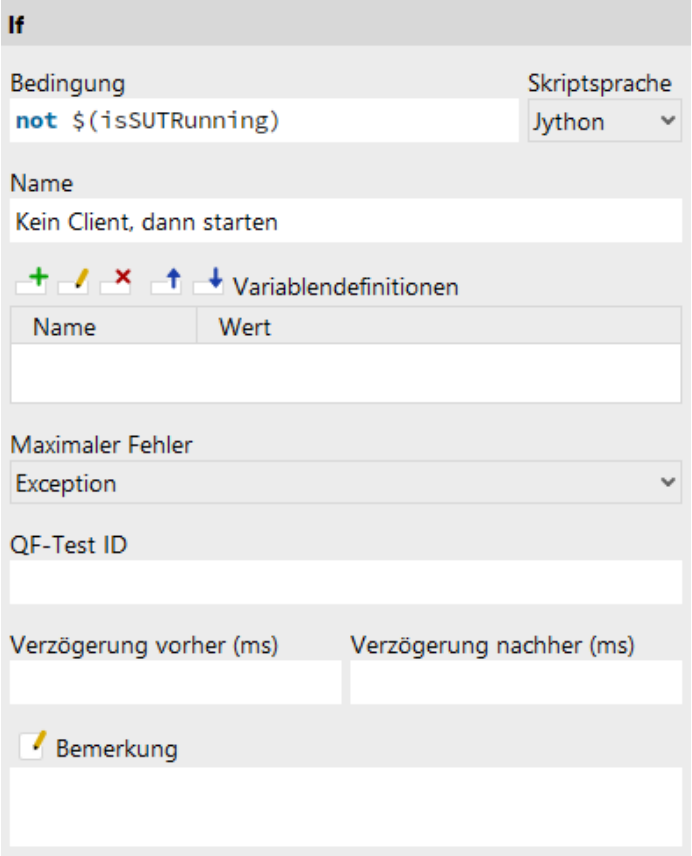

Abbildung 8.3: Der If Knoten wertet die Variable aus

Je nachdem, ob die Applikation bereits läuft, führt QF-Test die im If Knoten enthaltenen Knoten aus oder nicht.

Aktion

- **Beenden Sie das CarConfig Demo**, falls es läuft.
- **Führen Sie den Vorbereitung Knoten mit Einzelschritten aus.**
- **Führen Sie den Vorbereitung Knoten nochmals mit Einzelschritten aus** während das CarConfig Demo läuft.

In der Variablendefinition-Tabelle können Sie den Wert der Variablen `isSUTRunning` prüfen. Beim ersten Mal ist der Wert `false` und damit die Bedingung `not $(isSUTRunning)` wahr, d.h. die Knoten für den SUT-Start werden ausgeführt. Beim zweiten Mal ist der Wert `true` und damit die Bedingung `false`. Die Knoten im If Knoten werden übersprungen.

Im ersten If Knoten befindet sich ein weiterer, der prüft, um welches Betriebssystem es sich handelt. Bei Windows werden die Knoten, die direkt unter dem If Knoten liegen,

ausgeführt. Alternativ, d.h. wenn das Betriebssystem nicht Windows ist, werden die Kindknoten im Else Knoten ausgeführt.

Bei der Prüfung des Betriebssystems wird direkt auf eine QF-Test Variable zugegriffen. QF-Test speichert die Betriebssysteminformation in einer Gruppenvariable ab, wobei die Gruppe `qftest` und die Variablen `linux`, `macos` oder `windows` heißen. Die Syntax für den Zugriff auf Gruppenvariablen ist `${group:varname}`, z.B. `${qftest:windows}`.

8.2 Schleifen

QF-Test stellt zwei Knotentypen für die Implementierung von Schleifen zur Verfügung:

- Schleife Knoten führen ihre Kindknoten so oft aus, wie angegeben ist. Man kann die Schleife jedoch über einen Break Knoten jederzeit verlassen.
- While Knoten führen ihre Kindknoten so oft aus, bis die angegebene Bedingung nicht mehr gegeben ist. Derartige Schleifen können ebenfalls über einen Break Knoten jederzeit verlassen werden.

Hinweis

Schleife Knoten enden auf jeden Fall nach der angegeben Anzahl von Wiederholungen. Bei While Knoten muss man jedoch selbst dafür sorgen, dass die Ausführung irgendwann endet, indem die Bedingung falsch wird. Ansonsten kommt es zur Endlosschleife. Im interaktiven Modus können Sie in so einem Fall einfach die Pausetaste ⏸ drücken. Im Batch-Modus, d.h. wenn Sie QF-Test mit dem Kommandozeilenparameter `-batch` starten um die angegebene Testsuite ohne die QF-Test Benutzeroberfläche auszuführen, müssen Sie dann jedoch den QF-Test Prozess "abschießen".

In der folgenden Übung wollen wir einen Testfall implementieren, der prüft, ob eine bestimmte Zeile in der Tabelle des CarConfig Demos angezeigt wird.

Die im Testfall durchgeführten Aktionen sind:

- Anzahl Tabellenzeilen bestimmen.
- Über alle Zeilen iterieren und prüfen, ob die Zeile passt.
- Wenn die Zeile gefunden wurde, die Schleife abbrechen.
- Falls die Zeile nicht gefunden wurde, einen Fehler ins Protokoll schreiben.

Bitte beginnen Sie mit der Aufnahme eines Checks auf die zu suchende Zeile:

Aktion

- **Aktivieren Sie den Check-Aufnahmemodus** über "Checks aufnehmen" ✓ .

- **Führen Sie einen Rechtsklick auf eine Tabellenzeile** im CarConfig Demo aus und wählen Sie den Menüpunkt Zeile aus dem Popup-Menü.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .
- **Ändern Sie den Namen der aufgenommenen Sequenz** z.B. in `Zeile prüfen`.
- **Wandeln Sie die Sequenz in einen Testfall um:** Rechtsklick auf den Sequenz Knoten und Auswahl des Untermenüpunkts Knoten konvertieren in→Testfall im Popup-Menü.

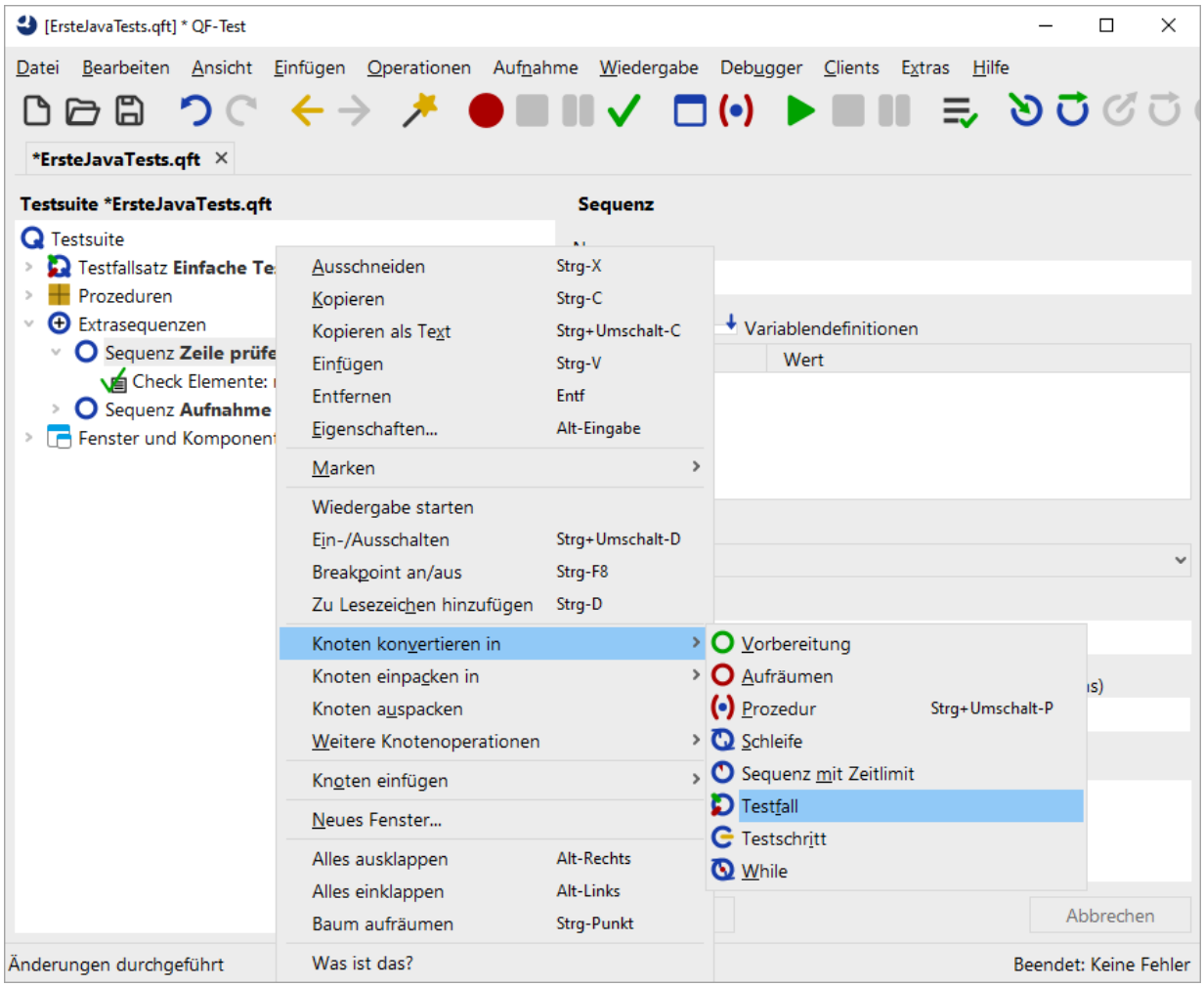

Abbildung 8.4: Knoten konvertieren

In QF-Test können Sie sehr effizient Knoten hinzufügen, indem Sie einen Knoten in einen anderen einpacken:

Aktion

- Öffnen Sie den Testfall Knoten und **packen Sie den aufgenommenen Check Knoten in eine Schleife** indem Sie rechts auf den Knoten klicken und in dem sich öffnenden Popupmenü den Punkt Knoten einpacken in→Schleife auswählen.

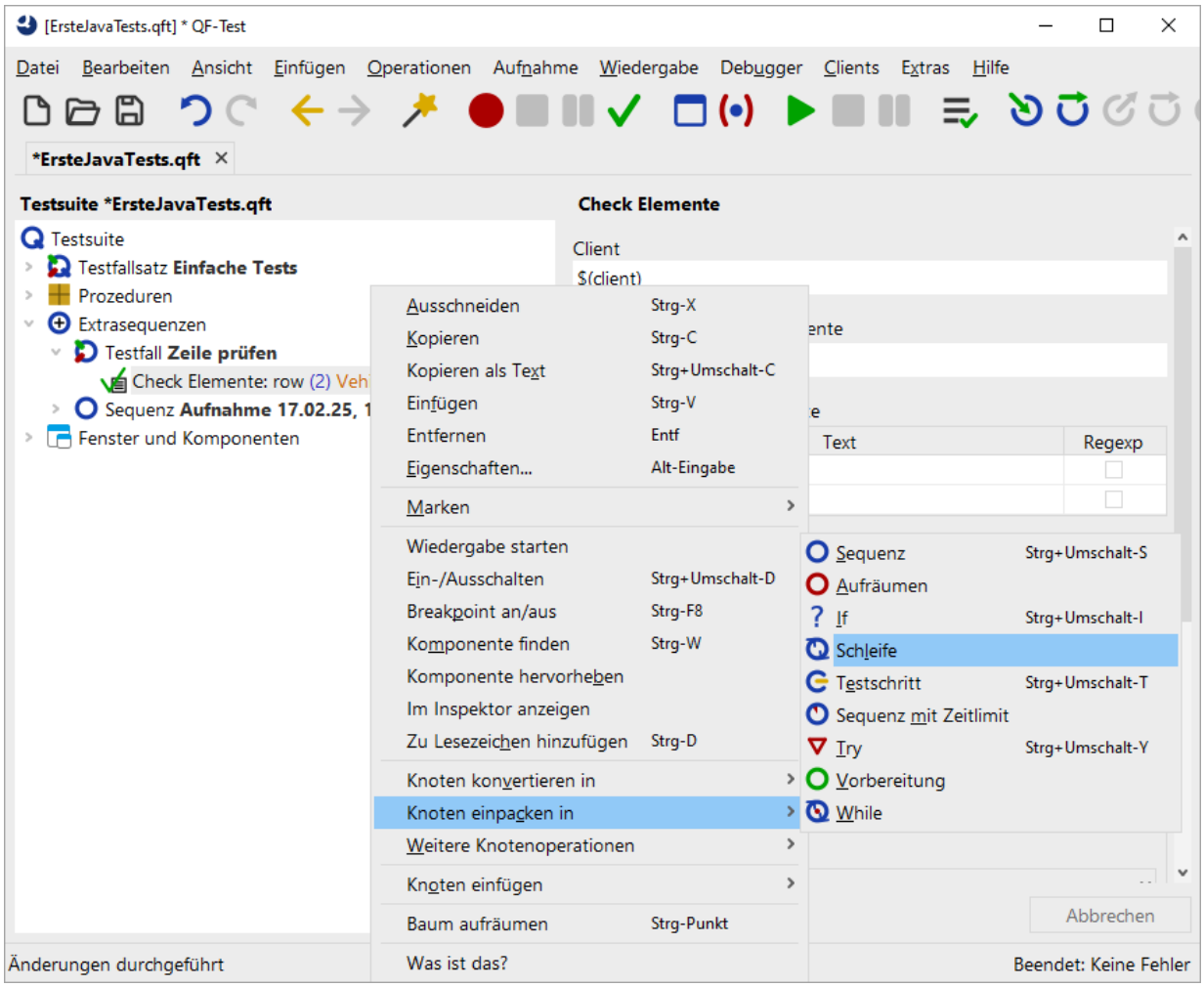

Abbildung 8.5: Knoten einpacken

QF-Test ermittelt dynamisch, in welche Knoten ein Knoten eingepackt werden kann und bietet nur diese zur Auswahl an. Entsprechend kann es passieren, dass Sie "Schleife" im Untermenü nicht finden. Sie sollten dann prüfen, ob Sie den Rechtsklick auf den richtigen Knoten ausgeführt haben. Dasselbe gilt für die Operationen "Knoten konvertieren in" und "Knoten einfügen".

Als nächstes setzen wir den Wert für das Attribut 'Anzahl Wiederholungen' des Schleife Knoten. Dazu müssen wir bestimmen, wie viele Zeilen die Tabelle hat. Es gibt keinen Knoten, der diese Operation direkt ausführen kann. Allerdings gibt es eine derartige

Prozedur in der im letzten Kapitel besprochenen Standardbibliothek. Diese befindet sich im Package `qfs.swing.table` und heißt `getRowCount`.

Aktion

- **Selektieren Sie den Testfall Knoten und drücken Strg-A**.
- **Klicken Sie die Schaltfläche "Prozedur auswählen"** links neben der Überschrift 'Name der Prozedur'.
- **Wählen Sie den Reiter 'qfs.qft'** im 'Prozedur auswählen' Dialog.
- **Navigieren Sie zu 'getRowCount' im Package 'qfs.swing.table'**
- **Drücken Sie 'OK'** um die Prozedur auszuwählen.
- **Drücken Sie 'OK'** um den 'Prozeduraufruf' Dialog zu schließen.

Das Hinzufügen einer Prozedur über Strg-A wurde bereits in Manuelle Erstellung von Prozeduren[35] behandelt. Dort finden Sie auch Screenshots zur Aktion.

Aktion

- **Fügen Sie eine Variable** mit dem Namen `Zeilen` **im Attribut 'Variable für Rückgabewert' ein**.
- **Ändern Sie den Standardwert für `id` in** der Variablendefinitionen-Tabelle auf die QF-Test Komponenten ID der Tabelle **`VehicleTable`**.
- Klicken Sie **OK**.
- **Wählen Sie den 'Schleife'-Knoten**.
- Im Attribut 'Anzahl Wiederholungen' des Schleife Knotens **tragen Sie eine Referenz auf die Variable `$(Zeilen)` ein**.
- **Tragen Sie den Namen der Zählervariable**, z.B. `i`, im entsprechenden Attribut des Schleife Knotens **ein**.
- Klicken Sie **OK**.

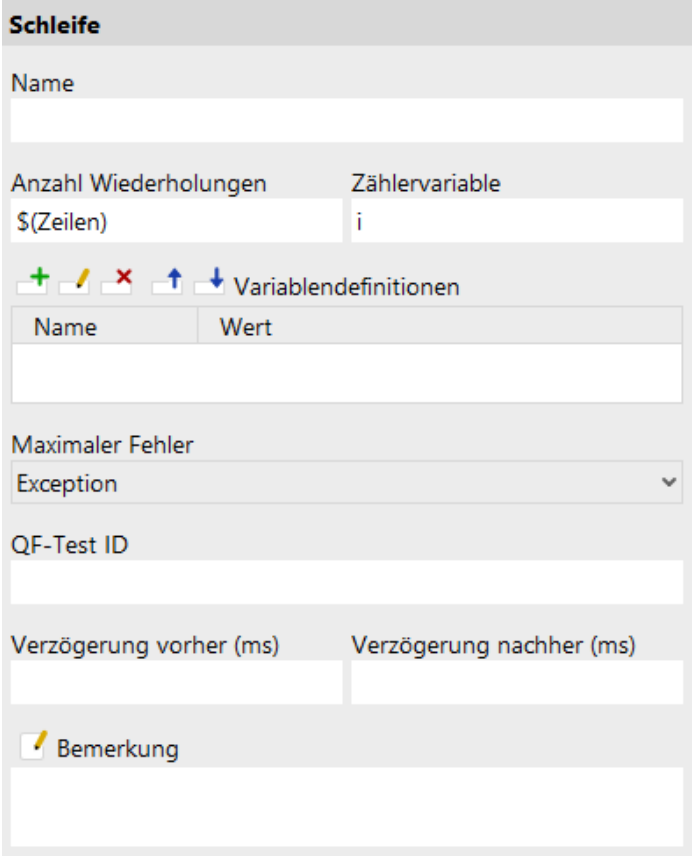

Abbildung 8.6: Details eines Schleife Knotens

In den Details des Check Knotens tragen wir nun in der QF-Test ID der Komponente statt des aufgenommenen Zeilenindex eine Referenz auf die Zählervariable ein und setzen eine Ergebnisvariable. Außerdem fügen wir einen If Knoten unter dem Check Knoten hinzu, der das Ergebnis auswertet und die Schleife über einen Break Knoten verlässt, wenn die entsprechende Zeile gefunden wurde.

Aktion

- **Öffnen Sie den Schleife Knoten**.
- **Selektieren Sie den Check Knoten**.
- **Ändern Sie den aufgenommenen Zeilenindex** der QF-Test ID der Komponente in Zählervariable `$(i)`. Die QF-Test ID der Komponente sollte nun `VehicleTable@Modell&$(i)` lauten.
- **Tragen Sie den Variablennamen `ZeileGefunden` in das Attribut 'Variable für Ergebnis' ein** und klicken **OK**.

- Führen Sie einen **Rechtsklick auf den Check Knoten** aus und wählen Sie aus dem Popup-Menü den Unterpunkt Knoten einfügen→Ablaufsteuerung→Break aus.
- **Drücken Sie 'OK'** im 'Break' Dialog.
- **Packen Sie den Break Knoten in einen If Knoten** mittels des Tastaturkürzels Strg-Umschalt-I (Sie können natürlich auch über das Menü gehen).
- In den Details des 'If'-Knotens **tragen Sie `$(ZeileGefunden)` im Attribut 'Bedingung' ein** und klicken **OK**.

Die Variable `ZeileGefunden` wird vom Check Knoten entweder auf den Wert 'true' oder auf den Wert 'false' gesetzt, so dass wir im Attribut 'Bedingung' des If Knoten nur die Referenz auf die Variable `$(ZeileGefunden)` einzutragen brauchen.

In den nächsten Schritten wollen wir einen Else Knoten als letzten Knoten im Schleife Knoten einfügen. Er wird nur ausgeführt, wenn die Schleife so oft wie angegeben ausgeführt wurde, was in unserem Fall bedeutet, dass die Variable `ZeileGefunden` nie wahr wurde, weil die Zeile nicht gefunden wurde.

Aktion

- **Schließen Sie den If Knoten**, falls dies nicht bereits der Fall ist. Dies ist wichtig, da sonst der Else Knoten zum If Knoten und nicht zum Schleife Knoten gehören würde.
- Führen Sie einen Rechtsklick auf den If Knoten aus und wählen Sie auf dem Popup-Menü den Unterpunkt **Knoten einfügen→Ablaufsteuerung→Else**.
- **Klicken Sie im 'Else' Dialog 'OK'**.
- **Öffnen Sie den Else Knoten**.
- **Fügen Sie** aus der Standardbibliothek **die Prozedur `logError`** aus dem Package `qfs.run-log` wie oben beschrieben **ein**.
- In der 'Variablendefinitionen' Tabelle **tragen Sie `Zeile nicht gefunden`** als Wert der Zeile `message` **ein.**
- **Tragen Sie `true`** als Wert der Zeile `withScreenshots` **ein.**
- Drücken Sie **OK**.

Wenn Sie die Tests im Batch-Modus ausführen, sind Screenshots eine gute Unterstützung bei der Fehleranalyse. Da aber eine große Zahl Screenshots sehr große Protokolldateien erzeugen würden, ist der Standardwert für `withScreenshots false`.

Nun bleibt nur noch, den Testfall mit Vorbereitung und Aufräumen Knoten zu vervollständigen und ihn in den oberen Teil der Testsuite zu verschieben.

Aktion

- **Kopieren Sie die Vorbereitung und Aufräumen Knoten** aus 'Testset: Einfache Tests' in den neuen Testfall als ersten und letzten Knoten.
- **Verschieben Sie den Testfall** aus dem Bereich Extrasequenzen in den oberen Bereich der Testsuite hinter den Knoten 'Testset: Einfache Tests'.

Damit würde der neue Testfall wie folgt aussehen:

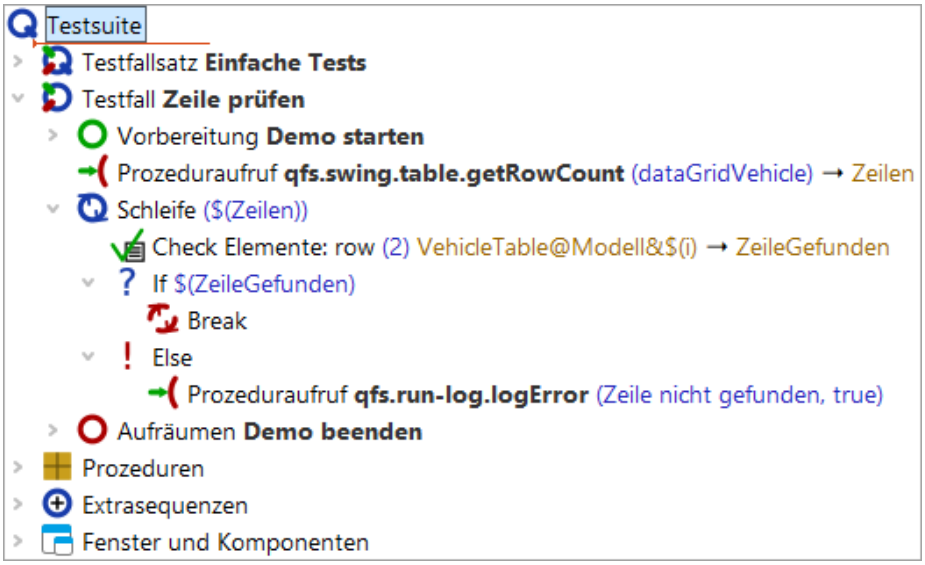

Abbildung 8.7: Der neue Testfall

Aktion

- **Führen Sie den neuen Testfall aus.**

Er sollte ohne Fehler laufen.

Aktion

- **Ändern Sie** nun in den Details des Check Elemente Knotens **den Namen des Fahrzeugs zum Beispiel auf `Falscher Wert`**.

Check Elemente

Client
$(client)

QF-Test ID der Komponente
VehicleTable@Modell&$(i)

Elemente

	Text	Regexp
0	Falscher Wert	☐
1	15.000,00 €	☐

Name des Check-Typs
row

Wartezeit (ms)

Ergebnisbehandlung
Variable für Ergebnis
ZeileGefunden

☐ Lokale Variable

Fehlerstufe der Meldung
Fehler

$ ☐ Im Fehlerfall Exception werfen

Name

QF-Test ID

Verzögerung vorher (ms) Verzögerung nachher (ms)

Bemerkung

Abbildung 8.8: Details eines Check Elemente Knoten

Aktion

- **Führen Sie den Testfall nochmals aus.**

Nun sollte der Test den Else Knoten der Schleife ausführen und eine Fehlermeldung anzeigen.

Kapitel 9

Nun ist es Zeit, Ihre eigene Anwendung zu starten (Java)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Nun ist es Zeit, Ihre eigene Anwendung zu starten'
https://www.qftest.com/de/yt/tutorial-9.html

Nachdem wir so viel Zeit mit all den Beispielanwendungen verbracht haben, sind Sie nun wirklich bereit, Ihre eigene Applikation zu starten (falls Sie dies nicht schon zwischendurch getan haben).

Der Schnellstart-Assistent, welcher über das Menü Extras→Schnellstart-Assistent... erreichbar ist, hilft Ihnen bei dieser Aufgabe. Folgen Sie einfach den Schritten innerhalb des Assistenten, um eine passende Startsequenz zu erzeugen. Bitte schauen Sie auch ins Handbuch Kapitel 3 "Schnellstart".

Es ist an der Zeit, das Gelernte in die Tat umzusetzen - kurze Sequenzen von Events und Checks aufzunehmen, Prozeduren zu erzeugen etc., um eine eigene Testbibliothek aufzubauen.

Damit endet der Basisteil in diesem Tutorial.

Teil II

Web GUIs testen mit QF-Test

Dieser zweite Teil des Tutorials soll Ihnen die Basiseigenschaften und -arbeitsabläufe von QF-Test erläutern. Er fokussiert sich auf das Testen von Web-Anwendungen und deren Besonderheiten.

Wenn Sie Java- oder native Windows-Anwendungen testen wollen, empfehlen wir Teil I(2) beziehungsweise Teil III(211). Alle Basisteile vermitteln die gleichen Schulungsinhalte, nutzen für die Beispiele jedoch eine jeweils passende Testanwendung.

Wenn Sie sich bereits durch den Teil I gearbeitet haben und zusätzlich zu Java auch Web-Anwendungen testen wollen, ist es i.d.R. nicht notwendig, dass Sie all die gleichen Szenarien noch einmal durchgehen. Jedoch sollte zumindest einen Blick in die Abschnitte Erzeugen der Startsequenz(123), Web-Komponentenerkennung(148) und Der Bereich Fenster und Komponenten(150) werfen, die spezifische Inhalte für Web haben.

Im Teil V(305) werden weiterführende Funktionalitäten von QF-Test erklärt, die für Tests sowohl von Java-, Web- und nativen Windows-Anwendungen genutzt werden können.

Kapitel 10

Bearbeiten einer Beispiel-Testsuite (Web)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Bearbeiten einer Beispiel-Testsuite'
https://www.qftest.com/de/yt/tutorial-1.html

In diesem ersten Kapitel werden wir uns die Struktur einer einfachen Testsuite anschauen, die wesentlichen Bestandteile erklären, sie ausführen und das Ergebnis auswerten.

10.1 Laden der Testsuite

Hinweis

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

Nach dem Starten von QF-Test laden Sie bitte unser erstes Beispiel:

Aktion

- Drücken Sie den Knopf , um den Dateiauswahl-Dialog zu öffnen.
- Wechseln Sie in das Unterverzeichnis `qftest-9.0.0/doc/tutorial` Ihrer QF-Test Installation.
- Dort wählen Sie bitte die Datei `ErsteWebTests.qft` aus und öffnen diese.

QF-Test präsentiert Ihnen die Testsuite wie im folgenden Bild dargestellt:

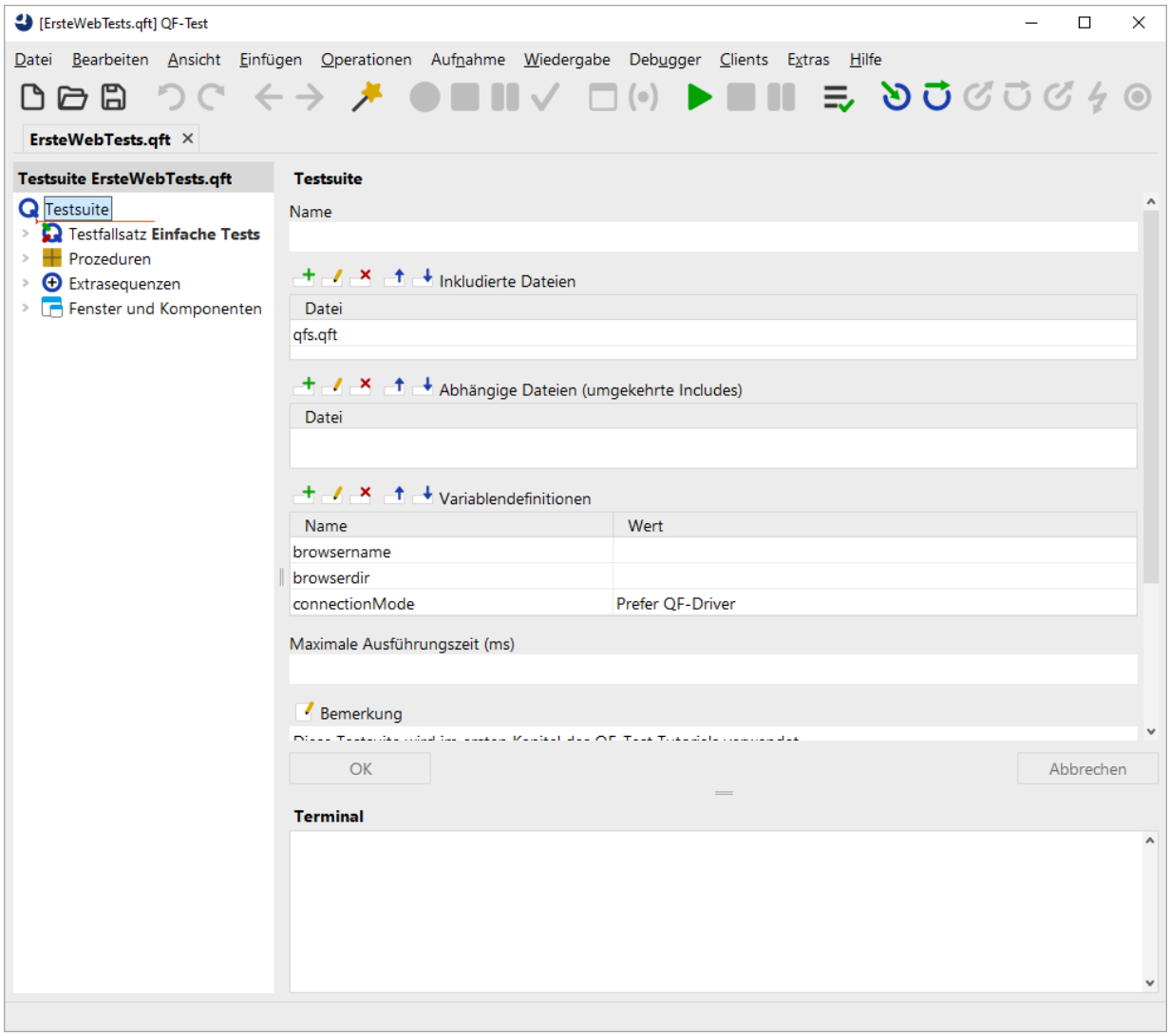

Abbildung 10.1: Die erste Testsuite

Der **linke Bereich** des Hauptfensters enthält die Testsuite, die in einer Baumstruktur dargestellt wird.
Rechts befindet sich die Detailansicht des Knotens, der im Baum gerade markiert ist. (Falls die Detailansicht bei Ihnen nicht zu sehen sein sollte, aktivieren Sie diese bitte über das Menü Ansicht→Details anzeigen.)
Im Bereich **unten rechts** befindet sich das Terminal, welches die Ausgaben von QF-Test und dem zu testenden Client protokolliert.

Mit Hilfe des Baumes können Sie durch die Testsuite navigieren und einzelne Knoten auswählen, für die dann jeweils die Details im rechten Fensterbereich eingeblendet werden.

Aktion

- **Doppelklicken** Sie bitte den Knoten **Testfallsatz: Einfache Tests** um ihn zu expandieren und die darin liegenden Knoten sehen zu können.

Der Testfallsatz enthält primär zwei Testfälle, umgeben von einem ”Vorbereitung”/”Aufräumen” Knotenpaar, das im Wesentlichen die Testanwendung startet bzw. beendet.

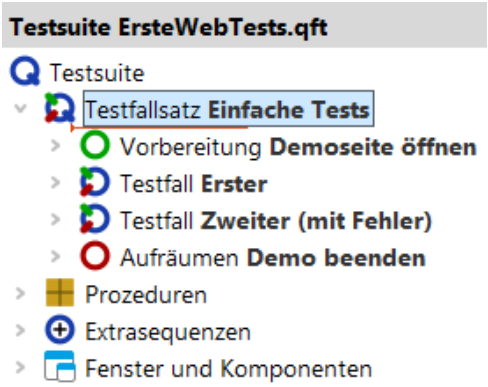

Abbildung 10.2: Der Inhalt des Testfallsatz Knotens

In den folgenden Abschnitten werden wir Funktion und Zweck der einzelnen Knoten erklären.

10.2 Starten des Browsers

Zuerst wollen wir die Vorbereitung genauer unter die Lupe nehmen:

Aktion

- **Expandieren** Sie den Knoten **Vorbereitung: Demoseite öffnen**, wie im folgenden Bild gezeigt.

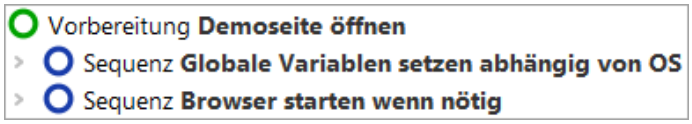

Abbildung 10.3: Der Knoten ”Vorbereitung”

Es werden zwei Kindknoten sichtbar:

1. **Globale Variablen setzen abhängig von OS** - definiert die globale Client Variable, die durchweg in der Testsuite benutzt wird, sowie den zu nutzenden Browser, abhängig vom Betriebssystem (Chrome für Windows und macOS, Firefox für Linux).

2. **Browser starten wenn nötig** - startet den entsprechenden Browser, wenn er nicht bereits läuft und lädt die Demoseite.

Lassen Sie uns noch einen kurzen Blick in die **Sequenz: Browser starten wenn nötig** werfen:

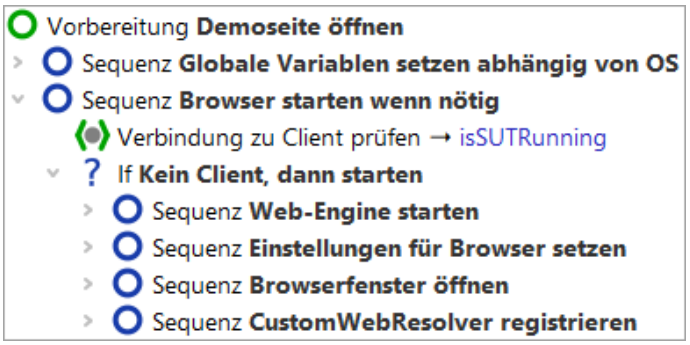

Abbildung 10.4: Die Sequenz zum Starten des Browsers

Zu Beginn steht ein **Warten auf Client** Knoten, der prüft, ob der Client bereits läuft. Nur wenn dies nicht der Fall ist, wird er gestartet.

Der Start des Browsers verläuft in vier Schritten:

1. **Web-Engine starten** - ist ein vorbereitendes Starten des Browser-Prozesses zur Konfiguration.

2. **Einstellungen für Browser setzen** - setzt die notwendigen Browser-Einstellungen (z.B. Cache, Cookies, Proxy, ...)

3. **Browserfenster öffnen** - öffnet das Browser-Fenster und wartet auf das Laden der Webseite.

4. **CustomWebResolver registrieren** - erlaubt die Konfiguration der Komponentenerkennung, so dass QF-Test die Funktionalität der Komponenten (Textfeld, Button, Datentabelle etc.) erkennt. Warum dies vorteilhaft ist sowie Informationen zur Konfiguration finden Sie im Abschnitt Web-Komponentenerkennung[148].

Diese vier Schritte werden automatisch generiert, wenn man den Schnellstart-Assistenten nutzt, der im nächsten Tutorialkapitel erklärt wird (Kapitel 11[123]).

Wir wollen nun die Anwendung wirklich starten:

Aktion

- **Markieren** Sie dazu bitte den Knoten **Vorbereitung: Demoseite öffnen**, doch belassen Sie ihn aufgeklappt.
- **Klicken Sie** den Knopf **Wiedergabe**. Dies führt den aktuellen ausgewählten Knoten aus.

Während der Ausführung wird der gerade aktive Knoten durch "**->**" markiert.

Nach Abschluss der Startsequenz sollte der Browser mit der "CarConfigurator" Demoseite am Bildschirm erscheinen. Da QF-Test nach Ende der Wiedergabe den Fokus zurückerhält, kann der Browser dadurch auch wieder verdeckt worden sein.

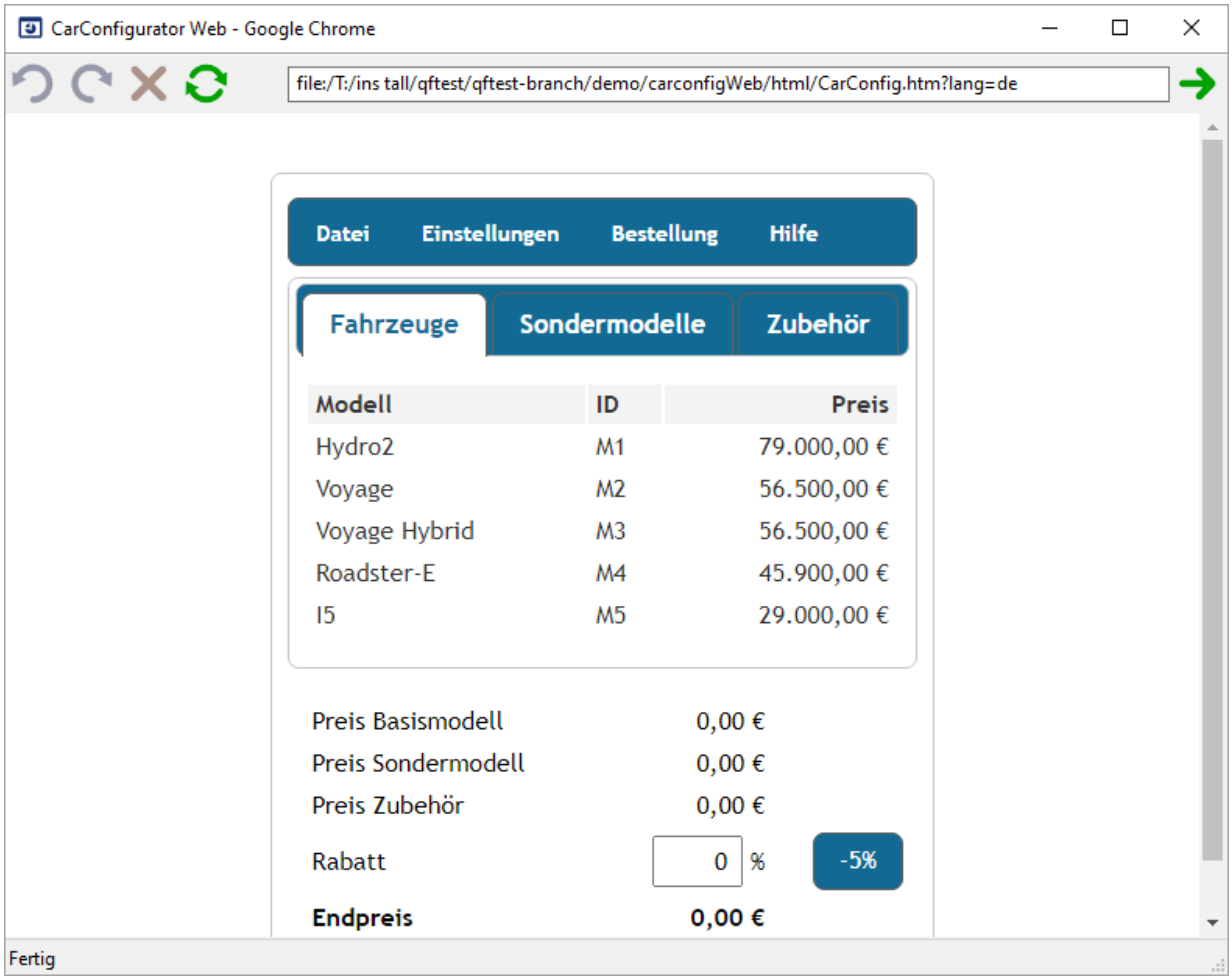

Abbildung 10.5: Das CarConfigurator Webdemo

10.3 Ein erster Testfall

Als nächstes wollen wir einen Blick auf den ersten Testfall werfen. Er besteht aus vier Testschritten:

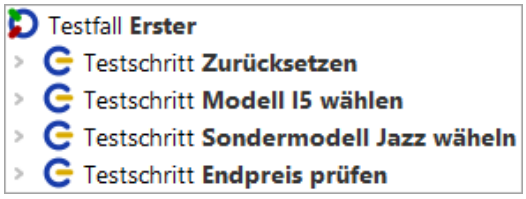

Abbildung 10.6: Der "Erste" Testfallknoten

1. **Zurücksetzen** - stellt den Anfangszustand der Anwendung über das Menü Datei->Zurücksetzen wieder her und selektiert den Tab Fahrzeuge.
2. **Modell I5 wählen** - Wählt das letzte Modell I5 in der Fahrzeugetabelle aus.
3. **Sondermodell Jazz wählen** - Wechselt zum Tab Sondermodelle und wählt dort Jazz.
4. **Endpreis prüfen** - Überprüft, dass der berechnete Wert dem Feldes Endpreis unten rechts einem vorgegebenen Wert entspricht.

Testschritte sind oft hilfreich, um einen Testfalls zu strukturieren und dadurch lesbar und verständlicher zu gestalten. Dies erleichtert später eine eventuelle Fehlersuche oder Anpassungen des Testfalls.

Aktion

- Bitte **expandieren** Sie die vier **Testschritt Knoten**.

Abbildung 10.7: Die Details des ersten Testfalls

Sie sehen diverse Mausklicks sowie einen Check. Zur besseren Lesbarkeit des Testfalls wurden sie mittels Testschrittknoten strukturiert. Neben der Aktionsart (Mausklick, Check) wird angezeigt, auf welche Anwendungskomponente sich die Aktion bezieht, also wohin z.B. der Mausklick geht. Diese Knoten können direkt über die Aufnahmefunktion von QF-Test erzeugt werden. Näheres hierzu erfahren Sie im nächsten Erstellen einer eigenen Testsuite (Web)(123).

Wir wollen uns nun die Ausführung des ersten Testfalls anschauen.

Aktion

- **Markieren** Sie dazu den **Testfall: Erster** Knoten.
- **Drücken Sie** anschließend den Wiedergabeknopf ▶ .

Die Testschritte werden nun der Reihe nach abgespielt, wobei dies typischerweise ziemlich rasch passiert.

Das aktuelle Testergebnis wird während und nach dem Testlauf in der Statuszeile am unteren rechten Rand des QF-Test Hauptfensters angezeigt und sollte ”Beendet: Keine Fehler” lauten. Daneben zeigt QF-Test verschiedene Zähler an. Der erste Zähler bezieht sich auf die Anzahl der ausgeführten Testfälle, der zweite auf die Zahl der ausgeführten Testfälle ohne Fehler. In unserem Fall wurde ein Testfall fehlerfrei ausgeführt, was einer Erfolgsquote von 100% entspricht.

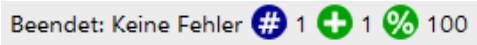

Abbildung 10.8: Die Ergebnisanzeige in der Statusleiste

Wenn Sie den Mauszeiger auf dem Symbol eines Testfallzählers ruhen lassen, wird Ihnen eine entsprechende Beschreibung angezeigt. Eine Auflistung aller Testfallzähler finden Sie im Kapitel Aufnahme und Wiedergabe des Handbuchs.

10.4 Ein zweiter Testfall - mit Fehler

Der zweite Testfall wird uns zeigen, was passiert, wenn ein Fehler bei der Testausführung auftritt.

Aktion

- Bitte **expandieren** Sie den Knoten **Testfall: Zweiter (mit Fehler)**.

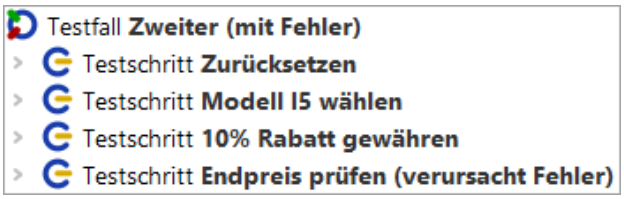

Abbildung 10.9: Der "Zweite" Testfallknoten

Bis auf den dritten Testschritt sieht es bekannt aus. Was tut der Unbekannte?

Testschritt: 10% Rabatt gewähren - Schreibt den Wert 10 in das Rabattfeld

Die Texteingabe ist eine weitere Basisaktion. Eingabe-Knoten kann man ebenfalls direkt über die Aufnahmefunktion generieren lassen. Den Wert 10 sieht man im Feld "Text" rechts und auch direkt im Text des Baumknotens.

Aktion

- **Expandieren** Sie den Knoten **Testschritt: 10% Rabatt gewähren**.

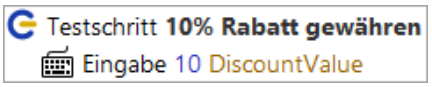

Abbildung 10.10: Die Details des zweiten Testfalls

Wir wollen uns die Ausführung des zweiten Testfalls anschauen.

Aktion

- **Markieren Sie** dazu den **Testfall: Zweiter (mit Fehler)** Knoten.
- **Drücken Sie** anschließend den Wiedergabeknopf ▶ .

Diesmal erscheint ein Dialog mit der Information, dass ein Fehler aufgetreten ist.

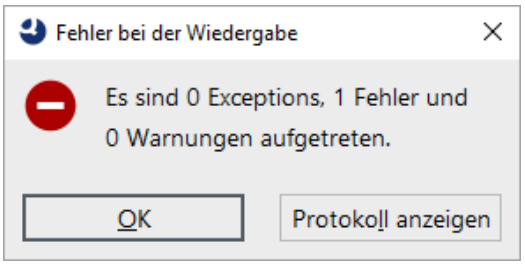

Abbildung 10.11: Fehler im zweiten Testfall

Was ist passiert? Fast immer wenn so ein Fall auftritt, ist es sinnvoll das Protokoll zu Rate zu ziehen.

Alternativ könnte man den Testfall zur Fehlersuche nochmal im Debug-Modus ausführen. Diese Vorgehensweise wird in Kapitel Benutzen des Debuggers (Web)(160) erläutert.

10.5 Das Protokoll zur Fehlerdiagnose

QF-Test protokolliert detaillierte Informationen für jede Testausführung.

Aktion

- **Öffnen** Sie nun bitte das **letzte Protokoll** über eine der folgenden Möglichkeiten:
  - den **Protokoll anzeigen** Knopf im Fehlerdialog

  oder falls Sie den Dialog bereits geschlossen haben

  - den **Button** in der Werkzeugleiste oder
  - über die Tastenkombination **Strg-L**.

Hinweis

Die Protokolle der letzten Testläufe können auch über die unteren Einträge im Menü 'Wiedergabe' aufgerufen werden.

Das Protokoll öffnet sich in einem separaten Fenster und zeigt die protokollierten Aktionen des zweiten Testfalls, den Sie soeben ausgeführt haben:

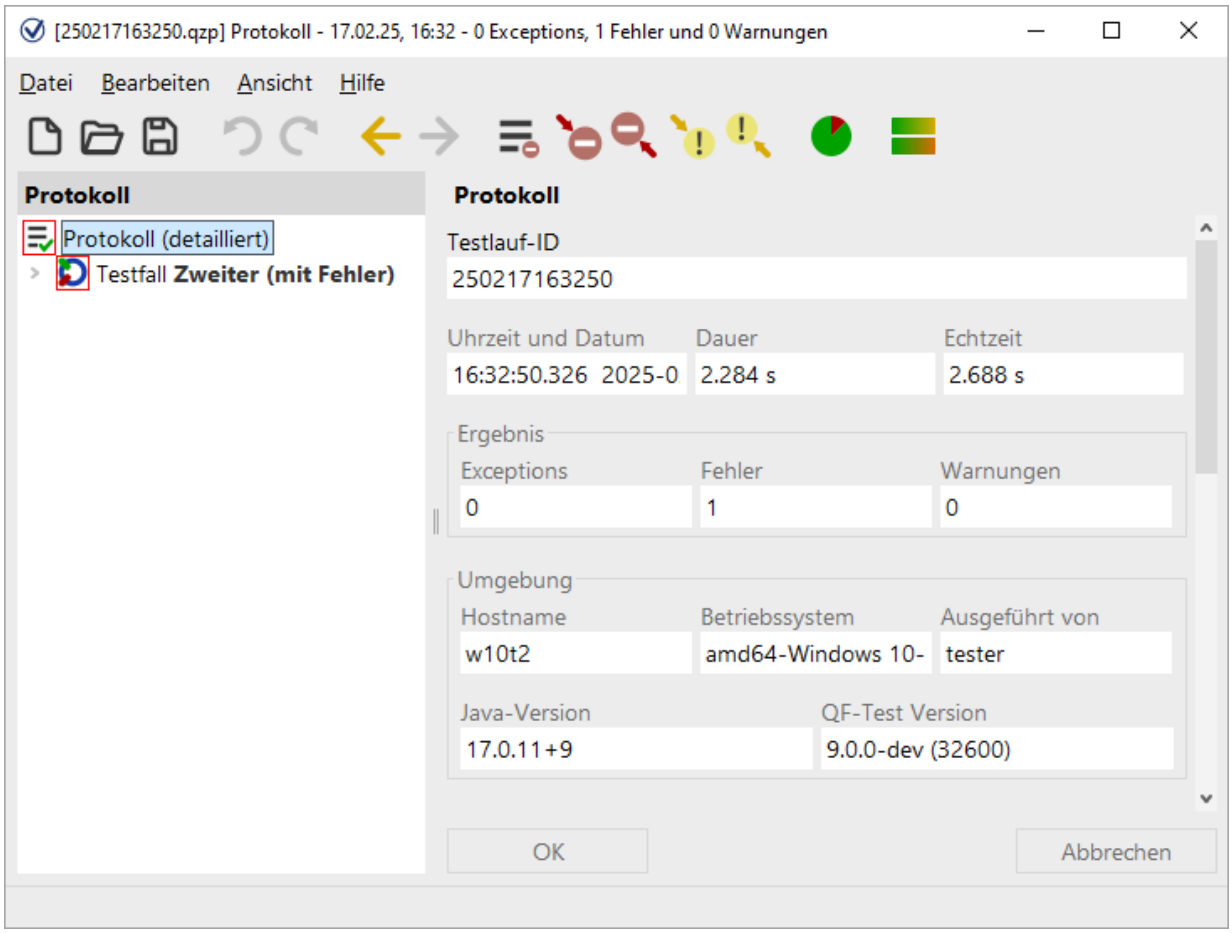

Abbildung 10.12: Protokoll des zweiten Testfalls

Das Protokoll ist in seinem Aufbau ähnlich zu dem der Testsuite. Der Baum links enthält wieder die bekannten Knoten, jedoch dieses Mal in der zeitlichen Abfolge des Testlaufs. Wenn man einen Knoten anwählt, sieht man rechts die Details inklusive Zeitstempel und Ausführungsdauer.

Im Baum links werden Ihnen **rote Rahmen** um einige Knoten auffallen. Diese zeigen an, dass sich darunter Fehler befinden. Wenn man den rot umrandeten Knoten Ebene für Ebene folgt, erreicht man irgendwann den Fehler.

Aktion

- Schneller und bequemer geht es über den Button **Nächsten Fehler finden** in der Werkzeugleiste oder auch die Tastenkombination **[Strg-N]**.

Alle rot markierten Knoten werden expandiert und der Knoten mit dem eigentlichen Fehler wird selektiert:

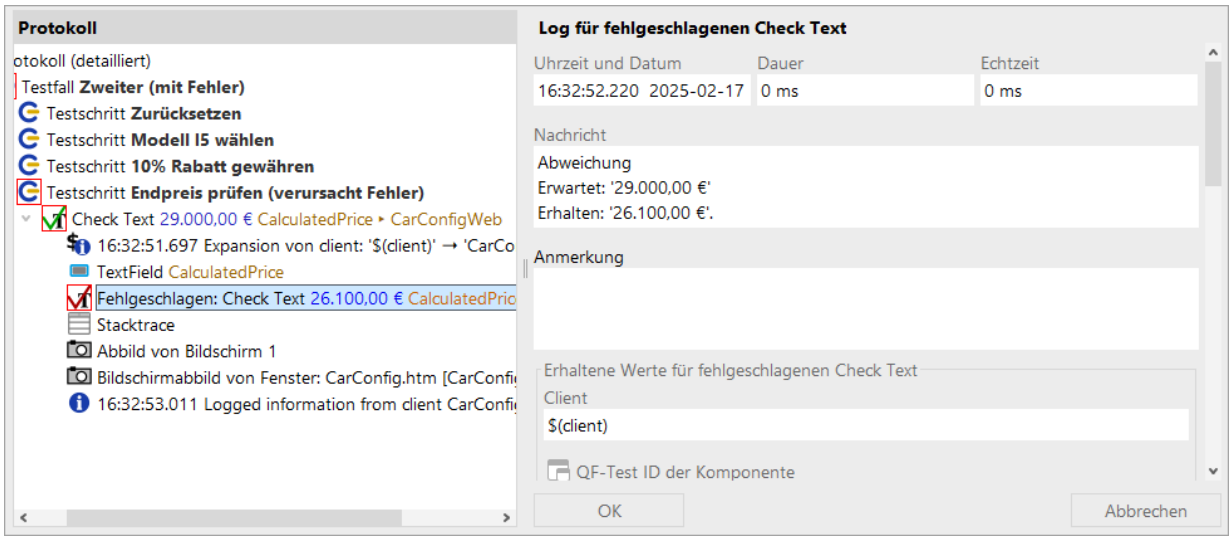

Abbildung 10.13: Fehlerdiagnose für den zweiten Testfall

Die Fehlermeldung auf der rechten Seite gibt an, dass der erhaltene Wert des Endpreis Feldes nicht dem erwarteten entspricht. Dieser Fehler wurde natürlich mit Absicht eingebaut, um zu zeigen, wie man bei der Analyse vorgehen kann.

Hilfreich bei der Fehleranalyse ist üblicherweise auch der übernächste Protokollknoten **Bildschirmabbild**. Seine Detailansicht enthält ein vollständiges Abbild des Bildschirms zum Zeitpunkt des Fehlers. Dies ist sehr nützlich, um den Zustand des SUTs zu sehen und daraus eventuell die Fehlerursache ableiten zu können. Die folgende Grafik zeigt den Knoten:

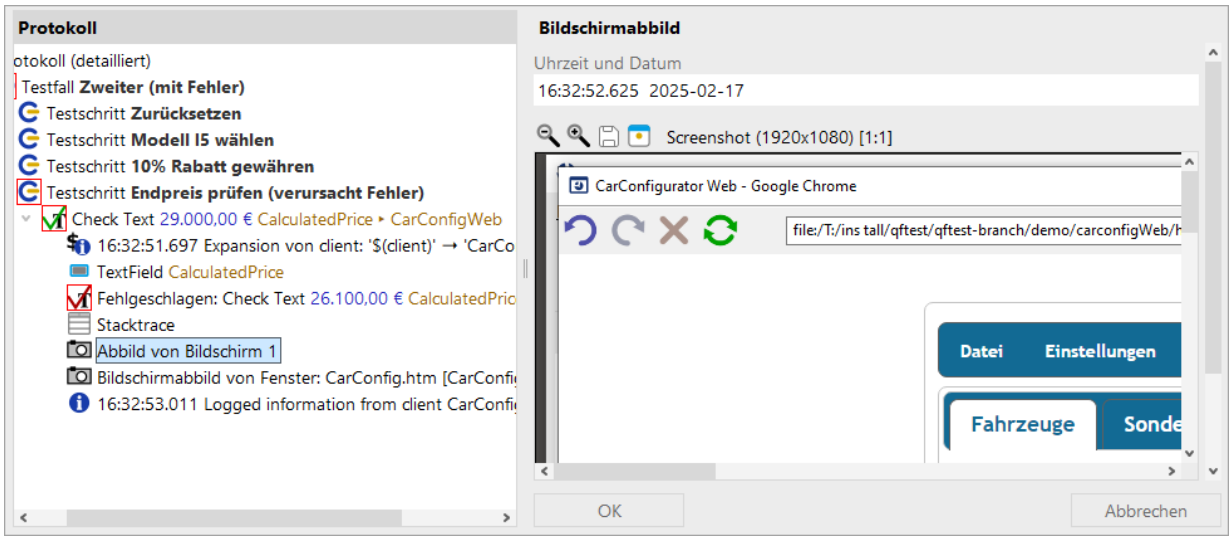

Abbildung 10.14: Knoten mit Bildschirmabbild der Fehlersituation

Neben dem Abbild aller Bildschirme speichert QF-Test auch Bilder der einzelnen Fenster des SUT zum Fehlerzeitpunkt. Dies erlaubt Ihnen deren Inhalt zu analysieren, auch wenn diese eigentlich durch andere Fenster oder Dialoge verdeckt sind.

Hinweis Die in einem längeren Testlauf im Protokoll gesammelten Informationen können große Mengen an Arbeitsspeicher verbrauchen. Deshalb ist QF-Test so voreingestellt, dass es kompakte Protokolle erstellt, wobei nur die für Fehlerdiagnose und Reportgenerierung wichtigen Informationen erhalten bleiben.
Diese Funktion ist mit der Option "Kompakte Protokolle erstellen" über Bearbeiten→Optionen...→Protokoll→Inhalt konfigurierbar. Der Typ eines Protokolls wird in seinem Wurzelknoten angezeigt. Auch die Anzahl der Bildschirmabbilder, die im Protokoll gespeichert werden, ist konfigurierbar.

10.6 Wo finde ich Hilfe?

In diesem Abschnitt machen wir eine kleine Pause, um einige allgemeine Hinweise zu geben.

Es gibt verschiedene Möglichkeiten, um Hilfe oder Antworten zu finden:

Die umfassendste Suche kann man über **Hilfe→Online Suche...** anstoßen. Dies bringt Sie auf die Suchfunktionalität unserer Homepage und erlaubt Ihnen die **Abfrage aller verfügbarer Dokumentation** (Handbuch https://www.qftest.com/qf-test-handbuch.html, Tutorial https://www.qftest.com/qf-test-tutorial.html, Standardbibliothek https://www.qftest.com/qf-test-support/dokumentation/standardbibliothek.html, Blog https://www.qftest.com/blog.html und unsere Videos https://www.qftest.com/los-gehts-mit-qf-test/videos.html). Die angezeigten Suchergebnisse können **passend gefiltert** werden.

Wenn Sie **offline** arbeiten und nach einem Thema suchen wollen, können Sie die **PDF Versionen von Handbuch und Tutorial** nutzen, die über das **Hilfe** Menü verfügbar sind. Offline HTML Versionen haben keine übergreifende Inhaltssuche. Jedoch gibt es auf jeder HTML Seite in Kopf- und Fußzeile einen Link auf die PDF Version, so dass der Wechsel dorthin einfach möglich ist.

QF-Test bietet eine **kontextsensitive Hilfe** für alle Baumknoten und deren Detailattribute an. Um diese zu nutzen, klicken Sie einfach mit der **rechten Maustaste** auf den gewünschten Knoten oder das Attribut in der Detailansicht. Im Kontextmenü wählen Sie dann den Eintrag **Was ist das?**. Dieser bringt Sie direkt zur passenden Referenzbeschreibung ins Handbuch.

Neben der Hilfestellung in der Dokumentation haben Sie auch die Möglichkeit unser Support-Team zu kontaktieren. Während Ihrer Evaluationsphase und anschließend als

Kunde mit einem gültigen Pflegevertrag können Sie Ihre Fragen direkt an unsere Supportexperten richten über das Support-Formular im QF-Test Hilfe-Menü Support-Team kontaktieren oder direkt über unsere Webseite.

10.7 Beenden der Anwendung

Wir haben noch nicht die Aufräumsequenz angeschaut und wollen dies nun tun:

Aktion

- **Expandieren** Sie den **Aufräumen: Demo beenden** Knoten.

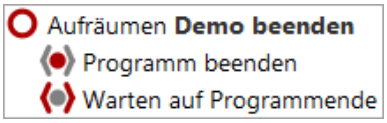

Abbildung 10.15: Die Aufräumsequenz

Unsere Aufräumsequenz stoppt "hart" den Client-Prozess und wartet anschließend, bis sich dieser vollständig beendet hat. Dies ist eine sehr einfache Variante aber für den Moment ausreichend.

Aktion

Führen Sie die **Aufräumsequenz aus** und lassen Sie damit den Browser mit dem Car-Configurator Demo verschwinden.

10.8 Ein vollständiger Testlauf

Nachdem wir uns Schritt für Schritt durch den Testfallsatz gearbeitet haben, wollen wir nun alles in einem Rutsch ausführen.

Aktion

- **Schließen Sie bitte das "CarConfigurator" Demo**, falls es noch läuft.
- **Markieren** Sie den **Testfallsatz "Einfache Tests"**.
- Führen Sie ihn aus mittels ▶ .

Der Testlauf endet mit dem bekannten Fehler.

Aktion

- Wenn Sie nun bitte mittels das **Protokoll öffnen**, sehen Sie, wie QF-Test den Test abgearbeitet hat.

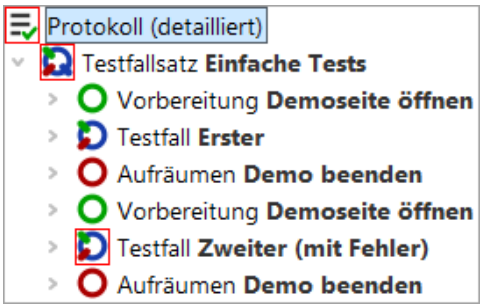

Abbildung 10.16: Das Protokoll des gesamten Testfallsatzes

Man sieht, dass die Vorbereitungs- und Aufräumenknoten vor bzw. nach **jedem Testfall** ausgeführt werden. Dies ist eine Eigenschaft, die diese im Zusammenspiel mit einem Testfallsatzknoten entwickeln. Dadurch wird für jeden Testfall immer ein sauberer Ausgangszustand hergestellt.

Hinweis Das SUT nach jedem Testfall zu beenden ist nicht die eleganteste Art, einen sauberen Ausgangszustand zu erreichen. Elegantere Wege zur Herstellung einer definierten Testausgangssituation und Durchführung der notwendigen Aufräumarbeiten werden in Kapitel (Kapitel 29[316]) dieses Tutorials erklärt.

10.9 Reportgenerierung

Im Qualitätssicherungsprozess ist es wichtig, Testergebnisse zu dokumentieren und auch zu archivieren. QF-Test bietet die Möglichkeit, aus Protokollen Testreports zu generieren. Wir wollen dies für das gerade aufgezeichnete Protokoll beispielhaft durchführen.

Aktion

- **Öffnen** Sie bitte das **Protokoll** und
- **wählen** im Menü **Datei→Report erstellen...**.

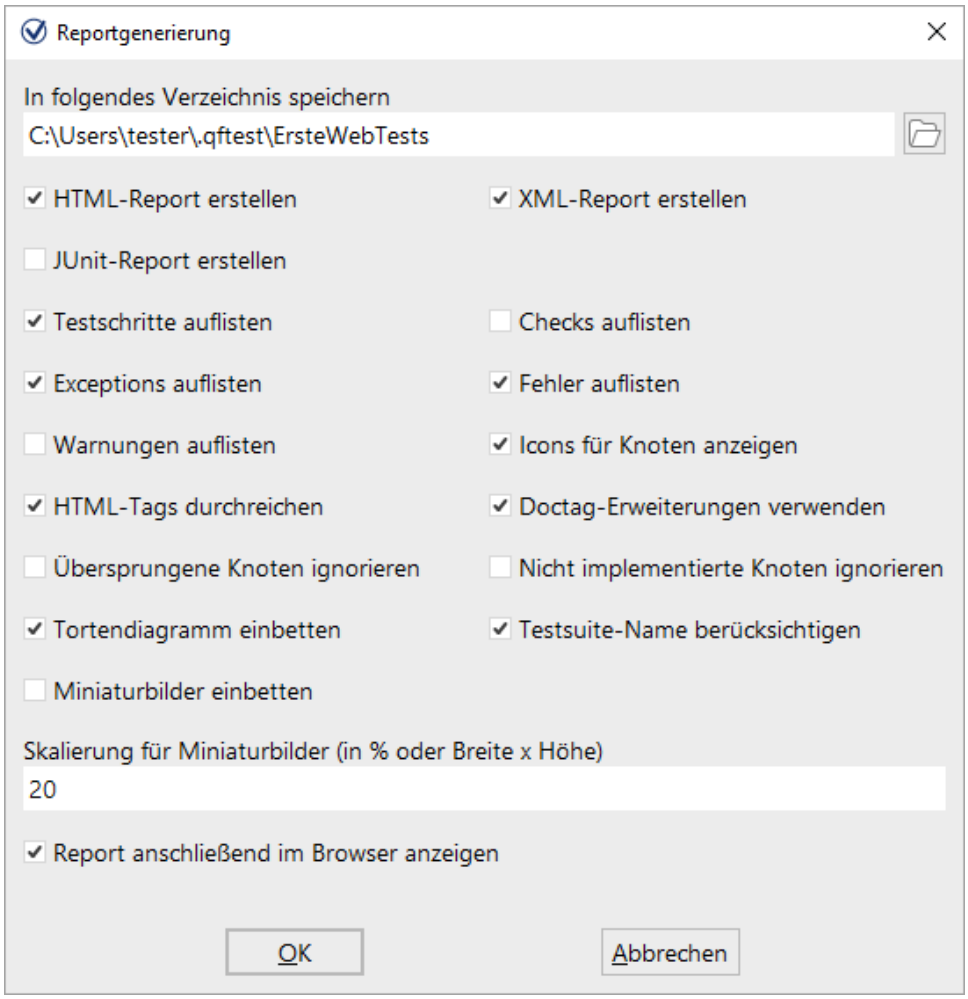

Abbildung 10.17: Auswahldialog für die Reportgenerierung

Im ersten Feld können Sie den Dateinamen des Reports festlegen. QF-Test bietet drei Arten von Reports - HTML, XML und JUnit Format. Das XML-Format können Sie verwenden, wenn Sie die Reports zum Beispiel mit Hilfe eigener XSLT-Stylesheets selbst gestalten wollen. JUnit-Reports erweisen sich als hilfreich, wenn es darum geht, Resultate in Build- oder Testmanagement-Tools zu importieren.

Wir wollen uns nun einen einfachen HTML Report zu unserem letzten Testlauf erzeugen lassen.

Aktion

- Lassen Sie bitte die vorgegebenen **Optionen unverändert**.
- Bestätigen Sie den Reportdialog mit **OK**.

Anschließend sollte sich Ihr Browser automatisch mit einem Ergebnis äquivalent zum folgenden Bild öffnen:

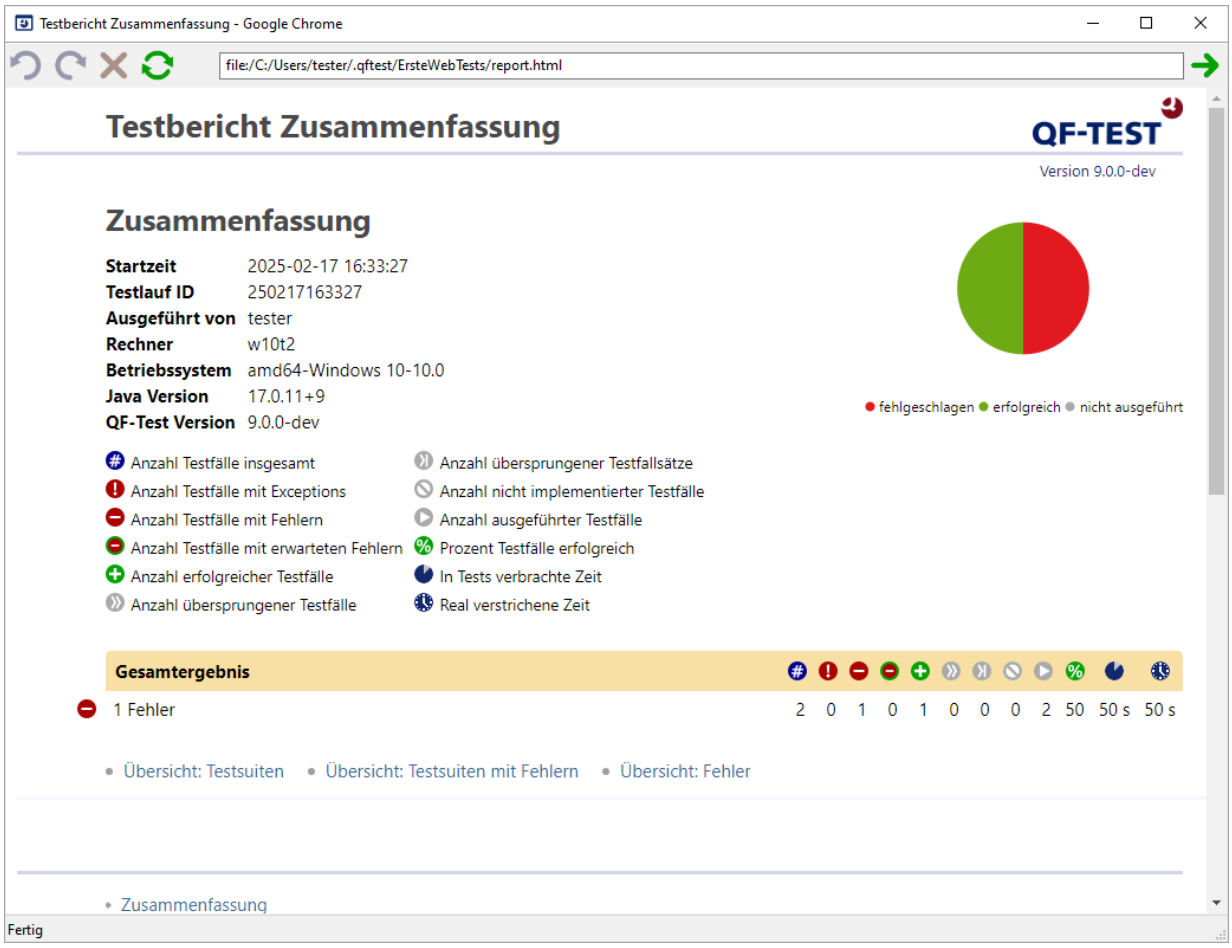

Abbildung 10.18: Ein HTML Report

Der Testbericht beginnt mit einer Zusammenfassung mit allgemeinen Systeminformationen im linken Bereich, einer Legende der verwendeten Symbole rechts, einem Überblicks-Tortendiagramm in der Mitte und dem Gesamtergebnis darunter. In unserem Fall bei einem fehlerhaften von zwei ausgeführten Testfällen eine Erfolgsquote von 50%.

Auf die Zusammenfassung folgen drei Übersichten:

1. Testsuiten, die in diesem Testlauf ausgeführt wurden.
2. Testsuiten, in denen Fehler aufgetreten sind.
3. Fehler, mit Ihrem genauen Ort und Fehlermeldung

Die Reporterstellung in QF-Test ist ein praktisches Hilfsmittel, um einen Überblick über einen Testlauf zu gewinnen und ein Dokument zu Präsentations- und Archivierungszwecken zu erstellen.

Kapitel 11

Erstellen einer eigenen Testsuite (Web)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Erstellen einer eigenen Testsuite'
https://www.qftest.com/de/yt/tutorial-2.html

In diesem zweiten Kapitel des Web Tutorials werden wir selbst Sequenzen zum Starten und Beenden eines Browsers als SUT erstellen. Zusätzlich wollen wir Aktionen und Checks aufnehmen und damit einen einfachen Testfall aufbauen.

11.1 Erzeugen der Startsequenz

Zu Beginn muss die zu testende Anwendung aus QF-Test heraus gestartet werden. Es gibt einen **Schnellstart-Assistenten**, der uns hilft, eine passende Startsequenz zu erzeugen.

Aktion

- Öffnen Sie bitte eine neue, leere Testsuite mittels Datei→Neue Testsuite...
- Öffnen Sie den Schnellstart-Assistenten über das **Menü** **Extras→Schnellstart-Assistent...**.

Der Assistent startet mit einem Willkommen und allgemeinen Informationen.

Aktion

- Nach einem kurzen Hallo drücken Sie bitte den **"Weiter" Knopf**.

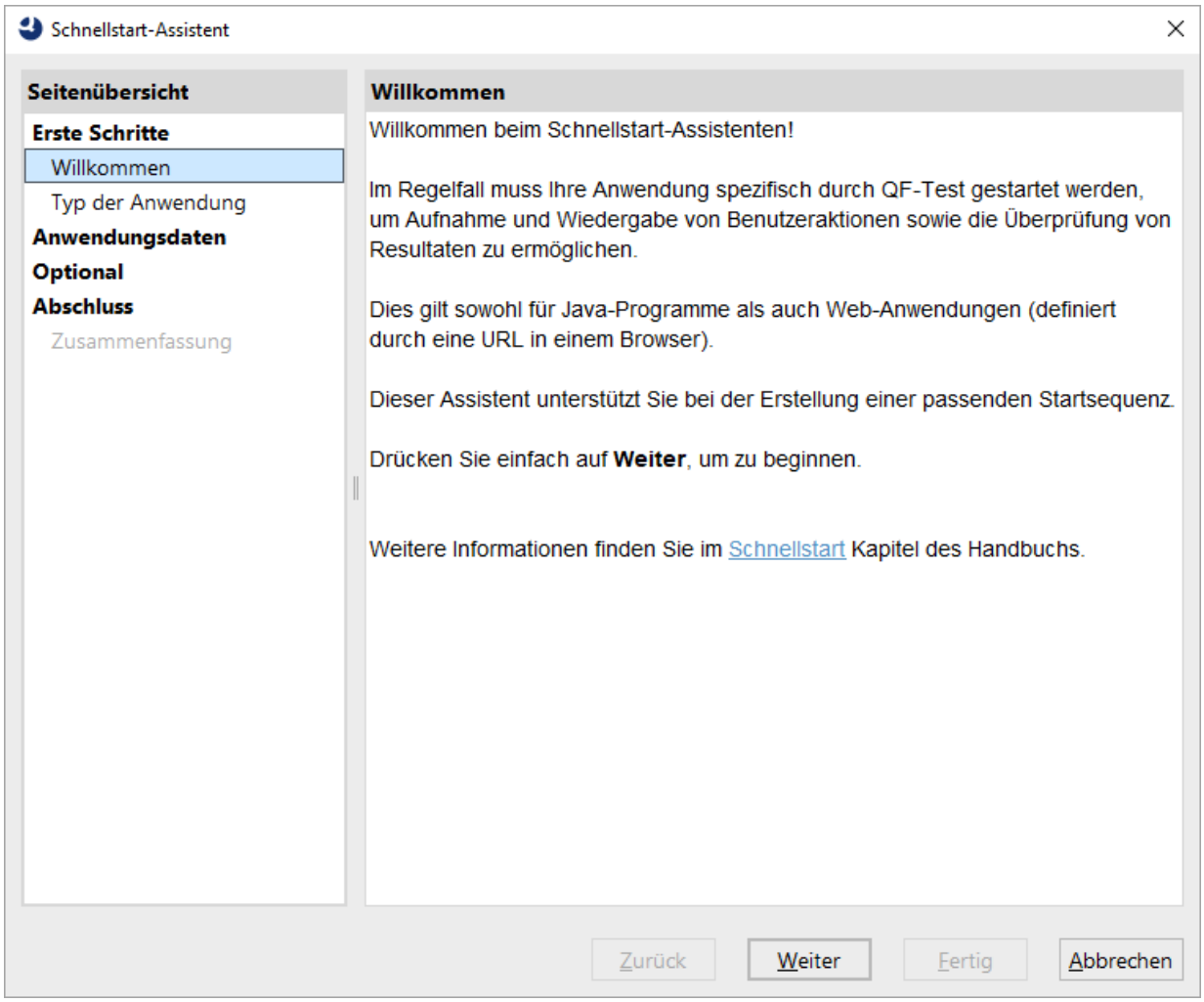

Abbildung 11.1: Der Schnellstart-Assistent

Im zweiten Schritt werden Sie dazu aufgefordert, die Art der zu testenden Applikation auszuwählen.

Aktion

- Wählen Sie bitte die zweite Option **Eine Web-Anwendung in einem Browser**.
- Drücken Sie **Weiter**.

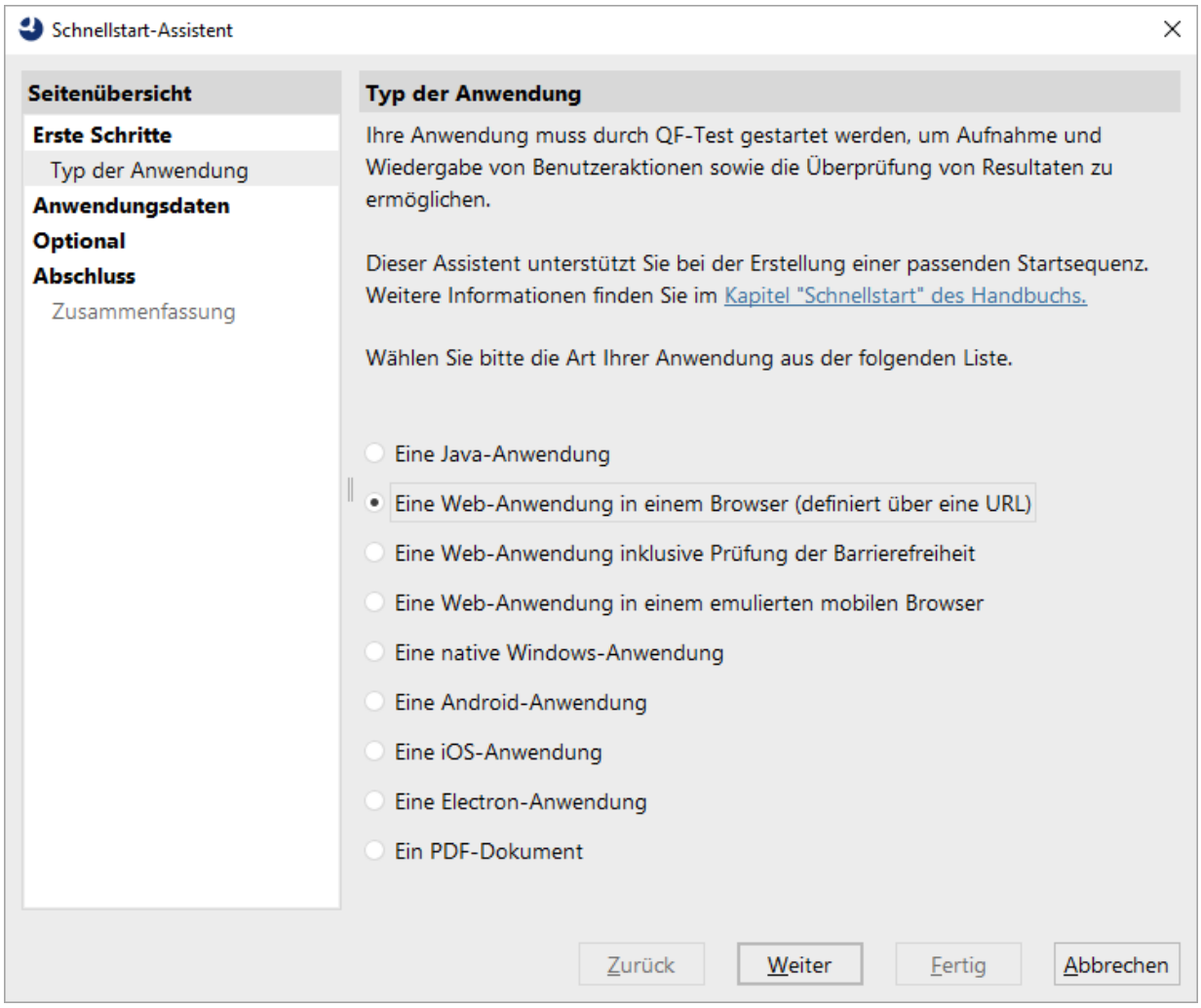

Abbildung 11.2: Auswählen der SUT Art

Im dritten Schritt wird nach der URL der zu testenden Webseite gefragt.

Typischerweise gibt man hier eine http(s) basierte URL an. Wir werden jedoch die lokal abgespeicherte CarConfigWeb Demoseite benutzen.

Aktion

- Nutzen Sie hierzu den **Webseite auswählen** 🗁 Knopf auf der rechten Seite.
- Wechseln Sie in das **Verzeichnis** `.../qftest-9.0.0/demo/carconfigWeb/html` in Ihrer QF-Test Installation.
- **Wählen** Sie dort die Datei `CarConfig.htm`
- Schließen die Dateiauswahl ab.

Hinweis Im Bild sieht man eine weitere Möglichkeit: Die Verwendung der Variablen `${qftest:dir.version}` am Beginn, die automatisch zum versionsspezifischen Installationsverzeichnis von QF-Test expandiert. Details zu speziellen QF-Test Variablen findet Sie im Handbuch Kapitel Variables.

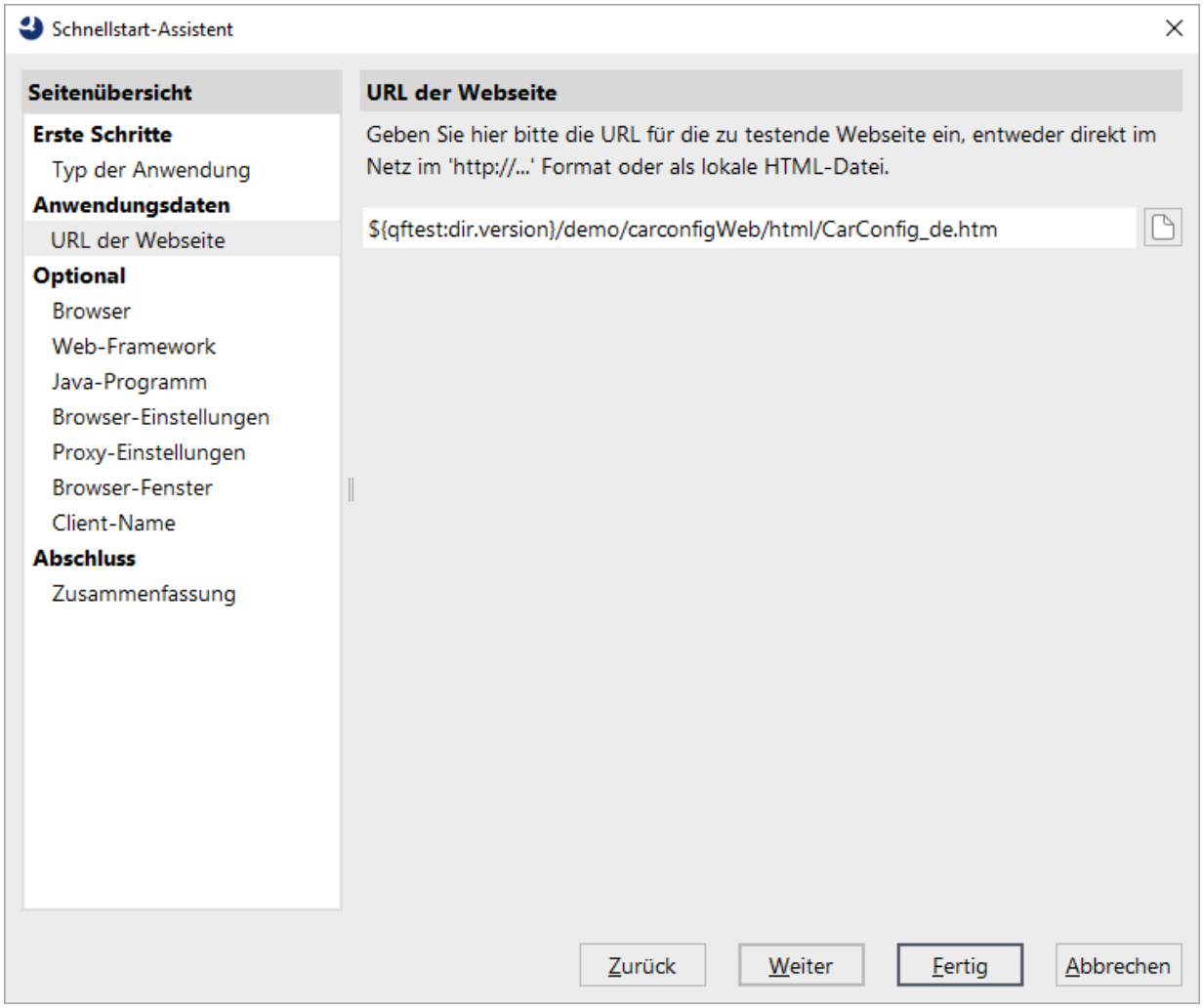

Abbildung 11.3: Auswahl der Programm Datei

Aktion

- Drücken Sie den **Fertig** Knopf, da wir die weiteren optionalen Schritte für unser einfaches Demo nicht benötigen.

Hinweis Für den verwendeten Browser bedeutet das, dass der Standardwert gewählt wird (Chrome für Windows und macOS, Firefox für Linux). Sollte dies bei Ihnen aus irgendwelchen Gründen nicht passen, nutzen Sie den optionalen nächsten Schritt im Assistenten um einen anderen Browser anzugeben.

Wir gelangen direkt zur Zusammenfassung, die beschreibt, wie es nach dem Beenden des Schnellstart-Assistenten weiter geht.

Aktion

- Drücken Sie den **Fertig** Knopf, um den Assistenten zu beenden.

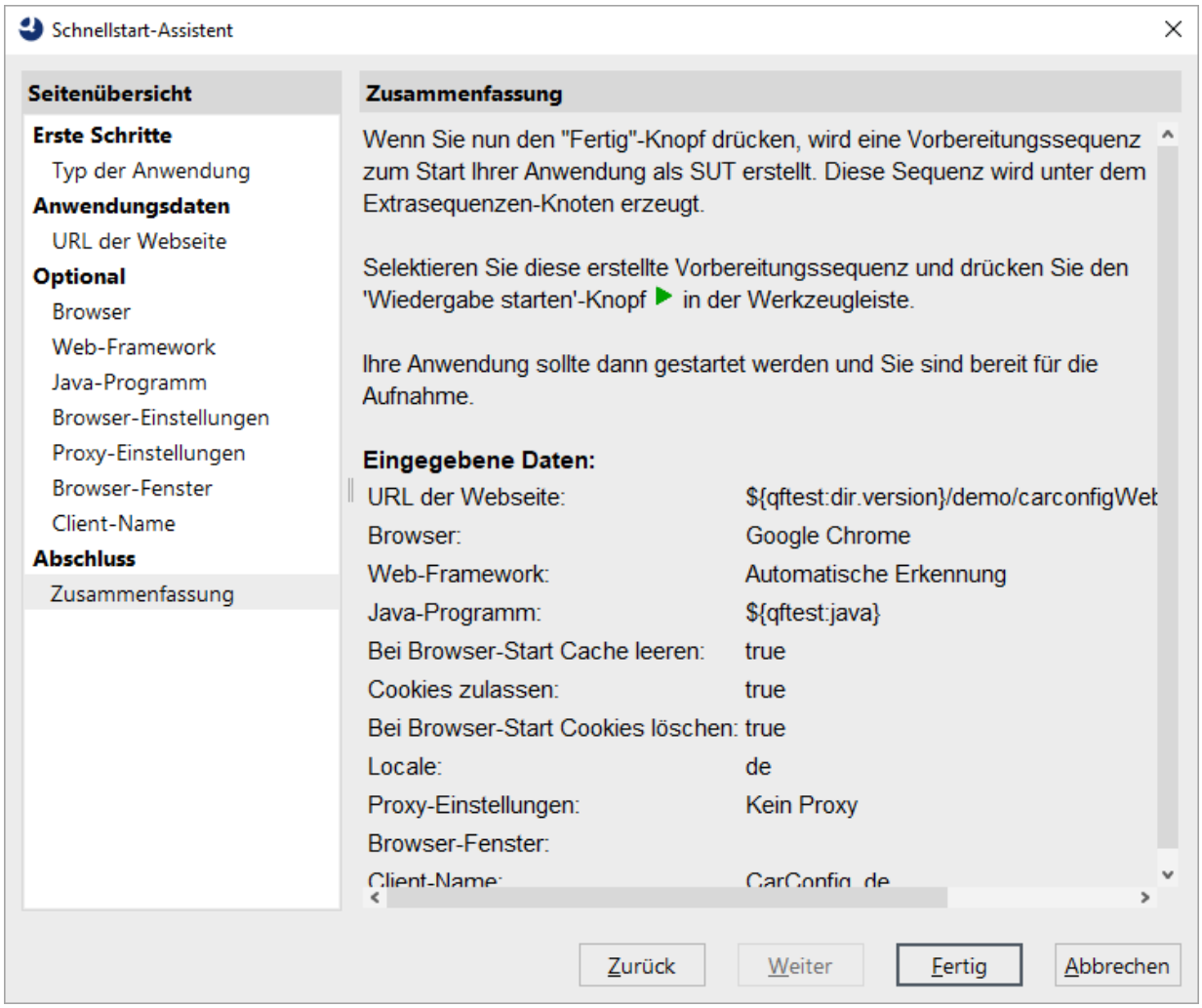

Abbildung 11.4: Zusammenfassung

Die generierte Startsequenz erscheint in den ”Extrasequenzen” der Testsuite und ist äquivalent zu der, die wir bereits aus dem letzten Kapitel kennen (Abschnitt 10.2[109]).

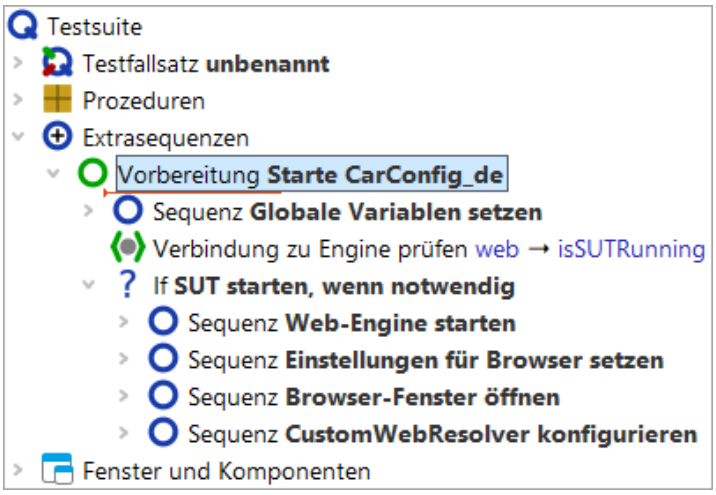

Abbildung 11.5: Generierte Startsequenz

Probieren wir sie aus:

Aktion

- Stellen Sie bitte sicher, dass der Knoten **Vorbereitung: Starte CarConfig_de** ausgewählt ist.
- **Drücken Sie** ▶ oder betätigen Sie einfach die Eingabe Taste.

Nach kurzer Zeit sollte ein Browser-Fenster erscheinen. Zuerst wird eine Seite gezeigt, die dann zum CarConfig Demo weiterleitet.

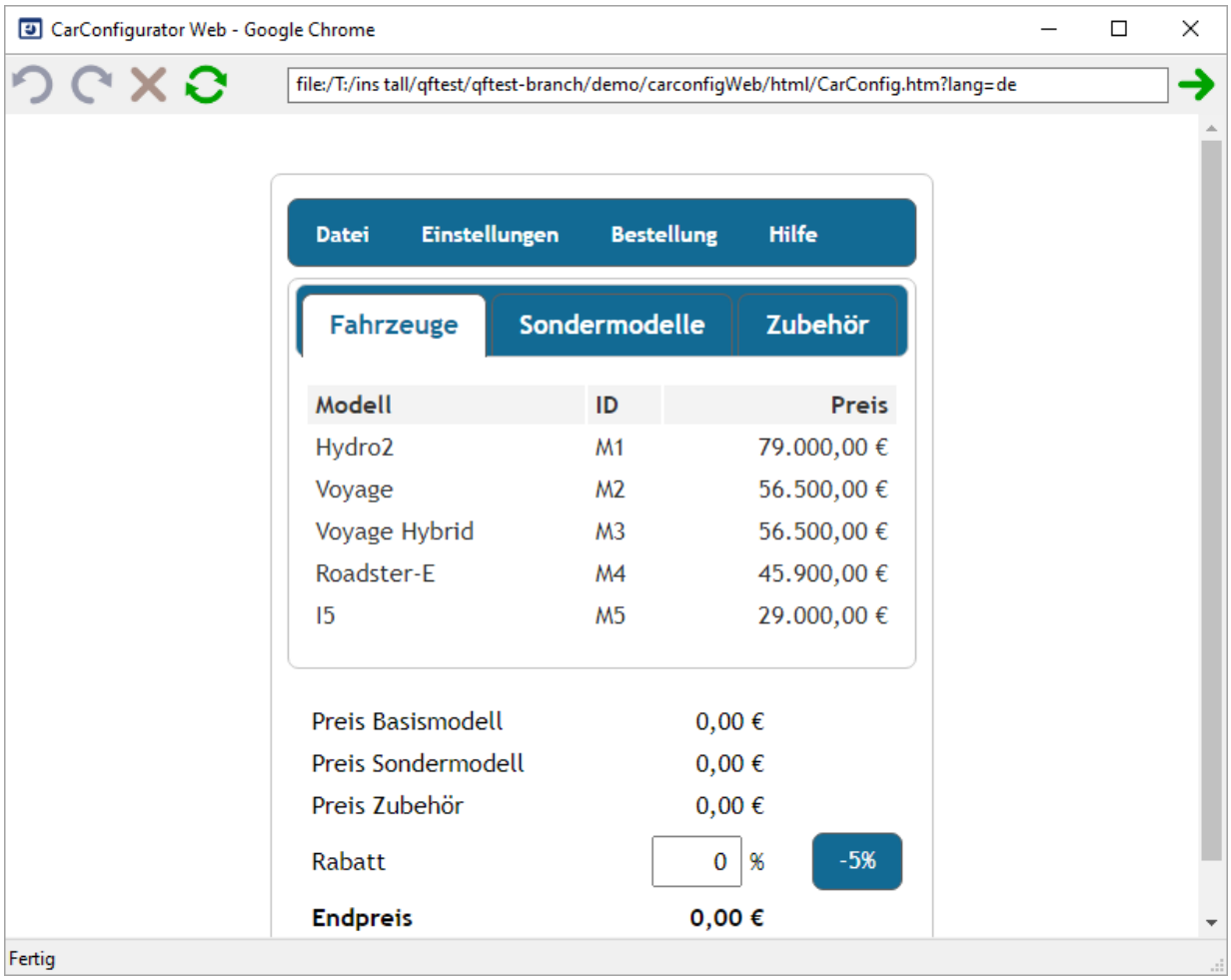

Abbildung 11.6: Das ”CarConfigurator Web” Demo im Browser

Am Ende dieses Abschnitts wollen wir unsere Testsuite speichern.

Aktion

- Drücken Sie den Knopf in der Werkzeugleiste oder nutzen Sie die Datei→Speichern Menüaktion bzw. das Tastenkürzel Strg-S.
- Im Datei-Explorer navigieren Sie in ein passendes Verzeichnis, in dem Sie Schreibrechte besitzen, z.B. `Dokumente` in Ihrem Benutzerverzeichnis.
- Geben Sie einen Namen ein z.B. `MeineErstenTests.qft`.
- Beenden Sie die Speicheraktion über den Speichern-Knopf.

11.2 Aufnehmen von Aktionen

Wir werden nun erste Aktionen in unserem Demo aufnehmen:

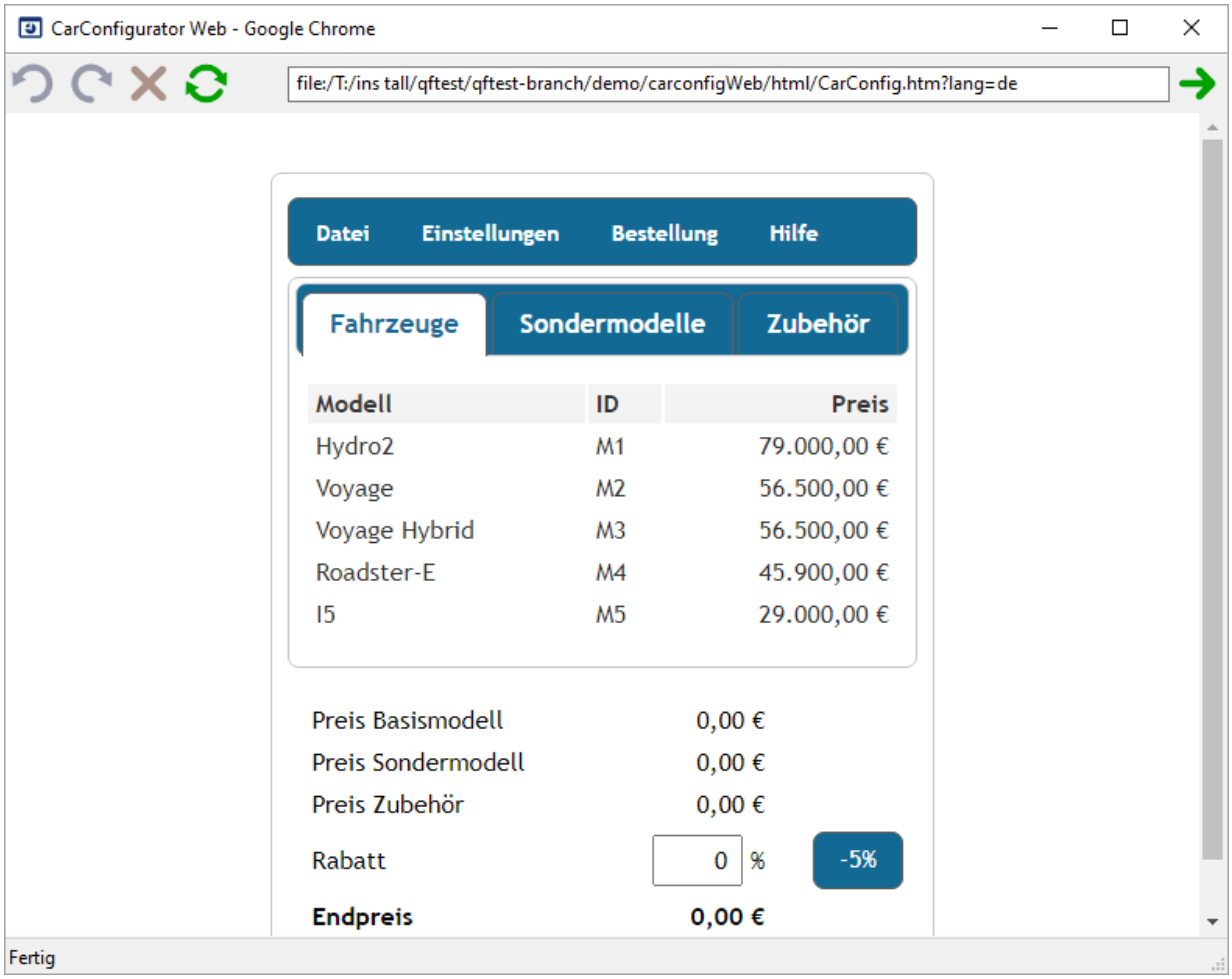

Abbildung 11.7: Aktionen im "CarConfigurator Web" Demo aufnehmen

Aktion

- Drücken Sie dazu den ● **Aufnahmeknopf**
- **Wechseln** Sie zum **"CarConfigurator Web"** Fenster. Von jetzt ab wird jede Maus- oder Tastaturaktion aufgenommen.
- Wählen Sie mit der Maus das Modell **I5** ganz unten in der Tabelle aus.
- Wechseln Sie zum Tab **Sondermodelle**.
- Wählen Sie dort das Sondermodell **Jazz** über das Dropdown-Menü.

- Zum Schluss klicken Sie wieder auf den ersten Tab **Fahrzeuge**.
- **Beenden Sie die Aufnahme**, indem Sie zurück zum QF-Test Fenster wechseln und dort den Knopf für "Aufnahme beenden" ■ drücken.

Sie finden die aufgenommene Sequenz unter dem **"Extrasequenzen"** Knoten, wie im folgenden Bild dargestellt.

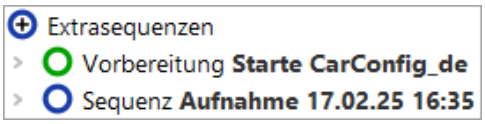

Abbildung 11.8: Der Baum nach Aufnahme der Sequenz

Als Sequenzname wird standardmäßig Datum und Zeit der Erstellung verwendet. Dieser kann anschließend in den Details rechts beliebig angepasst werden.

Aktion

- **Ändern** Sie den Sequenznamen bitte ab zu "Modell I5 Jazz wählen"
- **Öffnen** Sie die Sequenz um die enthaltenen Knoten zu sehen. Es sollten die erwarteten Mausklicks sein. Sie sollten sogar in der Lage sein, die angesprochenen Komponenten zuordnen zu können.

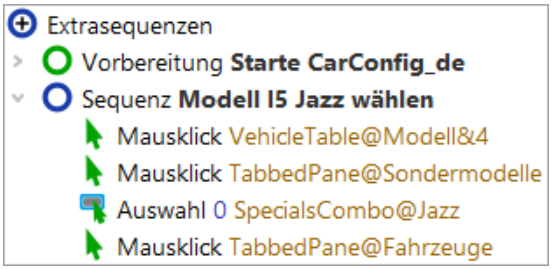

Abbildung 11.9: Die umbenannte Sequenz

Wir wollen nun die aufgenommene Sequenz abspielen.

Aktion

- Markieren die Sequenz **Modell I5 Jazz wählen**.
- Drücken Sie ▶ **Wiedergabe**.

Sie sollten die exakt gleichen Aktionen sehen, die Sie zuvor aufgenommen haben.

Den aufgenommenen Ablauf sollten Sie auch wiederholt ohne Fehler abspielen können. Rechts unten im Fenster der Testsuite sollte "Beendet: Keine Fehler" zu sehen sein.

Hinweis In der Aufnahme hat QF-Test die funktionalen Einheiten wie Datentabelle, Tab-Reiter und Dropdown-Menü erkannt. Die Ansprache der Unterelemente erfolgt über Index, Details siehe Adressierung von Unterelementen von Tabellen, Bäumen und Listen[146]. Voraussetzung hierfür ist eine passende Konfiguration der Komponentenerkennung. Weitere Informationen siehe Web-Komponentenerkennung[148].

11.3 Aufnahme von Checks

Um das Verhalten des Clients zu überprüfen, verwenden wir Check-Knoten, mit denen man Zustand und Eigenschaften von Elementen abfragen kann. Auch Checks können aufgezeichnet werden.

Aktion

- Zum Aufnehmen eines Checks drücken Sie die den ✓ **"Check aufnehmen"** Knopf.
- Wechseln Sie zum Fenster des SUT. Es erscheint ein Rahmen um die Komponente, über der sich der Mauszeiger befindet.
- Klicken Sie mit der **rechten Maustaste** auf das Wertfeld des **Endpreises**. Das erscheinende Kontextmenü erlaubt Ihnen die Auswahl eines Checks. Die Liste der verfügbaren Checks hängt vom Typ der Komponente ab.
- Wählen Sie den **ersten Eintrag "Text"**, um den textuellen Wert des Feldes zu überprüfen.
- Beenden Sie die Aufnahme durch Drücken des **Stoppknopfs** ■ .

Wieder taucht die neue Aufnahme unter den "Extrasequenzen" auf.

Aktion

- **Benennen** Sie den Sequenzknoten um auf den Namen **"Endpreis prüfen"**.
- **Öffnen** Sie anschließend den Sequenzknoten, um den Checkknoten zu sehen.

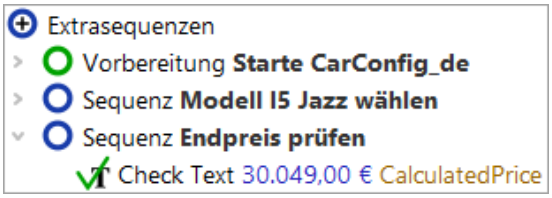

Abbildung 11.10: Die aufgenommene Check-Sequenz

In den Details des ”Check Text” Knotens sieht man ebenfalls den erwarteten Wert des Endpreis Feldes.

Aktion

- Auch diese Sequenz können Sie wieder **selbst ausführen**, um die Wiedergabe zu testen.

Im nächsten Schritt wollen wir aus den beiden Sequenzen einen richtigen Testfall aufbauen.

11.4 Erstellen einer Testsuite

Die Basisstruktur unterhalb des Wurzelknotens einer Testsuite ist durch folgende Knoten festgelegt:

- Eine beliebige Anzahl von ”Testfallsatz” und ”Testfall” Knoten, um funktionale Tests zu spezifizieren und zu strukturieren.
- ”Prozeduren” - hier können wiederverwertbare Sequenzen in Prozeduren organisiert werden
- ”Extrasequenzen” - unsere Spielwiese für Aufnahmen etc.
- ”Fenster und Komponenten” - das eigentliche Herz der Testsuite. Hier sind alle aufgenommenen Fenster und Komponenten des SUT mit ihren Eigenschaften enthalten

Funktionale Testfälle werden durch ”Testfall” Knoten repräsentiert und mittels ”Testfallsatz” Knoten gruppiert bzw. strukturiert.

”Vorbereitung” und ”Aufräumen” Knoten können Aktionen enthalten, um einen wohldefinierten Zustand vor und nach einem Testfall sicherzustellen.

Aktion

- Wir beginnen mit dem **Umbenennen** des ”Testfallsatz” Knotens von ”unbenannt” in ”Demo Tests”.
- Falls ein **Dialog** bzgl. der Aktualisierung von Verweisen erscheint, können wir diesen einfach mit **”Ja”** beantworten.
- Der nächste Schritt ist, den vom Schnellstart-Assistenten erzeugten Knoten **”Vorbereitung”** in den **”Testfallsatz” zu verschieben** und zwar an die **erste Position** vor den enthaltenen Testfall. Das Verschieben kann mit Hilfe der Maus (Drag&Drop), des Kontextmenüs (rechte Maustaste Ausschneiden/Einfügen) oder der Tastenkombination Strg-X und Strg-V durchgeführt werden.

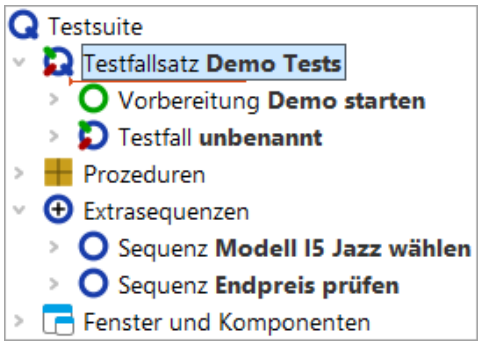

Abbildung 11.11: Beginn der Strukturierung

Als Nächstes gilt es, aus den beiden vorher aufgezeichneten Sequenzen einen Testfall zu machen.

Aktion

- **Benennen** Sie dazu den Testfall Knoten von ”unbenannt” in ”Erster” um.
- **Öffnen** Sie den Testfall Knoten durch einen Klick auf das ’>’ Symbol.
- **Verschieben** Sie die beiden Sequenzen aus den ”Extrasequenzen” in den Testfall.

Wenn Sie den Testfall Knoten nicht öffnen, versucht QF-Test die Sequenzknoten hinter dem Testfall Knoten auf der gleichen Ebene einzufügen. Dies ist jedoch für Sequenzknoten nicht zulässig.

QF-Test nimmt immer Sequenzen auf. Diese haben die gleiche Funktion wie Testschritte. Testschritte werden jedoch im Bericht aufgeführt. Man kann sie ineinander umwandeln, was wir Ihnen in den nächsten Schritten informationshalber zeigen wollen.

Aktion

- Öffnen Sie das **Kontextmenü** für den **ersten** der beiden Sequenzknoten mit der rechten Maustaste.
- Wählen Sie Knoten konvertieren in...→Testschritt
- Führen Sie dasselbe für den **zweiten** Sequenzknoten durch.

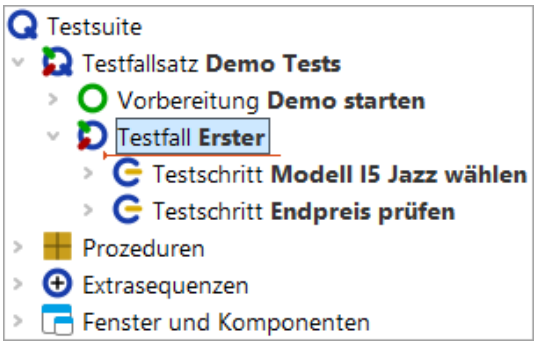

Abbildung 11.12: Der Baum nach der Neustrukturierung

Damit haben wir die wichtigsten Schritte zur Strukturierung unserer Testsuite abgeschlossen.

11.5 Beenden der Anwendung

Was uns als Basiselement noch fehlt, ist eine Aufräumsequenz, die das SUT sauber beendet.

Es gibt verschiedene Wege eine Anwendung zu beenden, z.B. über den "Fenster schließen" Knopf rechts oben, durch Drücken von Alt-F4 oder das Menü Datei→Beenden. Alle diese Varianten lassen sich direkt aufzeichnen.

Wir werden die erste Möglichkeit nutzen:

Aktion

- **Aufnahme starten** ● .
- Drücken Sie den **Fenster-schließen Knopf** des Browsers.
- Das Fenster der Demoanwendung verschwindet.
- **Aufnahme beenden** ■ .
- Benennen Sie die aufgenommene Sequenz **in "Demo beenden" um**.
- Öffnen Sie das **Kontextmenü** für den Sequenzknoten und wählen Sie den Menüpunkt **Knoten konvertieren in...→Aufräumen** .
- Zuletzt **verschieben** Sie den Aufräumenknoten nach oben, so dass er **der letzte Knoten im Testfallsatz** ist.

Hinweis Der Aufräumknoten kann nur per Drag and Drop in den Testfallsatz verschoben werden, wenn dessen letzter Kindknoten eingeklappt ist. Um einen Knoten während einer Drag and Drop Operation ein- oder auszuklappen, verweilen Sie einen Moment mit dem Mauszeiger über dem Dreieck neben dem Knoten.

Sie sollten folgendes Resultat erhalten:

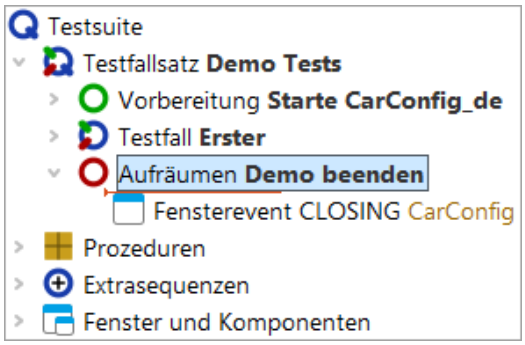

Abbildung 11.13: Die einfache Aufräumsequenz

Damit haben wir die wichtigsten Schritte zur Strukturierung unserer Testsuite abgeschlossen.

11.6 Gesamte Suite ausführen

Als Abschluss wollen wir unsere neue Suite ausführen:

Aktion

- **Beenden** Sie dazu nun bitte den SUT Client, falls er läuft.
- **Markieren** Sie den "Testsuite" **Wurzelknoten**.
- Führen Sie diesen durch Drücken von "Wiedergabe" ▶ oder der Eingabe Taste aus.

Das SUT sollte erscheinen, der Testfall aufgeführt und das SUT wieder beendet werden.

Wie wir wissen, wird das Ergebnis des Testlaufs im Protokoll festgehalten:

Aktion

- Um dieses anzuschauen, können wir den Button **"Protokoll anzeigen"** in der Werkzeugleiste oder alternative die Tastenkombination Strg-L nutzen.

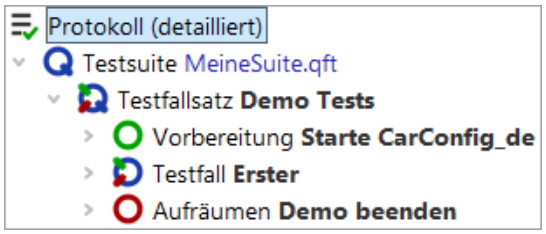

Abbildung 11.14: Der Protokollbaum der eigenen Testsuite

Wir hatten bereits im ersten Kapitel gesehen, wie das Protokoll für die Fehleranalyse genutzt werden kann.

Kapitel 12

Eine Prozedur erstellen (Web)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Eine Prozedur erstellen'
https://www.qftest.com/de/yt/tutorial-3.html

In Kapitel eins und zwei haben Sie gelernt wie man eine Applikation über QF-Test startet, so dass Maus- und Tastatureingaben aufgenommen werden können, auch wie man Checks aufnimmt und wie man das Ergebnis zu einem Testfall zusammenfasst. Diese Herangehensweise ist ausreichend, solange die Tests einfach und nicht allzu viele sind. Sobald jedoch die Zahl der Tests zunimmt, ist es wichtig, sogenannte "Prozeduren" einzusetzen.

Prozeduren sind ein Mittel um Sequenzen wiederverwendbar zu machen und damit Doppelvorkommen zu vermeiden. Dies ist wichtig, um eine einfache und effiziente Wartbarkeit von Tests über die Zeit zu erreichen.

Prozeduren können in Packages gruppiert werden. Prozeduren und Packages sind die Basis für die Modularisierung der Tests.

12.1 Wiederverwendbare Abschnitte identifizieren

In diesem Abschnitt werden wir die Testsuite ErsteWebTests.qft, die Sie bereits aus Kapitel 1 kennen, weiterentwickeln.

Aktion

- **Kopieren Sie ErsteWebTests.qft** aus dem Unterverzeichnis `qftest-9.0.0/doc/tutorial` der QF-Test Installation in ein Arbeitsverzeichnis und
- **öffnen Sie ErsteWebTests.qft**.

- Wenn Sie die Änderung, die wir an der Demo-Testsuite vornehmen werden, sichern wollen, so **speichern Sie diese in einem Arbeitsverzeichnis** wie am Ende von Abschnitt 11.1 (123) beschrieben.

Bitte sehen Sie sich den Testschritt "Zurücksetzen" in den beiden Testfällen an. Die beiden Testschritte sind identisch.

Abbildung 12.1: Zwei identische Testschritte

Gemäß obiger Überlegungen wäre es also sinnvoll, den Testschritt in eine Prozedur umzuwandeln.

12.2 Manuelle Erstellung von Prozeduren

Es gibt mehrere Methoden Prozeduren zu erstellen und Prozeduraufrufe einzufügen. Wir fangen mit der manuellen an, bei der ein (leerer) Prozedurknoten eingefügt wird, in den dann die entsprechenden Aktionen verschoben werden. Danach erstellen wir den zugehörigen Prozeduraufruf.

Es ist gut, wenn man diese grundlegenden Schritte kennt. Es gibt jedoch eine zweite, elegantere Methode Prozeduren zu erstellen, die wir im Anschluss zeigen werden.

Also los, fügen wir eine Prozedur von Hand ein: Wir beginnen mit dem Anlegen des Prozedurknotens, dem wir einen geeigneten Namen geben.

Aktion

- **Öffnen Sie den Prozedur Knoten** und achten Sie darauf, dass er auch selektiert (blau markiert) ist.

- Wählen Sie **Einfügen→Prozedurknoten→Prozedur**.
- Tragen Sie als Name "**zurücksetzen**" ein. Die anderen Felder brauchen nicht befüllt zu werden.
- Drücken Sie **OK** um die Erstellung der Prozedur abzuschließen.
- **Öffnen Sie** die neu erstellte "zurücksetzen" Prozedur.

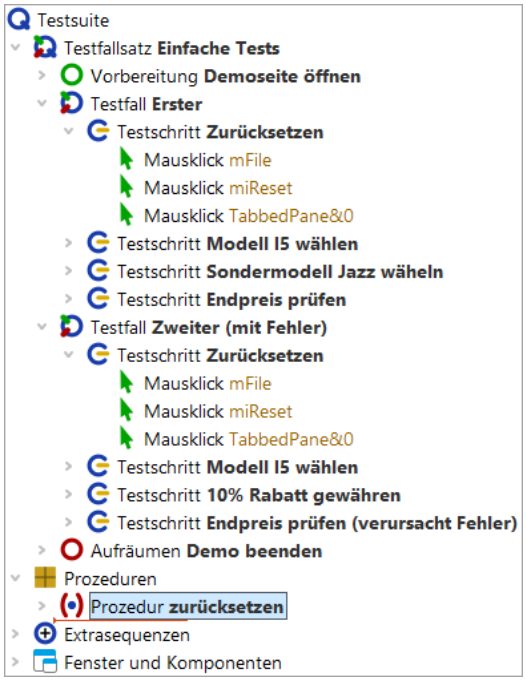

Abbildung 12.2: Prozedurknoten erstellen

Im zweiten Schritt befüllen wir die Prozedur mit den entsprechenden wiederverwendbaren Aktionen.

Aktion

- **Selektieren Sie** die drei "Mausklick" Knoten im Testschritt. Um mehr als einen Knoten zu selektieren, klicken Sie den ersten der Knoten an, dann drücken Sie die Shift Taste und klicken den letzten der zu selektierenden Knoten während Sie die Shift Taste gedrückt halten.
- **Verschieben** Sie diese nach unten in die Prozedur, z.B. mit der Maus (Drag and drop) oder über Ausschneiden/Einfügen im Menü Bearbeiten oder über das Kontextmenü.

Abbildung 12.3: Prozedur mit Inhalt befüllen

Im dritten Schritt fügen wir einen Prozeduraufruf an Stelle der drei verschobenen Mausklicks ein.

Aktion

- **Selektieren** Sie den **Testschritt "Zurücksetzen"**, der geöffnet sein sollte.
- **Wählen Sie** den Menüpunkt Einfügen→Prozedurknoten→Prozeduraufruf oder verwenden Sie das Tastaturkürzel Strg-A.

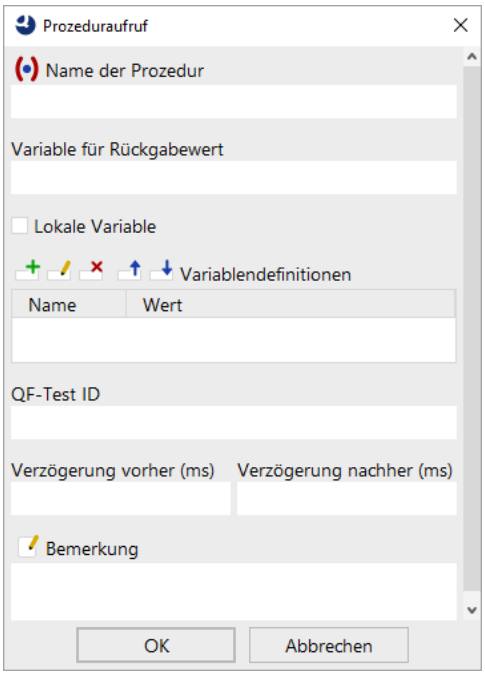

Abbildung 12.4: Prozeduraufruf einfügen

Aktion

- **Drücken Sie** den Prozedur-Auswahlknopf in dem Dialog links neben der Beschriftung "Name der Prozedur".
- **Wählen Sie "zurücksetzen"** aus der Prozedurliste. Weitere Eingaben sind nicht nötig.
- Drücken Sie **OK** in beiden Dialogen um die Erstellung des Prozeduraufrufs abzuschließen.

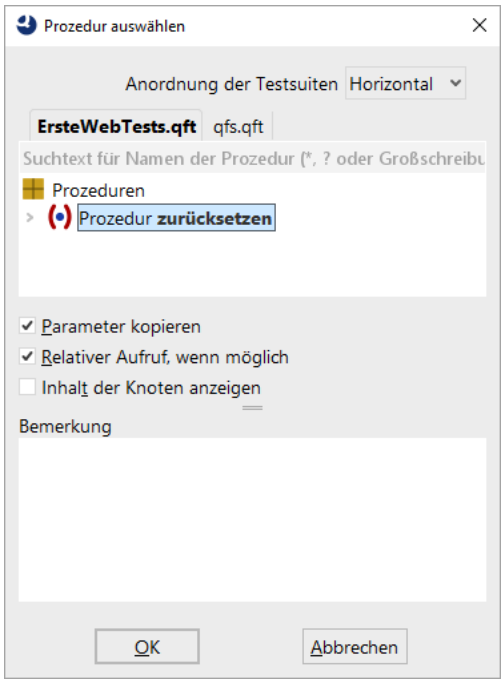

Abbildung 12.5: Prozedur auswählen

Um wirklich einen Mehrwert durch die Prozedur zu erlangen, müssen wir nun den Inhalt des Testschritts im zweiten Testfall ebenfalls durch einen Aufruf der Prozedur ”zurücksetzen” ersetzen.

Sie können dies wie oben beschrieben tun oder Sie führen folgende **alternative Schritte** zur Erstellung des Prozeduraufrufs aus:

Aktion

- **Öffnen Sie** den Testschritt ’Zurücksetzen’ des zweiten Testfalls.
- **Löschen Sie** die drei Mausklick Knoten daraus.
- **Selektieren Sie** den Prozedurknoten ”zurücksetzen”.
- **Ziehen Sie** den Prozedurknoten ”zurücksetzen” mit der Maus in den Testschrittknoten. Kopieren/einfügen kann ebenfalls verwendet werden. Dadurch wird der Prozedurknoten nicht verschoben, sondern ein entsprechender Prozeduraufruf erzeugt.

Die Testsuite sollte anschließend wie folgt aussehen:

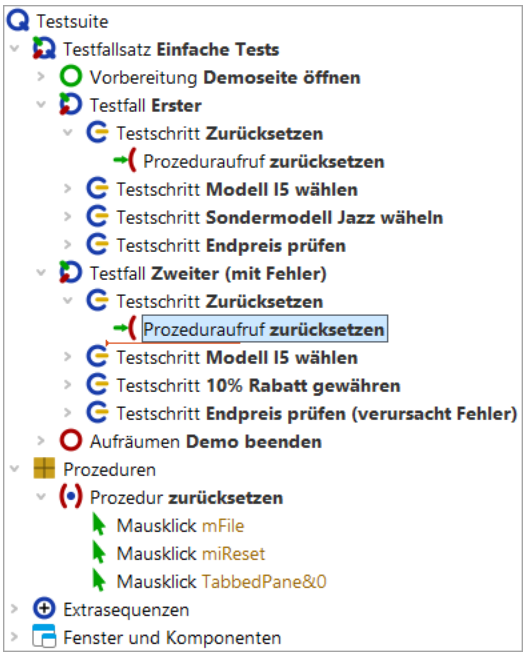

Abbildung 12.6: Testsuite mit Prozedur

Wenn Sie nun die Testfälle ausführen, sollten diese funktionieren wie zuvor. Im Protokoll sind jetzt auch die Prozeduraufrufe und deren Ausführung zu sehen.

12.3 Knoten in Prozedur konvertieren

Wie bereits am Anfang des letzten Abschnitts erwähnt, bietet QF-Test eine Alternative um Prozeduren wesentlich schneller zu erstellen.

Aktion

- **Markieren Sie den Testschritt oder Sequenz-Knoten**, der die wiederverwendbaren Schritte enthält, die zur Prozedur umgewandelt werden sollen.
- Wählen Sie den Menüpunkt **Operationen→Knoten konvertieren in→Prozedur** aus oder verwenden Sie das Tastaturkürzel Strg-Umschalt-P.

Wie Sie sehen, ist der Testschritt bzw. der Sequenzknoten verschwunden. Anstelle dessen befindet sich ein Prozeduraufruf. Außerdem wurde eine Prozedur mit dem Namen des ehemaligen Testschritts bzw. der Sequenz im Abschnitt "Prozeduren" erstellt. Sie enthält genau die gleichen Kindknoten wie zuvor der Testschritt bzw. die Sequenz.

Bei der Aufnahme einer Sequenz in QF-Test hat sich das Vorgehen bewährt, der Sequenz sofort einen Namen zu geben und sie anschließend in eine Prozedur zu konvertieren. Auch wenn man nur eine Vermutung hat, dass sich die aufgenommenen Schritte irgendwo wiederholen könnten.

Kapitel 13

Komponenten (Web)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Komponenten'
https://www.qftest.com/de/yt/tutorial-4.html

Werfen wir nun einen Blick auf den letzten verbleibenden Bereich des Testsuite Fensters, den Fenster und Komponenten Knoten. Zuvor möchten wir Ihnen jedoch zeigen, wie Unterelemente von Komponenten wie Tabellen, Bäumen und Listen adressiert werden.

13.1 Adressierung von Unterelementen von Tabellen, Bäumen und Listen

Unterelemente von Tabellen, Bäumen und Listen werden über Indizes angesprochen. Die wichtigsten beiden Indextypen sind der numerische und der Textindex. Zur Demonstration nehmen wir als nächstes einen Mausklick auf eine Tabellenzelle auf und sehen uns die aufgenommene QF-Test ID der Komponente näher an.

Aktion

- **Starten Sie das CarConfig Demo**, falls dieses nicht bereits läuft. Führen Sie dazu den Vorbereitung Knoten in der Testsuite aus.
- **Aktivieren Sie den Aufnahmemodus** über "Aufnahme starten" ● .
- **Klicken Sie auf eine Tabellenzelle**, z.B. das erste Modell.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .

Den aufgenommenen Mausklick finden Sie im Bereich Extrasequenzen.

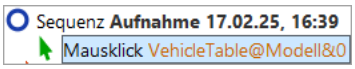

Abbildung 13.1: Adressierung einer Tabellenzelle

Die aufgenommene QF-Test ID der Komponente ist `VehicleTable@Modell&0`. Sie setzt sich aus den folgenden Teilen zusammen:

- `VehicleTable` ist die QF-Test ID der Komponente der Tabelle selbst.
- `@` und `&` trennen die einzelnen Teile voneinander. Gleichzeitig definieren sie den Typ des darauf folgenden Index: auf `@` folgt ein Textindex, auf `&` ein numerischer Index.
- `Modell` ist der Textindex für die Spalte mit der Überschrift 'Modell'.
- `0` ist der numerische Index für die erste Tabellenzeile.

Hinweis Numerische Indizes beginnen immer mit 0.

Sie können beide Indextypen für Zeilen und Spalten verwenden. Dabei ist nur wichtig, dass das Trennzeichen und der Typ des folgenden Index zusammenpassen.

Aktion

- **Ändern Sie die QF-Test ID der Komponente** so, dass das dritte Preisfeld adressiert wird. Verwenden Sie dafür numerische Indizes.

Die Lösung hierfür lautet `VehicleTable&1&2`.

Um das Modell 'I5' über Textindizes anzusprechen, tragen Sie `VehicleTable@Modell@I5` ein. Das gleiche Feld kann man numerisch mittels `VehicleTable&0&4` ansprechen oder mit gemischten Indizes mittels `VehicleTable&0@I5` oder `VehicleTable@Modell&4`.

Der dritte Indextyp von QF-Test ist ein Index mit regulärem Ausdruck. Reguläre Ausdrücke werden verwendet, um Zeichenketten durch einen Ausdruck zu ersetzen, der verschiedene Zeichenketten adressieren kann. Sozusagen eine "Sternchensuche", wobei reguläre Ausdrücke wesentlich mächtiger sind und eine eigene Syntax besitzen. Eine genauere Beschreibung regulärer Ausdrücke finden Sie im Handbuch. Beispiel: Das Modell 'I5' könnte man also auch über `VehicleTable@Modell%I.*` ansprechen.

Listen werden analog zu Tabellen adressiert, nur dass sie nur einen einzigen Index benötigen.

Bäume haben ebenfalls nur einen einzigen Index. Dieser ist der Pfad durch den Baum zu dem adressierten Baumknoten. Der Pfad setzt sich aus den einzelnen Knoten zusammen, die durch Schrägstriche ("/") voneinander getrennt werden.

Aktion

- **Starten Sie das CarConfig Demo**, falls dieses nicht bereits läuft. Führen Sie dazu den Vorbereitung Knoten in der Testsuite aus.
- **Öffnen Sie das Baum-Beispiel:** Wählen Sie im CarConfig Demo den Menüpunkt Einstellungen→Sondermodelle..., selektieren Sie ein Modell und drücken die Schaltfläche 'Details'.
- **Aktivieren Sie den Aufnahmemodus** über "Aufnahme starten" ● .
- **Klicken Sie auf einen Baumknoten**, z.B. 'Beschreibung'.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .

Für den Baumknoten "Beschreibung" wird die folgende QF-Test ID der Komponente aufgenommen: `DetailsTree@/Information/Beschreibung`. Die einzelnen Bestandteile davon sind:

- `DetailsTree` ist die QF-Test ID der Komponente des Baums selbst.
- `@` trennt die QF-Test ID der Komponente des Baums vom Index. Die Syntax ist hierbei analog zu der der Tabellenindizes, d.h. `@` steht für einen Textindex, `&` für einen numerischen Index und `%` für einen Index mit regulärem Ausdruck.
- `/Information/Beschreibung` ist der Textindex für den Baumpfad zum Knoten 'Beschreibung'.

Wenn Sie den Knoten über einen numerischen Index adressieren wollen, verwenden Sie `DetailsTree&/0/1`.

13.2 Web-Komponentenerkennung

Die Adressierung von Unterelementen komplexer Komponenten wie Listen, Tabellen und Bäumen, via Index, wie im letzten Abschnitt beschrieben, ist nur möglich, weil im Rahmen der erzeugten Vorbereitung eine Konfiguration der Komponentenerkennung erfolgt. Damit wird QF-Test mitgeteilt, wie bestimmte Funktionalitäten wie Textfelder, Buttons, Checkboxen, Datentabellen, Bäume etc. im HTML-Code der Applikation implementiert sind.

Wenn die Funktionalität einer Komponente bekannt ist, hat dies bei der Aufnahme folgende Vorteile:

- Unterelemente werden, wie bereits bekannt, über Index angesprochen und nicht als Einzelkomponente aufgenommen.

- Bei Mausklicks wird die Stelle, an die später der Klick abgespielt werden soll, hinterlegt. So zum Beispiel Buttons am besten mittig, Textfelder hingegen genau an der gleichen Stelle wie bei der Aufnahme, damit Text bei Bedarf passend eingefügt wird.
- Für die Aufnahme von Checks passt QF-Test das Popup-Menü entsprechend an. So können zum Beispiel für Tabellen Checks für ganze Spalten oder Zeilen aufgenommen werden. Bei Textfeldern hingegen wird ein zusätzlicher Check bezüglich Editierbarkeit angeboten.
- Auch bei der Speicherung von Merkmalen für die Komponentenerkennung berücksichtigt QF-Test die Funktionalität. Beim Button ist zum Beispiel die Beschriftung des Buttons selbst ein wichtiges Wiedererkennungsmerkmal. Bei Textfeldern hingegen ist es wenig sinnvoll, den jeweiligen Text abzuspeichern. Hier sucht QF-Test nach einer passenden Beschriftung und speichert diese ab.

Außerdem wirken sich die Informationen, die auf Grund der bekannten Funktionalität zusätzlich abgespeichert wurden, positiv auf die Stabilität der Tests aus.

Die Konfiguration der Komponentenerkennung ist notwendig, da Web-Anwendungen einen begrenzten Satz von Basiskomponenten haben, die in sehr flexibler Weise genutzt werden können, um komplexe Komponenten und Designs aufzubauen. Ein Beispiel sind HTML Tabellen, die sowohl für das Seitenlayout genutzt werden als auch für die Darstellung logischer Tabellen wie der Fahrzeugtabelle. Ohne zusätzliche Informationen kann QF-Test nicht wissen, wo genau die Layout-Tabelle endet und die logische Tabelle beginnt. Die Tabellenzelle würde nicht als solche erkannt und als eigene Komponente, z.B. `VehicleTable.td`, aufgezeichnet. Das ”td” kommt vom HTML Table Data Element ”TD”, das im vorliegenden HTML-Code die Tabellenzelle darstellt.

In der Vorbereitung in der Demosuite `ErsteWebTests.qft` aus dem ersten Kapitel oder auch in der über den Schnellstart-Assistenten im zweiten Kapitel erstellte, erfolgt die Konfiguration direkt nach dem Browser-Start. Das folgende Bild zeigt den Knoten für die Registrierung der Resolver in der Testsuite `ErsteWebTests.qft`.

Abbildung 13.2: Web Resolver Registrierung in der ErsteWebTests.qft

Über den Knoten erhält QF-Test z.B. die Information, welches HTML-Element eine Datentabelle enthält und welche HTML-Elemente darin dann die Zeilen und Datenzellen darstellen. Diese Konfiguration wurde speziell für den ”CarConfigurator Web” erstellt.

Hinweis An dieser Stelle ist es nur wichtig, dass Sie wissen, dass es verschiedene Möglichkeiten gibt, die Komponentenerkennung durch Resolver zu verbessern.

QF-Test analysiert automatisch die Struktur von Web-Anwendungen, erkennt die meistverwendeten Web GUI Toolkits (wie Angular Material, ExtJS, GWT, SmartGWT, Vaadin,...) und installiert passende vordefinierte Resolver.

Offensichtlich sind aber nicht alle Web-Anwendungen ausschließlich mit solch einem Toolkit entwickelt und nicht für alle Toolkits gibt es vordefinierte Resolver in QF-Test. Sie können durchaus auch auf selbstentwickelte Komponenten stoßen.

Wenn Sie also mit dem Testen Ihrer eignen Web-Anwendung beginnen und auf Schwierigkeiten stoßen, dass Komponenten beim Abspielen von Aktionen nicht verlässlich wieder erkannt werden, dann kann es Zeit sein, über einen Resolver nachzudenken. Sollten Sie für die Resolvererstellung Hilfe benötigen, kontaktieren Sie bitte unser Support-Team für Unterstützung.

Es gibt ein Kapitel in der Technischen Referenz des Handbuchs, das tiefergehende Informationen zu CustomWebResolvern enthält und auch wie man bei der Implementierung eines solchen vorgeht, wenn Sie dies probieren möchten.

13.3 Der Bereich Fenster und Komponenten

Zum Thema "Komponenten" gibt es mehrere 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:

Ein einfacher Fall wird im Video

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

erläutert.

Einen komplexeren Fall gibt es in

'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

.

QF-Test speichert die Informationen, wie es die im GUI des SUT angezeigten Komponenten wiederfindet, im Bereich Fenster und Komponenten ab. Dabei analysiert QF-Test bei der Aufnahme die erhaltenen GUI-Element-Informationen und speichert diejenigen, mit denen der Anwender interagiert hat, in den Details der Komponente Knoten ab.

In diesem Abschnitt wollen wir Ihnen eine Vorstellung davon vermitteln, welche Informationen in Komponente Knoten abgespeichert werden und wie diese von QF-Test für die Wiedererkennung verwendet werden. Es gibt zwei Fälle bei denen Sie sich mit Komponenten befassen müssen und wo ein Grundverständnis für die Komponentenerkennung hilfreich ist.

Der erste Fall betrifft Änderungen im GUI zwischen dem Zeitpunkt der Aufnahme und demjenigen, wenn Sie einen Test abspielen wollen, wie dies z.B. bei einer neuen Version der Anwendung passieren kann. QF-Test verfügt zwar über eine Reihe von Algorithmen zur Komponentenerkennung, so dass diese im allgemeinen stabil ist. Wenn sich aber zu viele Merkmale des GUI-Elements verändert haben, müssen Sie die Änderungen in der Testsuite nachziehen und entweder die Details des vorhandenen Komponente Knoten anpassen oder das GUI-Element neu aufnehmen. Detaillierte Informationen zur Vorgehensweise finden Sie im Handbuch, Kapitel Lösung von Problemen bei der Wiedererkennung. Dort gibt es auch Links auf Videos mit entsprechenden Beispielen.

Im zweiten Fall geht es um die Verbesserung der Komponentenerkennung. Wie Sie bereits aus Kapitel Web-Komponentenerkennung(148) wissen, werden bei Web-Anwendungen die funktionalen Komponenten wie Checkbox oder Tabelle, sogar Eingabefelder, in mannigfaltiger Art und Weise aus den verfügbaren HTML Basiskomponenten erstellt. So kommt es immer wieder vor, dass QF-Test bei einem GUI-Element nicht erkennen kann, welche Funktionalität es aus Anwendersicht erfüllen soll. Für solche Fälle stehen sogenannte "Resolver" zur Verfügung. Wie diese eingesetzt werden, ist im technischen Teil des Handbuchs im Kapitel 'CustomWebResolver' erläutert.

Wir wollen uns nun anhand einer 'TextField' Komponente die von QF-Test analysierten und abgespeicherten Merkmale ansehen.

Aktion

- **Starten Sie das CarConfig Demo**, falls dies nicht bereits läuft. Führen Sie dazu den Vorbereitung Knoten in der Testsuite aus.

- **Öffnen Sie die Prozedur 'Endpreis prüfen'.**
- **Öffnen Sie das Kontextmenü des 'Check Text'-Knotens.**
- **Springen Sie zum Knoten der Komponente** über den Menüpunkt Komponente finden im Popup-Menü oder über das Tastaturkürzel Strg-W.

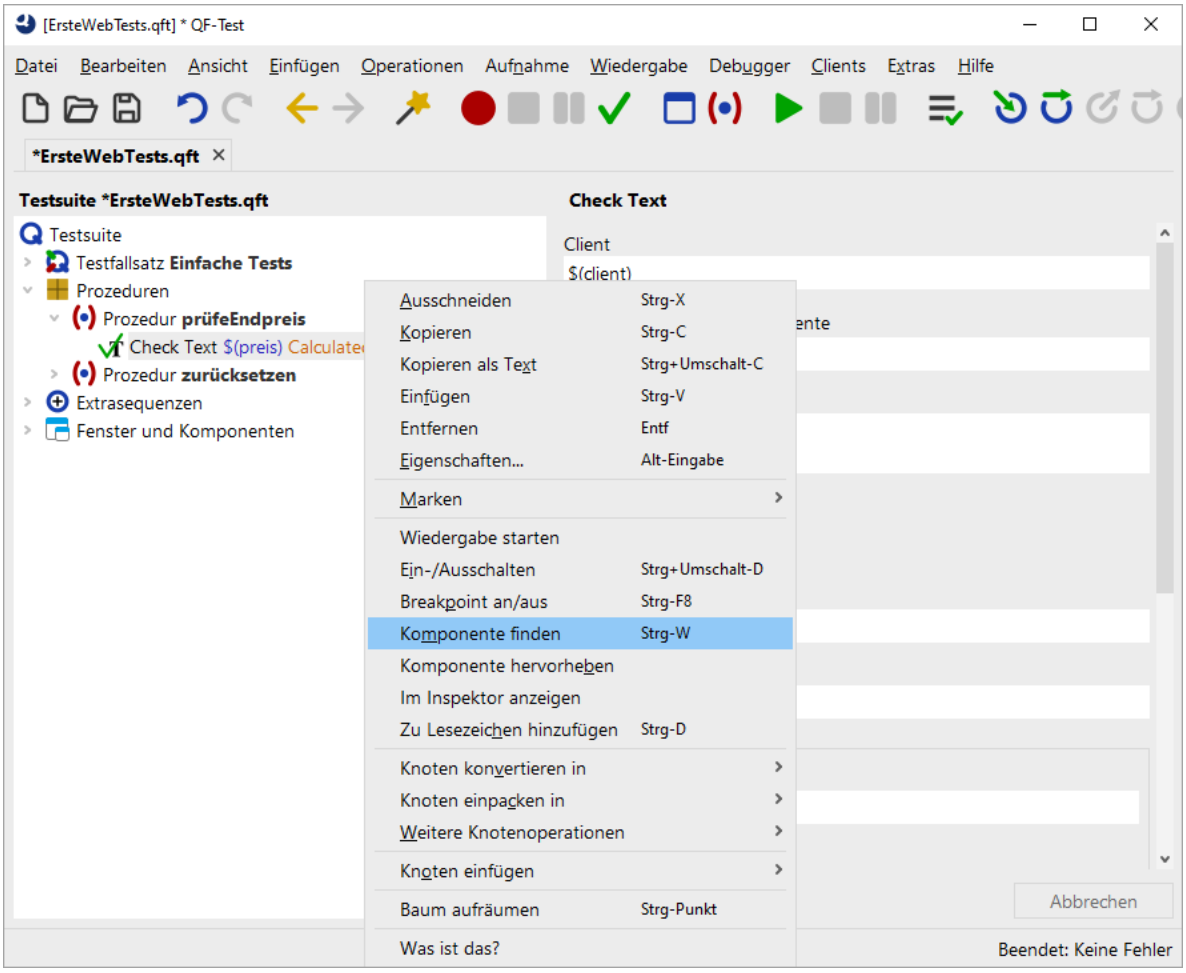

Abbildung 13.3: Komponente finden

Dadurch gelangen Sie direkt zum Knoten 'TextField CalculatedPrice' im Bereich Fenster und Komponenten.

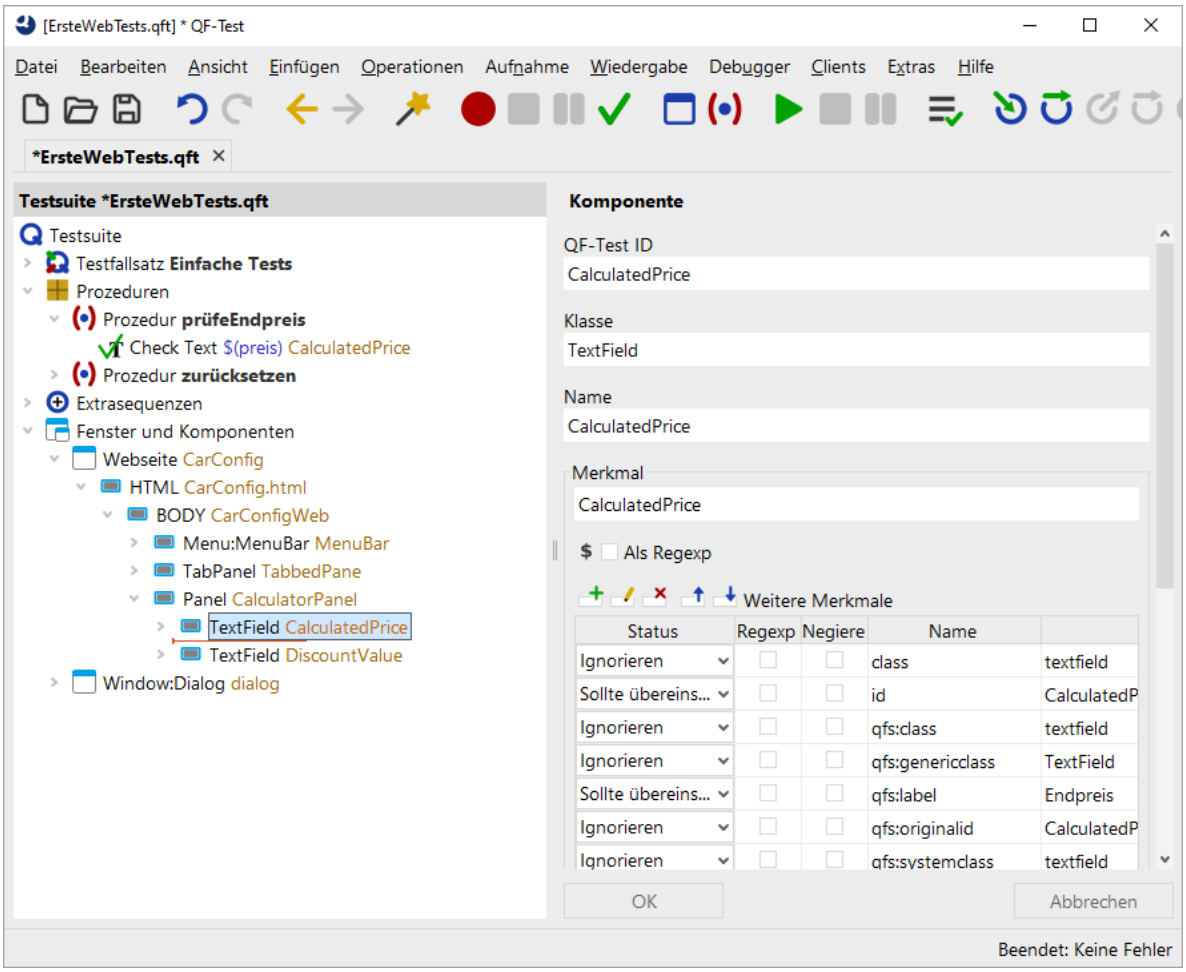

Abbildung 13.4: Komponentenbaum

Dieser Komponentenbaum hat nur wenige Ebenen. Dies ist eine Wirkung der in Abschnitt 13.2[148] beschriebenen verbesserten Web-Komponentenerkennung. Wenn Sie sich den Komponentenbaum in der in Kapitel 11[123] erstellten eigenen Testsuite ansehen, werden Sie feststellen, dass dort eine große Anzahl an DIV-Ebenen aufgenommen wird. Diese sind auf der HTML-Seite definiert. Für die Komponentenerkennung sind sie jedoch nicht relevant bzw. sogar störend. Daher wird ihre Aufnahme mittels des CustomWebResolvers in der Demo-Testsuite unterdrückt.

Für den Komponente Knoten mit der QF-Test ID `CalculatedPrice` sehen die Details wie folgt aus:

Komponente

QF-Test ID
CalculatedPrice

Klasse
TextField

Name
CalculatedPrice

Merkmal
CalculatedPrice

$ Als Regexp

Weitere Merkmale

Status	Regexp	Negiere	Name	Wert
Ignorieren			class	textfield
Sollte übereins...			id	CalculatedPrice
Ignorieren			qfs:class	textfield
Ignorieren			qfs:genericclass	TextField
Sollte übereins...			qfs:label	Endpreis
Ignorieren			qfs:originalid	CalculatedPrice
Ignorieren			qfs:systemclass	textfield
Ignorieren			tag	TD

Struktur
Index: 5
Insgesamt: 6

Geometrie
X: 174
Y: 81
Breite: 62
Höhe: 29

Bemerkung

Abbildung 13.5: Details eines Komponente Knoten

Wie werden die Attribute des Komponente Knotens zur Wiedererkennung der GUI-Elemente eingesetzt?

Das erste Attribut ist **QF-Test ID**, das in den Testfällen und Prozeduren zur Ansprache der Komponente verwendet wird. Alle anderen Attribute beziehen sich auf Eigenschaften des GUI-Elements.

Das folgende Attribut ist die **Klasse**. In unserem Fall `TextField`. Für die Komponentenerkennung ist die Klasse ein eindeutiges Merkmal. Die angezeigte Klasse ist eine

von QF-Test verallgemeinerte Klasse. Durch diese generische Klasse werden die Tests unabhängig von der konkreten Implementierung und können leicht portiert werden. Die speziellen Werte werden in der Tabelle ”Weitere Merkmale” abgespeichert. Sie spielen standardmäßig für die Erkennung keine Rolle, können aber für Sonderfälle genutzt werden.

Weitere Beispiele für generische Klassen sind ”Panel”, ”Dialog” und ”Button”.

Dass Attribut **”Name”** enthält den Namen oder die Id, die dem GUI-Element seitens des Programmierers gegeben wurde. Falls ein Name vorhanden ist, ist dies für QF-Test zusammen mit der Klasse zur Komponentenerkennung ausreichend. Die übrigen Attribute bleiben dann unberücksichtigt.

Wenn weder ein Name noch eine Id durch den Entwickler gesetzt wurde und das Attribut ”Name” daher keinen Wert enthält, verwendet QF-Test andere Kriterien für die Erkennung wie z.B. einen bestimmten Text, der zur Komponente gehört, Index und Geometrie.

Ein zur Komponente gehörendes Merkmal wäre bei einer Schaltfläche zum Beispiel der darauf angezeigte Text. QF-Test speichert Texte, die direkt zur Komponente gehören im Attribut **”Merkmal”** ab. Wenn das Web-Element keinen Text enthält, jedoch eine Id vorhanden ist, so wird diese im Attribut ”Merkmal” abgespeichert. Wenn keines von beiden vorhanden ist, dann ein Text in der Nähe der Komponente, den QF-Test als mögliche Beschriftung identifiziert, ebenfalls im Merkmal-Attribut abgespeichert werden. Auf jeden Fall wird dieser Text in der Tabelle **”Weitere Merkmale”** unter dem Namen `qfs:label` gespeichert.

Die **Struktur** Informationen beziehen sich auf alle GUI-Elemente der jeweiligen Klasse. Die Gesamtzahl der GUI-Elemente der Klasse wird im Attribut ”Insgesamt”, der Index der Komponente selbst im Attribut ”Index” abgespeichert.

Am Schluss befinden sich die Werte für die **Geometrie**. Diese erhalten im Wiedererkennungsalgorithmus die geringste Gewichtung. In seltenen Fällen kann es jedoch vorkommen, dass sie die einzigen Kennzeichen sind, die zur Identifizierung des GUI-Elements zur Verfügung stehen.

Wenn Sie an weiteren Details zum genauen Ablauf bei der Wiedererkennung interessiert sind, können Sie diese im Kapitel Komponentenerkennung der technischen Referenz im Handbuch nachlesen.

Um ein Gefühl für die Komponentenerkennung zu erhalten, können Sie ein bisschen mit den Attributwerten herumspielen, bis QF-Test das GUI-Element nicht mehr findet oder sogar eine falsche Komponente auswählt. Sie werden feststellen, dass die Änderungen beträchtlich sein müssen, bevor QF-Test ein falsches GUI-Element identifiziert. Das heißt, dass die Komponentenerkennung von QF-Test sehr robust ist und sich bei neuen Versionen einer Anwendung ein erheblicher Anteil der Attribute eines GUI-Elements verändern kann, bevor die Komponentenerkennung fehlschlägt - selbst wenn das GUI-Element keinen Namen oder keine Id hat.

Beim Klick auf einen Komponente Knoten markiert QF-Test das erkannte GUI-Element mit einem dunkelblauen Rand.

Aktion

- **Löschen Sie den Wert `CalculatedPrice` aus dem Name-Attribut**, da QF-Test sonst die nachfolgenden Attribute nicht berücksichtigt.
- Bitte ändern Sie in der Tabelle Extra features **den Status des weiteren Merkmals mit dem Namen `id` von `Sollte übereinstimmen` auf `Ignorieren`** .
- **Ändern Sie das Merkmal-Attribut** von `CalculatedPrice` auf `xxx`.
- **Klicken Sie auf den "TextField" Knoten** um zu sehen, ob QF-Test immer noch das Endpreis-Feld markiert.
- **Setzen Sie das Merkmal-Attribut** zurück auf `CalculatedPrice`, entweder über den entsprechenden Button in der Werkzeugleiste oder über das Tastaturkürzel Ctrl-Z.
- **Ändern Sie den Wert von `qfs:label` in der Weitere-Merkmale-Tabelle** von `Endpreis` auf `Rabatt`.
- **Klicken Sie auf den "TextField" Knoten** um zu sehen, ob QF-Test immer noch das Endpreis-Feld markiert.
- **Setzen Sie den Wert von `qfs:label` in der Weitere-Merkmale-Tabelle** zurück auf `Endpreis`.
- **Setzen Sie alle Struktur- und Geometrie-Attribute** auf andere Werte und überprüfen Sie, ob QF-Test immer noch das Endpreisfeld im GUI markiert.
- **Löschen Sie das Merkmal-Attribut** and Ändern Sie den Wert von **`qfs:label`** in der Weitere-Merkmale-Tabelle **von `Endpreis` auf `Rabatt`**.

Nun markiert QF-Test das Rabattfeld.

Diese Übung ist nur eine kleine Spielerei mit der Komponentenerkennung. Konkrete Informationen, wie Sie bei Problemen mit der Komponentenerkennung umgehen, finden Sie in den oben genannten und weiteren Kapiteln im Handbuch.

13.4 SmartIDs - direkte Komponentenadressierung

Seit QF-Test 7.0 bieten SmartIDs offiziell die Möglichkeit, Komponenten ohne Aufnahme eines Komponente Knoten zu referenzieren.

Für gewisse Anwendungen kann dies die Verwaltung und Pflege der Komponenteninformationen stark vereinfachen.

Auch auf die Editier- und Lesbarkeit der Tests können SmartIDs positiven Einfluss haben.

Nicht zuletzt bietet dies die Möglichkeit, Tests auch ohne Aufnahmefunktion zu erstellen, zum Beispiel, wenn eine Komponente oder die gesamte Anwendung noch gar nicht vorhanden ist, man den Test aber bereits implementieren möchte ("Test first" Ansatz).

Als Wiedererkennungskriterien stehen die Komponentenklasse, deren Name oder Beschriftung und der Index zur Verfügung. Die Werte sind die gleichen wie beim aufgenommenen Komponente Knoten. Zudem können auch Komponentenhierarchien abgebildet werden.

Eine SmartID wird an Stelle der QF-Test ID der Komponente verwendet. Ihr Kennzeichen ist die Raute `#` als erstes Zeichen. Anschließend kann der Name oder die Beschriftung der Komponente geschrieben werden, zum Beispiel:

- `#btnOK`, mit "btnOK" als Namen der Komponente oder
- `#Vorname`, mit "Vorname" als Beschriftung der Komponente.

Der Nachteil dieser einfachen Form der SmartID kann die Performanz bei der Wiedergabe sein, da QF-Test alle Komponenten nach diesen Kriterien durchsuchen muss. Aus diesem Grund ist es sinnvoll, die Klasse der Komponente mit anzugeben. Obige SmartIDs könnten dann so aussehen:

- `#Button:btnOK`
- `#TextField:Vorname`

Aktuell ist die Aufnahme von SmartIDs nicht als Standard aktiviert, kann aber direkt über das Menü eingeschaltet werden.

Nehmen Sie nun SmartIDs auf, indem Sie

Aktion

- im Menü **Aufnahme** die Einstellung **Aufnahme von SmartIDs** anhaken.
- **Aktivieren Sie den Aufnahmemodus** über "Aufnahme starten" ● .
- **Klicken Sie auf ein Textfeld**, z.B. das Eingabefeld für den Rabatt,
- **Klicken Sie auf eine Tabellenzelle**, z.B. das erste Modell.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .

Die aufgenommenen Mausklicks finden Sie im Bereich Extrasequenzen.

Die SmartID für das Eingabefeld lautet `#TextField:name=DiscountValue`, da der Name bevorzugt für die SmartID verwendet wird, wenn Name und Beschriftung vorhanden sind.

Der zweite Mausklick zeigt die Aufnahme der Tabelle als SmartID und daran anschließend einen Textindex für die Spalte, eingeleitet mit `@` und einen numerischen Index für die Zeile, eingeleitet mit `&`, wie Sie dies im vorletzten Abschnitt kennengelernt haben: `#Table:name=VehicleTable@Modell&0`.

Aktion

- Wenn Sie nicht weiter aktiv mit SmartIDs arbeiten möchten, können Sie diese in der Aufnahme-Option wieder deaktivieren.

Sie können aber auch gerne weiter mit SmartIDs arbeiten. Dabei werden Sie bei Aufnahmen auch weitere Ausdrücke in SmartIDs finden. Einige wollen wir hier zeigen und die Hintergründe dazu erläutern.

Zum Beispiel werden Sie bei einer Beschriftung, die mit einem Doppelpunkt endet, vor diesem Doppelpunkt einen Rückstrich sehen:

- `#TextField:Vorname\:`

Der Hintergrund ist, dass ein Doppelpunkt den davor stehenden Text als Komponentenklasse kennzeichnet. Daher müssen Doppelpunkte im Namen oder der Beschriftung mit einem Rückstrich geschützt werden. Dies gilt auch für die Sonderzeichen, die Indizes einleiten, also `@`, `&` und `%`.

Vor Beschriftungen werden Sie einen Kennzeichner, hier `left=` sehen:

- `#TextField:left=Vorname`

Hintergrund: Eine SmartID mit Angabe von Klasse und Namen der Komponente erreicht eine gleich gute Performanz bei der Wiedergabe wie die Verwendung von Komponente Knoten. Bei Beschriftungen sieht dies jedoch anders aus. Hier gibt es unterschiedlichste Möglichkeiten, was die beste Beschriftung für eine Komponente darstellt. QF-Test sucht unter den für eine Komponente in Frage kommenden Beschriftungen auf Basis der Komponentenklasse, Lage und Abstand der Beschriftungen die Beste aus. Für die Performanz bei der Wiedergabe ist es daher hilfreich, wenn direkt angegeben wird, nach welcher Beschriftung gesucht werden soll.

`left=` gibt hierbei an, dass die Beschriftung links der Komponente zu finden ist. Weitere Lagebezeichnungen in einer SmartID sind `right=`, `top=`, `topleft=` und `bottom=`. Wenn die Beschriftung der Text der Komponente ist, lautet der Kennzeichner `text=`, für den Tooltip `tooltip=`.

Wenn Komponenten mit dem gleichen Namen oder der gleichen Beschriftung mehrmals auf einer Anzeige vorhanden sind, können Sie auf SmartIDs stoßen, bei denen zwei über das Verbindungszeichen `@` zu einer vereinigt werden:

- `#TitledPanel:Kundenadresse@#TextField:left=Vorname`

- `#TitledPanel:title=Rechnungsadresse@#TextField:left=Vorname`

Im Beispiel gibt es die Beschriftung ”Vorname” sowohl in der Kachel ”Kundenadresse” als auch in der Kachel ”Rechnungsadresse”. Über die geschachtelte SmartID kann die Eindeutigkeit hergestellt werden.

Das zweite Beispiel ist etwas performanter wegen `title=` bei der SmartID bei der Kachel. Dafür ist die Lesbarkeit beim ersten etwas besser. Es hängt von der Anwendung ab, wie sehr auf Performanz geachtet werden muss. Bei Web-Anwendungen mit sehr vielen geladenen GUI-Elementen ist dies typischerweise relevant. Bei Java-Anwendungen kann man häufig der Lesbarkeit den Vorrang geben.

Bei langen Beschriftungen kann ein regulärer Ausdruck zur Verkürzung verwendet werden:

- `#%Dialog:Information.*@#Button:OK`

Das Prozentzeichen direkt nach der Raute gibt an, dass entweder der Name oder die Beschriftung als regulärer Ausdruck zu interpretieren ist. Im Beispiel wird der Titel verkürzt. Außerdem gilt die SmartID für jeden OK-Button, der in einem Dialog liegt, dessen Titel mit ”Information” beginnt. Eine genauere Beschreibung regulärer Ausdrücke finden Sie im Handbuch - reguläre Ausdrücke. Weitere Informationen zu Komponenten und SmartIDs finden Sie im Handbuch - Komponenten.

Kapitel 14

Benutzen des Debuggers (Web)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Benutzen des Debuggers'
https://www.qftest.com/de/yt/tutorial-5.html

In diesem Kapitel lernen Sie, wie der in QF-Test integrierte intuitive Debugger benutzt wird. Diejenigen unter Ihnen, die bereits Erfahrungen mit anderen IDEs, wie z.B. Eclipse haben, werden hier Ähnlichkeiten in Funktion und Nutzen des Debuggers feststellen.

Wir werden uns mit den folgenden Debugger-Funktionen beschäftigen:

- Setzen eines Breakpoints[161] mittels Strg-F8 (⇧-⌘-B auf macOS).
- Testausführung pausieren[170] mittels Pausetaste ⏸ oder der Tastenkombination Alt-F12.
- Schrittweise Ausführung[162] mittels "Einzelschritt ausführen" , "Gesamten Knoten ausführen" und "Bis Knotenende ausführen" .
- Knoten überspringen[164] mittels "Knoten überspringen" und "Aus Knoten herausspringen" .
- Debug-Modus bei Fehler oder Exception aktivieren[166].
- Fehlerbehebung aus dem Protokoll heraus[168].
- Den aktuellen Fehler im Protokoll direkt anspringen über Strg-J. (Ins Protokoll springen in Kapitel Abschnitt 14.5[168]).

Hinweis

Anstatt über die Schaltflächen können die Befehle auch über die Menüzeile oder Tastaturkürzel abgesetzt werden. Die Kürzel stehen neben den Optionen in den QF-Test Menüs, sofern vorhanden. Eine vollständige Übersicht der von QF-Test verwendeten

Tastaturkürzel finden Sie im Anhang Tastaturkürzel im Handbuch. Dort findet sich auch ein kleiner Helfer für die Funktionstastenbelegung von QF-Test zum Befestigen an der Tastatur.

Es gibt noch einige weitere Debugger-Funktionen wie

- "Aktuellen Knoten finden" (Aktuellen Knoten finden in Abschnitt 15.3[180]),
- "Ausführung hier fortsetzen" über das Popup-Menü des entsprechenden Knotens (Abbildung 15.9[183]),
- die "Exception erneut werfen" ,
- die Tabelle der Variablendefinitionen (Abschnitt 15.3[180]),

auf die wir in späteren Kapiteln eingehen werden.

14.1 Setzen eines Breakpoints

Zunächst einmal soll der Debugger aktiviert werden. Dies kann auf mehrere Arten erfolgen, zum Beispiel durch das Setzen eines Haltepunktes (Breakpoint) vor Start des Testlaufs. Der Zweck eines Haltepunktes ist es, den Testlauf an einer Stelle, die man näher untersuchen möchte, zu unterbrechen. Sobald QF-Test auf den Breakpoint trifft, wird die Testausführung pausiert und der Debugger-Modus aktiviert. Der Pauseknopf ist nun gedrückt.

Aktion

- Selektieren Sie einen Knoten und **drücken Sie Strg-F8 (⇧-⌘-B auf macOS)**. Der Haltepunkt wird durch das Symbol B kenntlich gemacht.

Abbildung 14.1: Breakpoint setzen

Aktion

- Selektieren Sie den Testsuite Knoten und starten Sie den Testlauf über die Taste Enter.

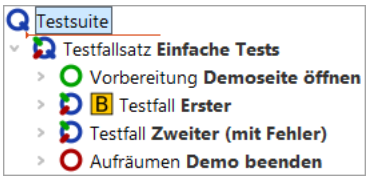

Abbildung 14.2: Testlauf starten

Aktion

- Löschen Sie den Breakpoint wieder, indem Sie nochmals **Strg-F8** (**⇧-⌘-B** **auf macOS) drücken**.

Abbildung 14.3: Breakpoint löschen

Man kann einen Breakpoint nicht nur über das Tastaturkürzel Strg-F8 sondern auch über den Menüpunkt Debugger→Breakpoint an/aus oder alternativ durch Rechtsklick auf den Knoten und Auswahl von Breakpoint an/aus im Popup-Menü setzen bzw. löschen. Für die weiteren Debugger-Funktionen werden wir hauptsächlich die jeweiligen Schaltflächen nennen, aber auch hier stehen die anderen Varianten zur Verfügung.

Wieder sehen Sie den kleinen Pfeil, der diesmal anzeigt, welcher Knoten als nächster ausgeführt wird. Dieser Knoten wird **aktueller Knoten** genannt. Bei der Aktivierung des Debug-Modus navigiert QF-Test zum aktuellen Knoten, falls dieser nicht bereits sichtbar ist, und selektiert diesen, d.h. die Zeile wird blau hinterlegt.

Das Kommando Debugger→Alle Breakpoints löschen ist ebenfalls nützlich, wenn alle Haltepunkte in allen geöffneten Testsuiten gelöscht werden sollen.

Es gibt keine Beschränkung für die Anzahl an Breakpoints, die Sie in Ihrer Testsuite setzen können. Haltepunkte werden beim Schließen der Testsuite nicht mit abgespeichert.

14.2 Schrittweise Ausführung

Nun wollen wir die Testfälle schrittweise ausführen.

Aktion

- Bitte experimentieren Sie ein wenig mit **"Einzelschritt ausführen"** , **"Gesamten Knoten ausführen"** und **"Bis Knotenende ausführen"** .

Wie Sie sicher festgestellt haben werden, öffnet **"Einzelschritt ausführen"** einen Knoten mit Kindern und macht den ersten Kindknoten zum aktiven Knoten. Dies ist wie immer an der Pfeilmarkierung des Knotens erkennbar.

Wenn Sie an dem Punkt weitergemacht haben, an dem die Ausführung der Testsuite im letzten Abschnitt pausiert war, d.h. vom Knoten "Testfall: Erster" aus, so würde nun der Testfall geöffnet werden:

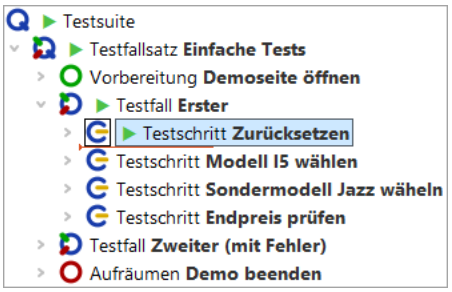

Abbildung 14.4: Einzelschritt ausführen

Im Falle eines Blattknotens, d.h. eines Knotens, der keine Kinder hat, ist die Wirkung die gleiche wie die der folgenden Funktion.

Mittels der Schaltfläche **"Gesamten Knoten ausführen"** wird ein Knoten inklusive aller Kindknoten ausgeführt. Der als nächstes auszuführende Knoten auf der gleichen Ebene wird dann der aktive und erhält den Pfeil.

Abbildung 14.5: Gesamten Knoten ausführen

"Bis Knotenende ausführen" führt die verbleibenden Geschwisterknoten aus und stoppt beim nächsten auszuführenden Knoten der übergeordneten Hierarchieebene.

Abbildung 14.6: Bis Knotenende ausführen

Im Beispiel ist dies der Aufräumen Knoten. Wie bereits im ersten Kapitel Ein vollständiger Testlauf[119] erläutert, haben Vorbereitung und Aufräumen Knoten die besondere Eigenschaft, dass sie vor und nach **jedem Testfall** ausgeführt werden, um so einen definierten Anfangszustand für jeden Testfall herzustellen.

Hinweis Dieses Verhalten tritt nur auf, wenn Sie die komplette Testsuite oder den Testfallsatz gestartet hatten und sich im Debug-Modus befinden. Wenn keine Testausführung aktiv war und Sie nur den Testfall selektiert hatten, so bewirkt die Funktion "Gesamten Knoten ausführen", dass der Testfall ausgeführt wird und dann der nächste Testfall selektiert wird.

Aktion

- Führen Sie die Aufräumen und Vorbereitung Knoten aus, indem Sie mit Hilfe der Schaltfläche die **gesamten Knoten ausführen** und dann über **den zweiten Testfall öffnen**. Dies ist eine Vorbereitung für das nächste Kapitel, in dem es um das Überspringen von Knoten geht.

Hinweis Bitte beachten Sie bei der interaktiven Testerstellung bei Menüs und Comboboxen, dass diese häufig zuklappen, wenn die Applikation den Fokus verliert, was beim Wechsel in den Debug-Modus der Fall ist. In diesem Fall empfiehlt es sich, den Knoten, der ein Menü oder eine Combobox öffnet, und den Knoten, der die Auswahlaktion durchführt, gemeinsam auszuführen, also nicht zwischendurch in den Debug-Modus zu gehen. Dies kann man zum Beispiel dadurch erreichen, dass man nach dem Knoten, der die Auswahlaktion durchführt, einen Haltepunkt B setzt und bei Erreichen des Knotens, der das Menü oder die Combobox öffnet, die Testausführung durch Lösen der Pausetaste freigibt.

14.3 Knoten überspringen

Die "Überspringen" Funktionen erweitern die Fähigkeiten des Debuggers von QF-Test in einer Weise, die über den Funktionsumfang von Standardprogrammierumgebungen

hinausgeht. Wie der Name andeutet, erlauben die ”Überspringen” Operationen einen oder mehrere Knoten während des Testlaufs auszulassen, d.h. weiter zu springen ohne diese auszuführen. Dies kann aus verschiedensten Gründen sinnvoll sein. Sei es um schnell an eine gewisse Position in Ihrem Testablauf zu gelangen oder um einen aktuell zu einem Fehler führenden Knoten zu überspringen.

Am Schluss des letzten Abschnitts haben wir den ersten Testschritt im zweiten Testfall zum aktiven Knoten gemacht. Dies ist er Ausgangspunkt für unsere nächste Aktion:

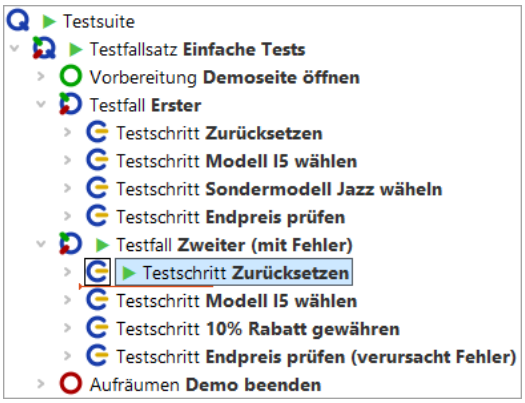

Abbildung 14.7: Testausführung am ersten Knoten des zweiten Testfalls pausiert

- Drücken Sie nun die Schaltfläche **”Knoten überspringen”** . QF-Test springt einfach über den aktiven Knoten ohne ihn oder seine Kindknoten auszuführen. Anschließend pausiert QF-Test beim nächsten auszuführenden Knoten auf der gleichen Ebene.

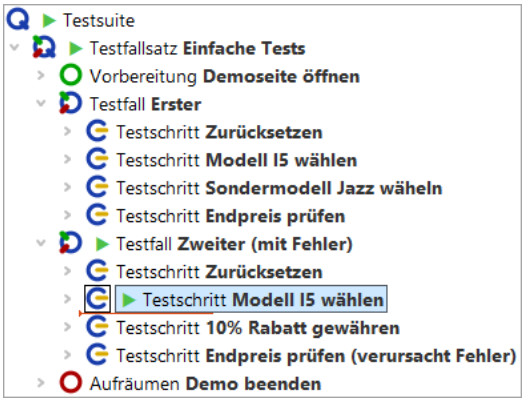

Abbildung 14.8: ”Knoten überspringen”

Aktion

- Und zuletzt die Schaltfläche **”Aus Knoten herausspringen”** . Sie sehen so-

fort, dass QF-Test aus dem Knoten, in dem Sie sich befinden, herausspringt ohne weitere Kindknoten auszuführen.

Abbildung 14.9: ”Aus Knoten herausspringen”

Hinweis Noch eine Bemerkung zu ”Knoten überspringen” und ”Aus Knoten herausspringen”: Benutzen Sie diese mit Vorsicht! Aus einer Sequenz herauszuspringen, bevor diese zu Ende gelaufen ist, kann dazu führen, dass Ihr SUT in einem Status belassen wird, mit dem andere Sequenzen oder Tests in der Suite nicht aufsetzen können.

14.4 Debug-Modus bei Fehler oder Exception aktivieren

Beim Debuggen eines Tests ist es oft hilfreich, wenn die Testausführung genau dann stoppt und in den Debug-Modus gewechselt wird, wenn ein Fehler, eine Exception oder manchmal auch nur eine Warnung auftritt.

Mittels dieser Technik werden wir in diesem Abschnitt und dem nächsten den zweiten Testfall debuggen.

Aktion

- Bitte **öffnen Sie das Debugger-Menü** und ändern Sie die Standardeinstellungen wie folgt:
- **Klicken Sie** auf den Menüpunkt **Debugger→Debugger aktivieren** um ihn zu aktivieren.
- **Klicken Sie** auf den Untermenüpunkt **Debugger→Optionen→Unterbrechen bei Fehler** um auch diese Funktion zu aktivieren.

Wenn Sie nun das Debugger-Menü und das Optionen-Untermenü wieder öffnen sollte es wie folgt aussehen:

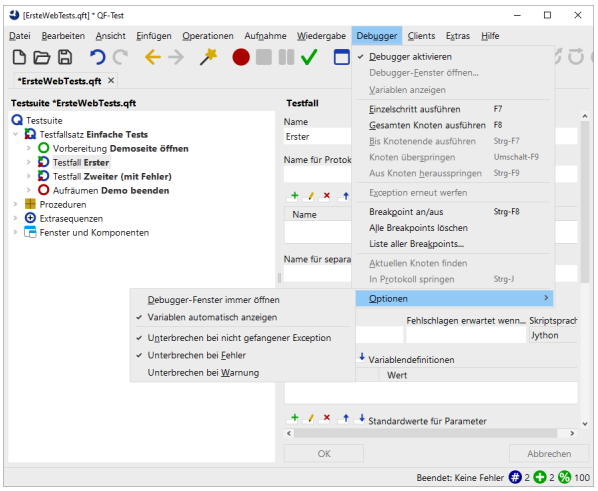

Abbildung 14.10: Debugger-Optionen: Test bei Fehler anhalten

Wir müssen die Debugger-Optionen ändern, da sonst der Test einfach durchlaufen würde, analog zu den vorherigen Beispielen aus Kapitel eins und zwei.

Aktion

- **Selektieren Sie den Testsuite Knoten** und starten Sie anschließend den Test mittels "Wiedergabe starten" ▶ .

QF-Test hält bei dem fehlerhaften Knoten an und wechselt in den Debug-Modus:

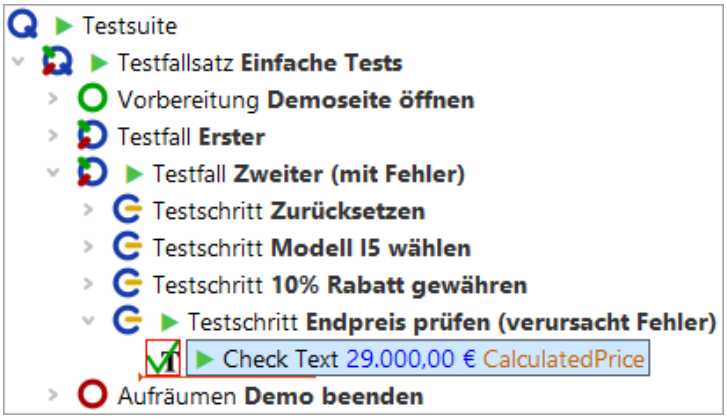

Abbildung 14.11: QF-Test pausiert bei Fehler

Der Knoten, der den Fehler verursacht hat, wird durch ein rotes Quadrat markiert. Außerdem erscheint ein Fehlerdialog, der uns Näheres zur Fehlerursache mitteilt. Über

diesen wechseln wir in das Protokoll, das wie so oft der Schlüssel zur Fehlerbehebung ist.

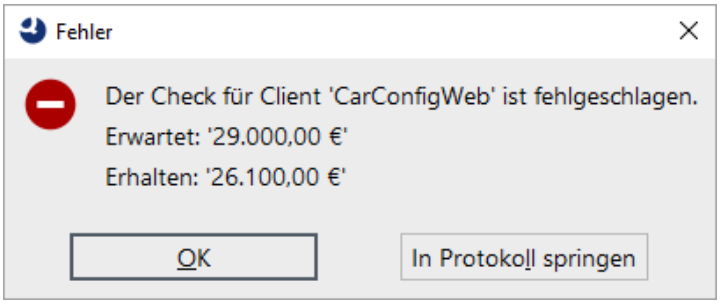

Abbildung 14.12: Fehlermeldung

Aktion

- **Klicken Sie** auf die Schaltfläche **In Protokoll springen** der Fehlermeldung.

14.5 Fehlerbehebung aus dem Protokoll heraus

Über die Schaltfläche **In Protokoll springen** (siehe Fehlermeldung in Abbildung Abbildung 14.12[168]) wird das Protokoll direkt bei dem entsprechenden Knoten geöffnet.

Neben der eigentlichen Fehlermeldung wurden etliche weitere Informationen zur Testumgebung zum Zeitpunkt des Fehlers protokolliert. Neben Bildschirmabbildern zum Fehlerzeitpunkt enthält das Protokoll unter dem Knoten, der den Fehler verursachte, eine Liste der gebundenen Variablen (Stacktrace). Auf die Nützlichkeit des Stacktrace werden wir zu einem späteren Zeitpunkt eingehen (Die Variablendefinitionen-Tabelle[176]).

Beim vorliegenden Fehler wird der falsche Wert im Check Text Knoten der Testsuite erwartet. Zur Fehlerbehebung muss dieser durch den tatsächlich angezeigten ersetzt werden. Dies geht bei einem Check mit festem Wert, um den es sich hier handelt, am einfachsten, indem Sie

Aktion

- auf den rot umrandeten Fehler-Knoten **"Fehlgeschlagen: Check Text: default ..." rechtsklicken** und
- im Kontextmenü **Check-Knoten mit erhaltenen Daten aktualisieren** **auswählen**.

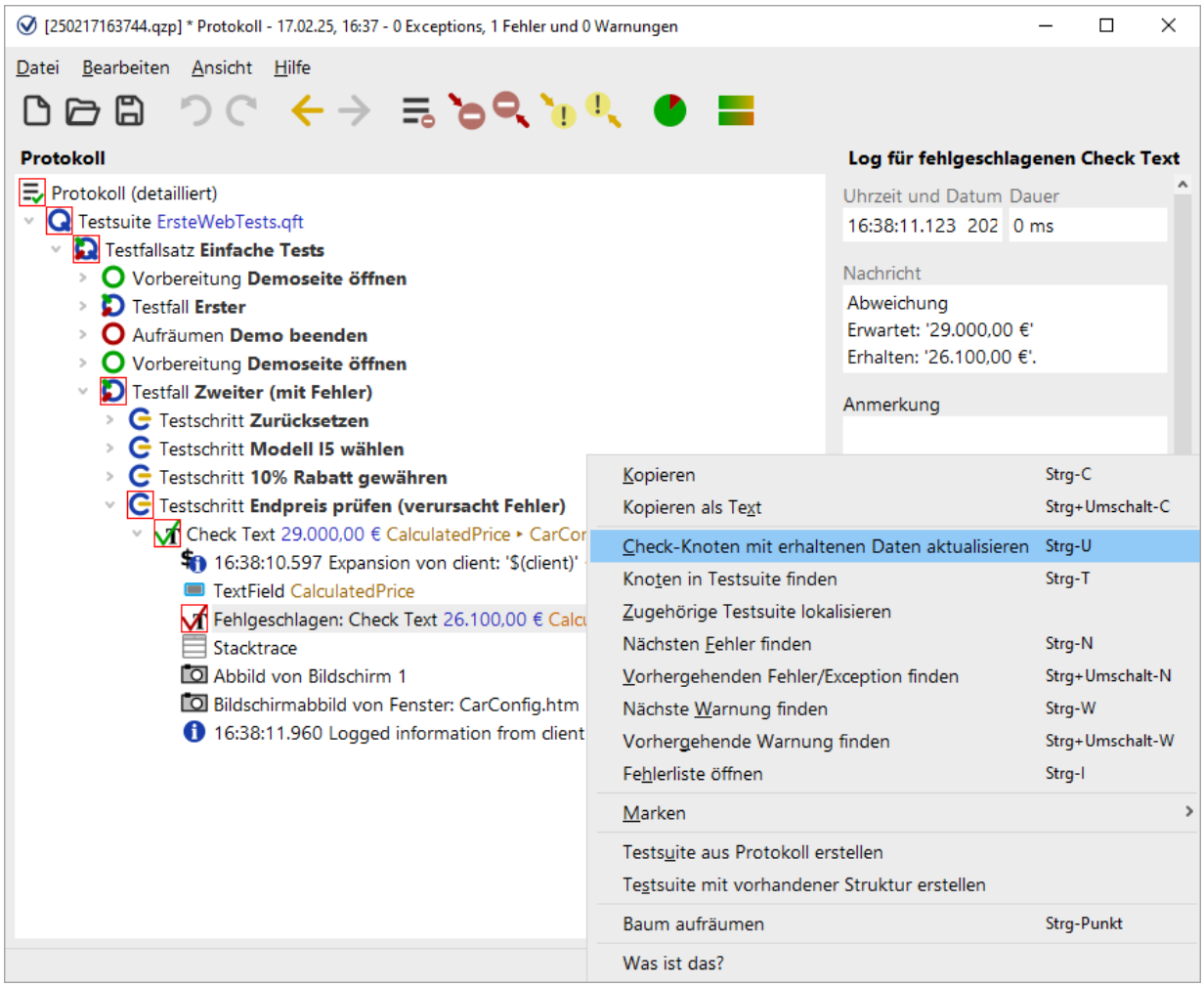

Abbildung 14.13: Check-Knoten mit erhaltenen Daten aktualisieren

QF-Test navigiert zum betroffenen Check Text Knoten in der Testsuite und aktualisiert den Wert des Attributs Text anhand der aus dem SUT ausgelesenen Daten.

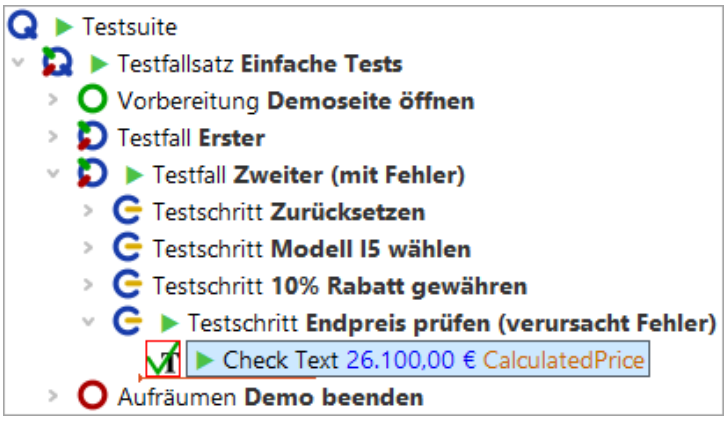

Abbildung 14.14: Korrigierter Check-Knoten

Der Knoten enthält nun zwar den korrekten Wert, ist aber immer noch rot umrandet, da er noch nicht wieder ausgeführt wurde. Dies wollen wir nun tun.

Aktion

- führen Sie den Test also fort, indem Sie auf den Pauseknopf ⏸ drücken und so **die Pause lösen**.

QF-Test führt den Rest der Testsuite aus. In unserem Fall sind das der Check Text und der Aufräumen Knoten. Anschließend informiert Sie QF-Test, dass ein Fehler aufgetreten ist. Diesen haben wir allerdings bereits während des Testlaufs behoben.

Ins Protokoll springen: Falls Sie das Protokoll an der Stelle, an der sich die Testausführung gerade befindet, öffnen wollen, brauchen Sie aus dem Debugging Modus heraus nur den Menüpunkt **Debugger→Jump to Run log** anzuklicken oder das Tastaturkürzel Strg-J zu drücken. Wenn Sie einfach nur das Protokoll öffnen wollen, ohne an die aktuelle Stelle zu springen, steht Ihnen Strg-L zur Verfügung, was auch nach Ende des Testlaufs weiterhin funktioniert.

14.6 Testausführung pausieren

Wenn ein Test gerade ausgeführt wird und Sie den Debug-Modus aktivieren wollen, so können Sie entweder schnell einen Breakpoint auf einen noch nicht ausgeführten Knoten setzen oder Sie drücken einfach die Schaltfläche "Pause" ⏸ um den Debug-Modus sofort zu aktivieren.

Um die Ausführung fortzusetzen lösen Sie den Pauseknopf ⏸ , völlig unabhängig von der Art und Weise wie der Debug-Modus aktiviert wurde.

Wir wollen die vorhandenen Testfälle im Verlauf dieses Tutorials weiter verwenden. Allerdings wurde der Fehler in dem zweiten Testfall jetzt behoben. Insofern macht es Sinn "(mit Fehler)" aus dem Namen des zweiten Testfalls zu löschen, genauso wie "(verursacht Fehler)" aus dem Namen des Testschritts.

Es gibt manchmal Situationen, in denen das SUT ständig den Fokus für sich beansprucht. Dann kann es schwierig sein, das QF-Test Fenster lange genug im Vordergrund zu halten, um die Pausetaste drücken zu können. In einem solchen Fall steht Ihnen die "Keine Panik"-Taste Alt-F12 zur Verfügung. Sie unterbricht alle laufenden Tests sofort. Zur Weiterführung des Tests können Sie diese Tastenkombination erneut drücken.

Kapitel 15

Variablen und Prozedurparameter (Web)

In diesem Kapitel lernen Sie, wie man eine Prozedur einsetzt um die gleichen Schritte auf unterschiedlichen Daten auszuführen. Außerdem sehen Sie, wie man Variablen einsetzt. Ebenso wird die Fehleranalyse in Bezug auf Variablen behandelt.

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Variablen und Prozedurparameter'
https://www.qftest.com/de/yt/tutorial-6.html

. Im Video wird eine Java-Applikation für die Erläuterungen verwendet. Bei Webapplikationen gibt es hinsichtlich der Variablenverwendung keine Unterschiede

15.1 Prozedur mit Variable

Sehen Sie sich den letzten Testschritt "Endpreis prüfen" in unseren beiden Testfällen an.

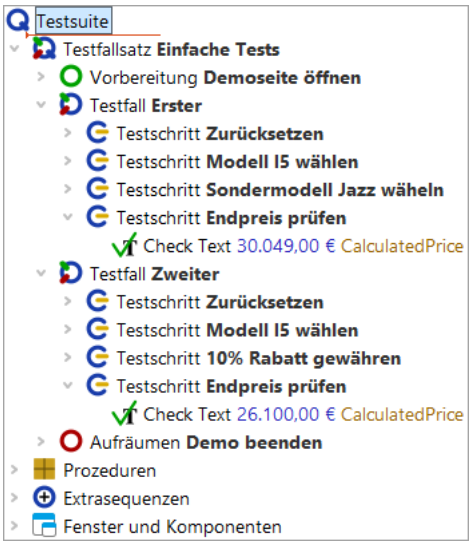

Abbildung 15.1: Zwei fast gleiche Testschritte

Es wird der gleiche Schritt ausgeführt, jedoch mit unterschiedlichen Daten. Auch wenn es sich nur um einen Schritt handelt, macht es Sinn eine Prozedur daraus zu machen. Vielleicht kommen wir später auf die Idee, die hartkodierten Werte `30.049,00 €` und `26.100,00 €` in ein anderes Format zu bringen, so dass der Check auf das Feld "Endpreis" auch für andere Währungen funktioniert. Diesen Algorithmus zweimal zu implementieren wäre auf jeden Fall nicht sinnvoll.

Aktion

- **Selektieren Sie den "Check text" Knoten im ersten Testfall.**
- Wählen Sie den Menüpunkt **Operationen→Knoten einpacken in→Sequenz** aus oder verwenden Sie das Tastaturkürzel Strg-Umschalt-S um ihn in eine Sequenz einzupacken.
- **Nennen Sie die Sequenz 'prüfeEndpreis'.** Dieser Name entspricht der Java-Konvention die Wörter zusammenzuschreiben. Andererseits erlaubt QF-Test auch Leerzeichen in Prozedurnamen, so dass Sie der Java-Konvention nicht zu folgen brauchen.
- Drücken Sie **Strg-Umschalt-P** um auf kürzestem Weg den Sequenzknoten in eine Prozedur zu konvertieren (wie aus dem letzten Kapitel bekannt). Wie Sie sehen, wurde die Sequenz durch einen Prozeduraufruf von "prüfeEndpreis" ersetzt.
- **Klicken Sie doppelt auf den Prozeduraufruf**, um zur Prozedur im Prozeduren Knoten zu springen.
- **Öffnen Sie** den neu erstellten Prozedurknoten um den Inhalt zu sehen.

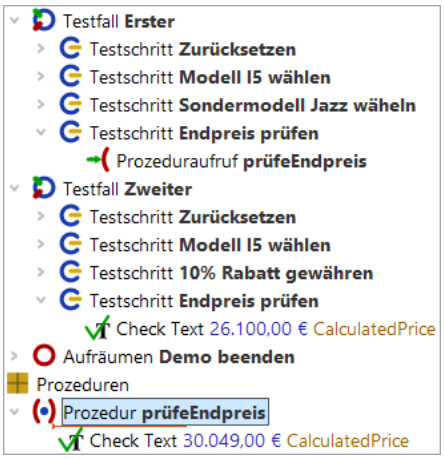

Abbildung 15.2: Prozedur mit hartkodiertem Wert

Wie erwartet befindet sich der "Check text" Knoten in der Prozedur. Er ist jedoch nur für einen einzigen Preis gültig, nämlich `30.049,00 €`. Da wir die gleiche Prozedur auch für den zweiten Testfall verwenden möchten, müssen wir den Preis durch eine Variable ersetzen. Der Wert dieser Variable sollte dann beim Prozeduraufruf mit übergeben werden.

Im nächsten Beispiel werden wir einen Parameter mit Standardwert im Prozedurknoten einfügen. Standardwerte werden häufig verwendet, wenn der entsprechende Parameter bei den meisten Prozeduraufrufen den Standardwert erhalten würde. Dann braucht man den Standardwert nicht jedes Mal zu spezifizieren, sondern kann auf den im Prozedurknoten definierten Wert zurückgreifen. Obwohl das auf den Preis als Parameter nicht zutrifft, können wir ihn gerade deshalb verwenden um zu zeigen, wie ein Standardwert funktioniert und wie man ihn bei Bedarf mit einem anderen Wert überschreiben kann.

Als erstes fügen wir also eine Variable mit Standardwert ein:

Aktion

- **Selektieren Sie die Prozedur 'prüfeEndpreis'**
- **Drücken Sie den** "Zeile einfügen" Knopf über der Tabelle "Standardwerte für Parameter".
- **Tragen Sie `preis`** als Namen des Parameter ein.
- **Tragen Sie `30.049,00 €`** als Wert ein.
- Drücken Sie **OK**.

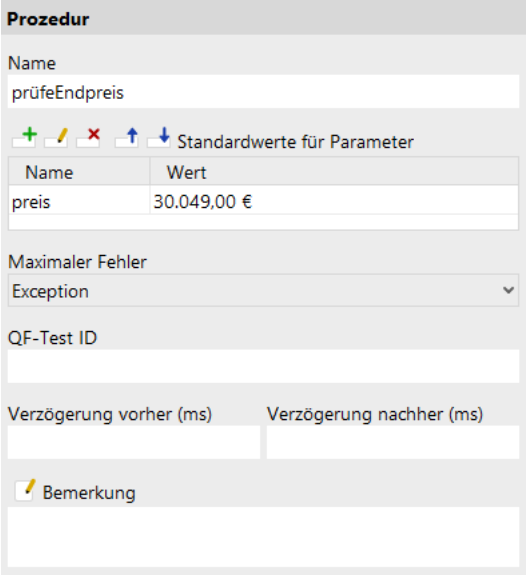

Abbildung 15.3: Die Details eines Prozedurknotens

Im nächsten Schritt ersetzen wir den Wert des Text Attributs des Check Text Knotens durch eine Referenz auf die Variable.

Hinweis

Variablensyntax: Wenn Sie mit Variablen arbeiten, ist es wichtig, sich bewusst zu sein, dass Sie an bestimmten Stellen QF-Test mitteilen wie, eine Variable heißt und an anderen, dass QF-Test auf den Wert einer Variablen zugreifen soll.

In obigem Beispiel wird QF-Test in der Namensspalte für die Standardwerte der Variablenname mitgeteilt. In diesem Fall brauchten Sie nur `preis` einzutragen.

Im Text Attribute des Check Text Knotens soll der Wert der Variablen verwendet werden. Bei QF-Test geschieht dies dadurch, dass Sie den Variablenname in `$()` setzen, hier `$(preis)`. Falls Sie den Variablennamen nicht in `$()` setzen, würde QF-Test den Preis mit der Zeichenfolge `preis` vergleichen.

Aktion

- **Selektieren Sie den Check Text Knoten** in der Prozedur ”prüfeEndpreis”.
- **Tragen Sie `$(preis)`** im Text Attribut der Check Text Knotendetails ein.
- **Drücken Sie ’OK’** in den Knotendetails.

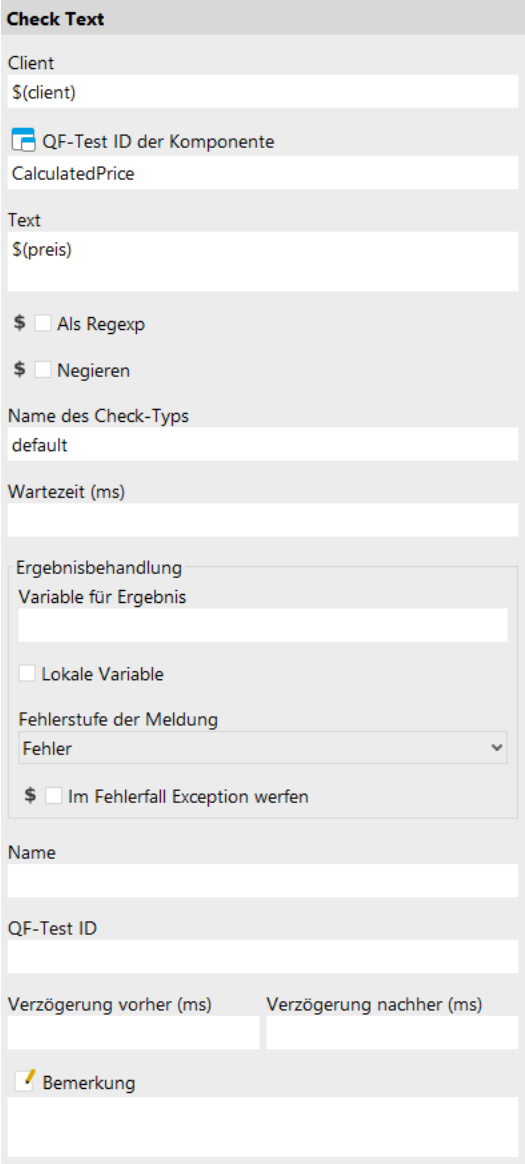

Abbildung 15.4: 'Check text'-Knoten

Aktion

- **Führen Sie den ersten Testfall aus.**

Der Testfall sollte fehlerfrei durchlaufen.

15.2 Die Variablendefinitionen-Tabelle

Im nächsten Schritt fügen wir einen Prozeduraufruf im zweiten Testfall ein.

Aktion

- **Ersetzen Sie den Check Text Knoten** des zweiten Testfalls **durch einen Prozeduraufruf von "prüfeEndPreis"**. Sie können einfach den Prozeduraufruf aus dem ersten Testfall kopieren oder den Prozeduraufruf wie oben beschrieben einfügen.

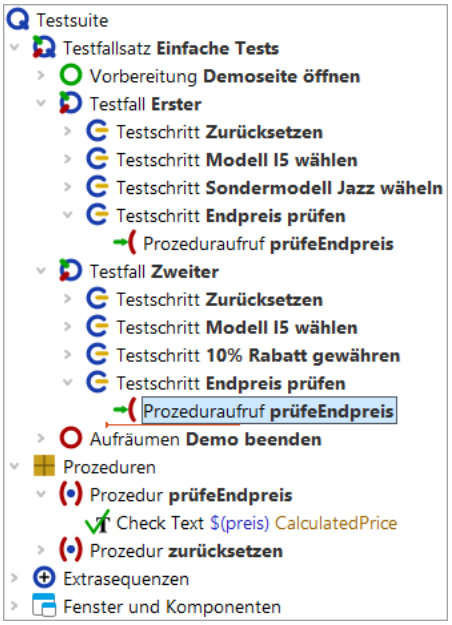

Abbildung 15.5: Prozeduraufruf von "prüfeEndPreis" in der zweiten Prozedur

Hinweis

Wenn im Prozeduraufruf bereits der Preis mit Standardwert eingetragen ist, rührt das daher, dass der Prozeduraufruf mit Hilfe der Prozedur selbst erzeugt wurde. Entweder durch Kopieren der Prozedur oder durch eine Drag-and-Drop Aktion unter Verwendung des Prozedurknoten oder über direktes Einfügen der Prozedur wie weiter oben erläutert. Aktuell geht es jedoch um den Standardwert. Daher bitten wir Sie, den evtl. vorhandenen Preis-Parameter zu löschen, wenn Sie dem Tutorial Schritt für Schritt folgen wollen. Dazu klicken Sie das rote X über der Variablendefinitionen-Tabelle.

Aktion

- Überprüfen Sie, ob der **Debugger so eingestellt ist, dass er bei Fehlern unterbricht** (siehe Debugger-Optionen: Test bei Fehler anhalten(167)).
- **Selektieren Sie den Knoten "Testfall: Zweiter"**.
- **Führen Sie ihn aus**, entweder über den Knopf ▶ oder durch Drücken der Eingabe Taste.

Eine Fehlermeldung zeigt an, dass ein anderer als der angezeigte Preis erwartet wurde. Was lief schief? Lassen Sie uns nach dem Fehler forschen. Üblicherweise würden wir ins Protokoll schauen, aber es gibt noch eine andere wichtige Informationsquelle.

Aktion

- **Klicken Sie OK**, um die Fehlermeldung zu schließen.

Im Debugging-Modus sehen Sie rechts unten im QF-Test Fenster eine Knotenliste mit Variablen, die von diesen gebunden sind.

Aktion

- Eventuell müssen Sie die Variablendefinitionen-Tabelle vergrößern, um alle Einträge sehen zu können.

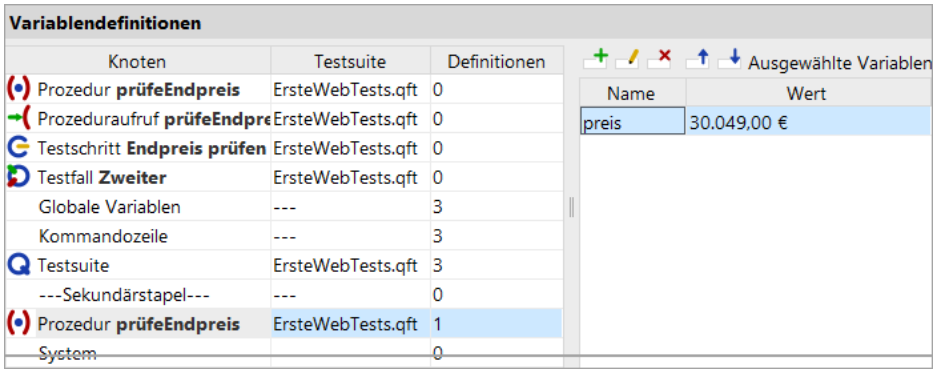

Abbildung 15.6: Variablendefinitionen

Die Variablendefinitionen-Tabelle ist beim Debuggen sehr hilfreich, da sie die aktuellen Werte der Variablen anzeigt. Sie unterstützt beim Arbeiten mit Prozeduren als auch beim Verständnis, wie QF-Test den richtigen Variablenwert ermittelt.

Hinweis

QF-Test geht die Variablendefinitionen-Tabelle immer von oben nach unten durch.

Sie sehen, dass in den ersten Zeilen keine Variablen gebunden sind. Auf der Ebene "Globale Variablen" ist eine Variable gebunden und auf dem Sekundärstapel für "Prozedur: prüfeEndpreis" eine weitere. Die globale Variable wird für die Verbindung zur SUT Applikation verwendet und wurde vor dem Anwendungsstart gesetzt. (vgl. Starten des Browsers(109)). Die andere Variable interessiert uns im Moment mehr - sie hat jedoch den falschen Wert.

Dieser Wert auf dem Sekundärstapel ist der Standardwert, da er dann verwendet wird, wenn nirgendwo sonst einer Variablen mit dem gleichen Namen ein Wert zugewiesen wurde.

Um es richtig zu machen, müssen wir den korrekten Wert beim Prozeduraufruf an die Prozedur übergeben. Wieder gibt es mehrere Arten, dies zu tun. Ein Weg wäre, eine neue Zeile in der Variablendefinitionen-Tabelle in den Details des Prozeduraufrufs einzufügen, ähnlich wie beim Prozedurknoten im vorigen Abschnitt.

Wenn es jedoch bereits mehrere Prozeduraufrufe gibt, ist folgendes einfacher:

Aktion

- **Beenden Sie die laufende Testausführung** mittels ■ .

- Führen Sie einen **Rechtsklick auf den Prozedurknoten** aus und wählen **Weitere Knotenoperationen→Parameter von Referenzen anpassen** im Popup-Menü.

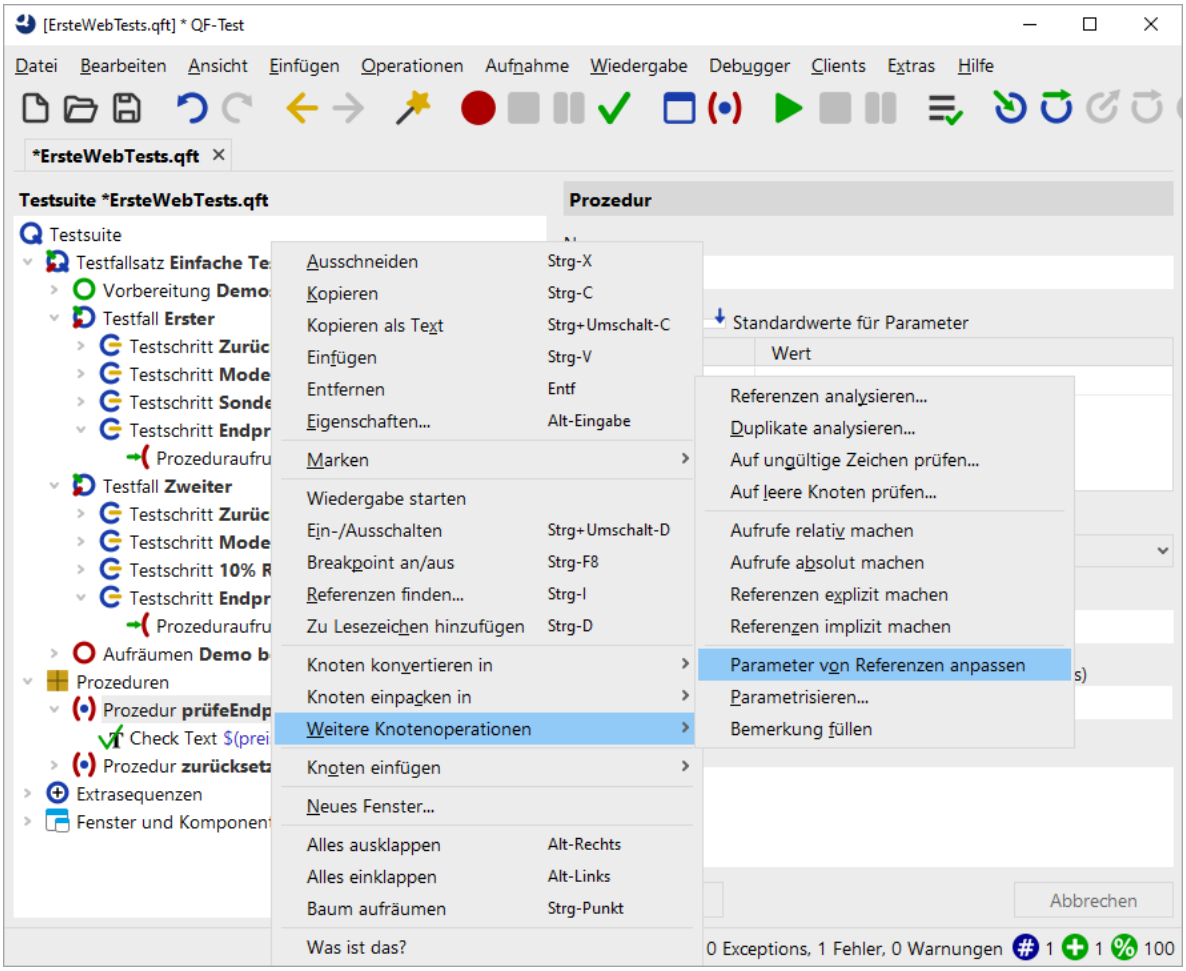

Abbildung 15.7: Popup-Menü für "Parameter von Referenzen anpassen"

- Prüfen Sie im folgenden Dialog, dass ein Häkchen bei **Fehlende Parameter beim Aufrufer hinzufügen** gesetzt ist, und **bestätigen Sie mit OK**.

Im Prozeduraufruf erzeugt QF-Test für jeden Standardwert eine Zeile in der Variablen Definitionen Tabelle. In unserem Fall wurde eine Zeile für den Parameter mit dem Namen `preis` und dem Wert `30.049,00` € hinzugefügt.

Auch damit wird es im zweiten Testfall noch nicht funktionieren, auch wenn der Wert direkt übergeben wird, weil es sich immer noch um den Standardwert handelt, der hier

nicht korrekt ist. Bitte verändern Sie den Wert noch nicht, damit wir Ihnen mittels des entstehenden Fehlers eine weitere Möglichkeit des Debuggens zeigen können.

Aktion

- **Schließen Sie den Dialog "Angepasste Knoten"**, den QF-Test anzeigt, um Sie über die vervollständigten Knoten zu informieren.

15.3 Fortgeschrittenes Debuggen mittels Variablendefinitionen-Tabelle

Als nächstes wollen wir die Variablendefinitionen-Tabelle unter die Lupe nehmen und herausfinden, wie man sie für Debugging-Zwecke einsetzen kann. Daher belassen Sie bitte den fehlerhaften Wert, der im vorigen Abschnitt im Prozeduraufruf eingefügt wurde.

Dazu soll die Ausführung des Testfalls beim Prozeduraufruf unterbrochen werden um dann mittels Einzelschritten in die Prozedur zu gehen. Dabei werden wir uns ansehen, was in der Variablendefinitionen-Tabelle passiert. Anschließend wollen wir direkt aus der Variablendefinitionen-Tabelle zum fehlerhaften Prozeduraufruf springen und dort den Parameterwert korrigieren.

Aktion

- **Setzen Sie einen Breakpoint** bei "Prozeduraufruf: prüfeEndpreis" im zweiten Testfall.
- **Führen Sie den zweiten Testfall aus**.
- Wenn QF-Test am Breakpoint anhält, **führen Sie zwei Einzelschritte in die Prozedur** mittels aus und **beobachten dabei die Variablendefinitionen-Tabelle**.

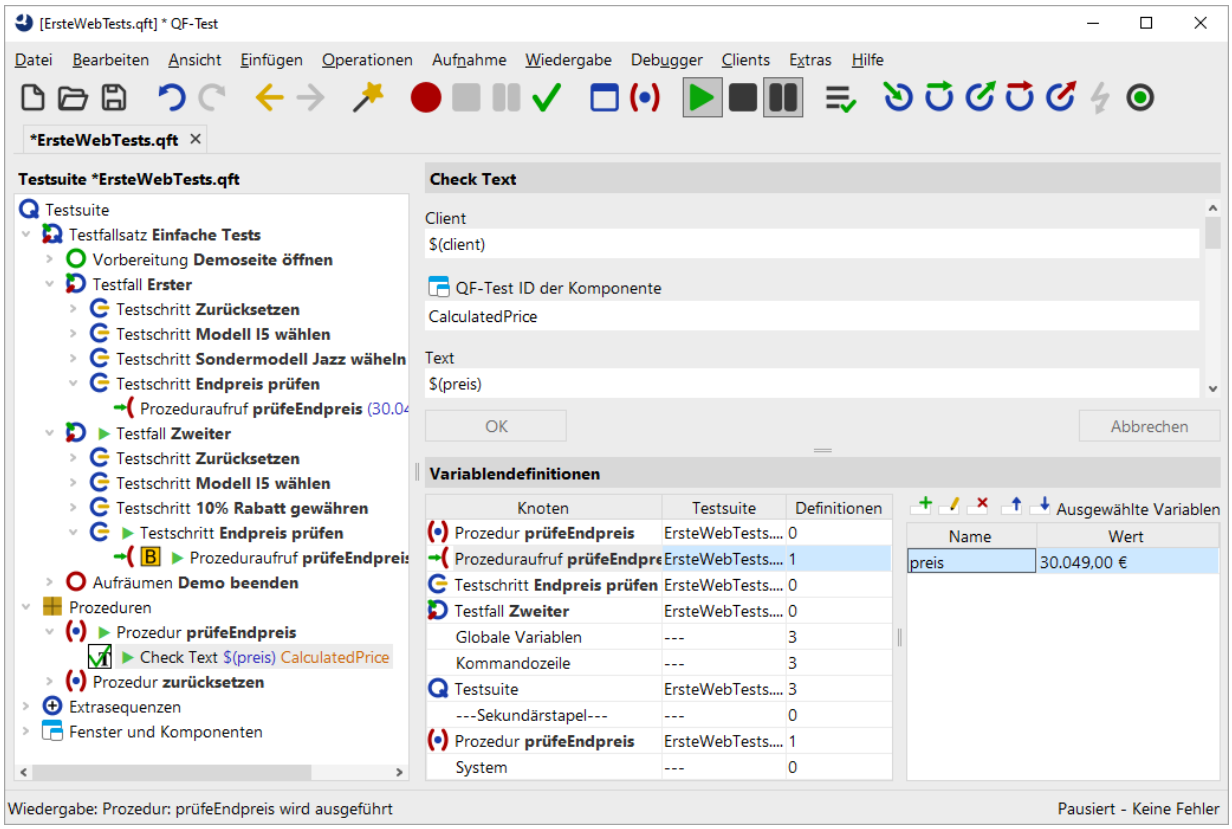

Abbildung 15.8: Variablendefinitionen-Tabelle zeigt den falschen Wert

Wenn Sie mit Einzelschritten in die Prozedur gehen, wird beim ersten eine Zeile für "Prozeduraufruf: prüfeEndpreis" oben in der Tabelle erzeugt und beim zweiten die Zeile "Prozedur: prüfeEndpreis".

Jetzt gibt es die Variable `preis` auf zwei verschiedenen Ebenen in der Variablen Definitionen Tabelle: in der Zeile für "Prozeduraufruf: prüfeEndpreis" und in der Zeile "Prozedur: prüfeEndpreis" auf dem Sekundärstapel, wobei keiner der beiden Variablenwerte der richtige ist.

In QF-Test können Sie interaktiv die Werte von Variablen in der Variablendefinitionen-Tabelle verändern, wenn Sie sich im Debugging-Modus befinden. Sie können sogar neue Variablen hinzufügen oder vorhandene löschen. Damit können Sie arbeiten, solange sich die Variablen auf dem Variablenstapel befinden, in unserem Fall solange wie die Prozedur ausgeführt wird.

Änderungen des aktuellen Variablenwertes in der Variablendefinitionen-Tabelle bewirken keine Anpassung des im Prozeduraufrufknoten eingetragenen Parameterwerts. Der Wert muss explizit im Prozeduraufruf geändert werden.

Die schnellste Methode um zum Prozeduraufruf zu gelangen ist ein Doppelklick auf die

Prozeduraufrufzeile (zweite Zeile) in der Variablendefinitionen-Tabelle. Diese Methode ist besonders hilfreich, wenn Sie umfangreiche Testsuiten debuggen und der Knoten, zu dem Sie springen wollen, nicht im Testsuite-Fenster angezeigt wird. Statt eines Doppelklicks können Sie auch einen Rechtsklick auf die Zeile ausführen und den Menüpunkt Zu Knoten in Testsuite springen wählen.

Aktion

- **Führen Sie einen Doppelklick auf die zweite Zeile mit dem Prozeduraufruf** in der Variablendefinitionen-Tabelle aus.
- **Setzen Sie den Parameterwert auf den richtigen Wert**, d.h. `26.100,00 €`.

Umgekehrt wird auch der aktuelle Wert in der Variablendefinitionen-Tabelle dadurch nicht verändert. Um dies zu erreichen müssen wir den Prozeduraufruf erneut ausführen. Allerdings ist die Testausführung über diesen Punkt bereits hinaus.

Hinweis

Daher wollen wir hier eine weitere nützliche Funktion des Debuggers zeigen, mit der man den QF-Test anweisen kann, den nächsten auszuführenden Knoten zu verändern. Dazu selektieren Sie den entsprechenden Knoten und wählen den Menüpunkt **Ausführung hier fortsetzen** oder verwenden das Tastaturkürzel Strg-,.

Also, um den neu gesetzten Wert auszuprobieren:

Aktion

- **Führen Sie einen Rechtsklick auf den Knoten "Prozeduraufruf: prüfeEndpreis"** in der zweiten Prozedur aus.
- **Wählen Sie "Ausführung hier fortsetzen"** im Popup-Menü.

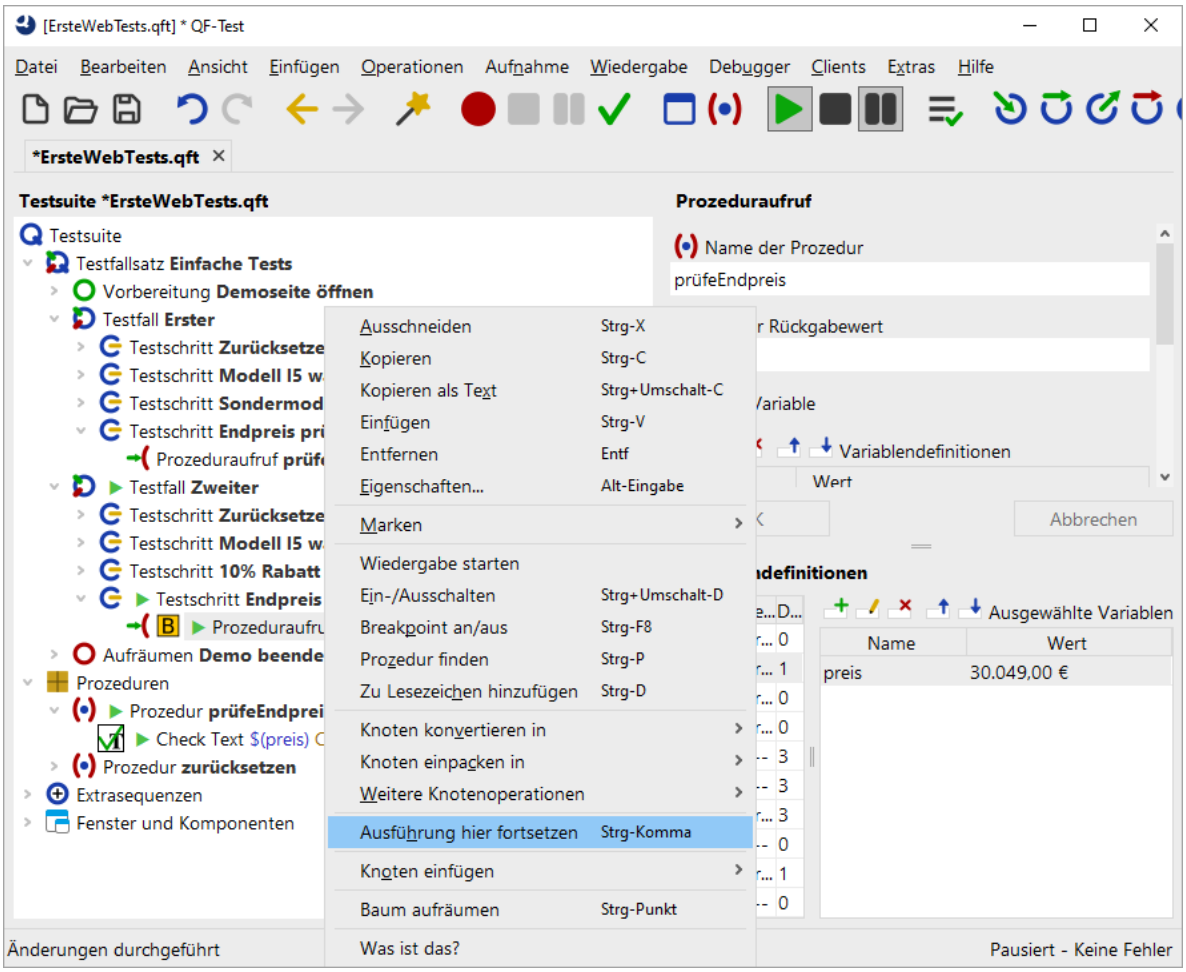

Abbildung 15.9: Ausführung hier fortsetzen

In der Variablendefinitionen-Tabelle sind die zwei obersten Zeilen verschwunden. Der Grund ist, dass Sie die Prozedur verlassen haben (wenn auch "rückwärts") und dass dadurch der Prozeduraufruf mit den daran gebundenen Variablen vom Aufrufstack genommen wurde.

Aktion

- Lösen Sie den Pauseknopf ⏸ .

Nun sollte kein Fehler mehr auftauchen.

Hinweis

Da die Variablendefinitionen-Tabelle äußerst hilfreich ist, wenn Sie nach fehlerhaften Variablenwerten fahnden, wird eine Kopie davon auch unter dem Knoten "Stacktrace" im Protokoll abgespeichert, in dem die Variablenwerte genau zum Zeitpunkt des Fehlers zu sehen sind.

Aktuellen Knoten finden: Manchmal entfernt man sich beim Debuggen ziemlich weit vom aktuellen Knoten und möchte anschließend wieder zu diesem Knoten zurückfinden.

Das geht am einfachsten indem man in der Toolbar "Aktuellen Knoten finden" drückt oder den Menüpunkt Debugger→Aktuellen Knoten finden wählt.

15.4 Variablen setzen

Zusätzlich zu den oben beschriebenen Wegen können die Variablen auch wie folgt gesetzt werden:

- Mittels Variable setzen Knoten,
- als Rückgabewert einer Prozedur,
- als Ergebnis eines QF-Test Knotens wie Text auslesen, Geometrie auslesen, Index auslesen und Check,
- in der 'Variablendefinitionen' Tabelle von Testsuite, Testfall, Testschritt, Sequenz und weiteren Knoten wie dem If oder Schleife Knoten,
- über Kommandozeilenparameter.

Informationen dazu, an welcher Stelle eine Variable am besten definiert wird, finden Sie im nächsten Abschnitt.

Ein Variable setzen Knoten kann über den Menüpunkt Einfügen→Diverse Knoten→Variable setzen eingefügt werden. In den Knotendetails können Sie angeben, ob es sich um eine lokale (Häkchen bei "Lokale Variable" setzen) oder eine globale Variable handeln soll.

Die folgende Abbildung zeigt die Details eines Variable setzen Knotens, den Sie als ersten Knoten im Vorbereitung Knoten finden. Es wird eine Variable mit dem Namen `client` definiert. Dass es sich um eine globale Variable handelt, erkennen Sie daran, dass das Attribut 'Lokale Variable' nicht gesetzt ist.

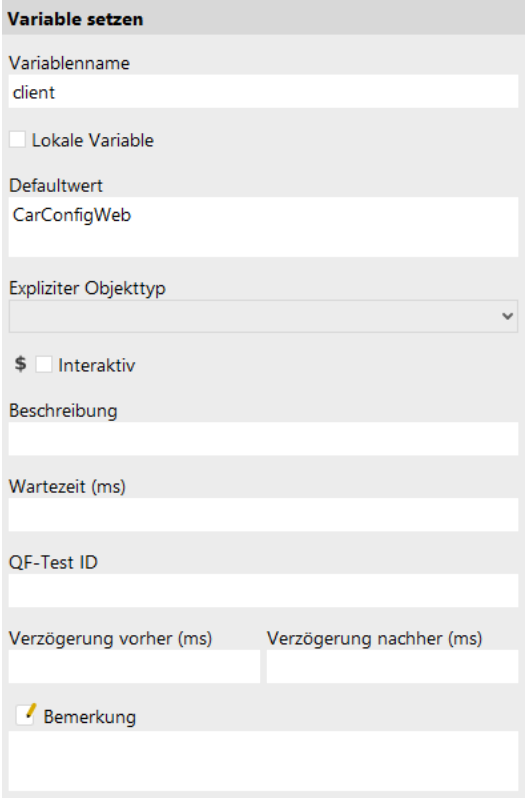

Abbildung 15.10: Details des Variable setzen Knoten

Wenn eine Variable mit dem Rückgabewert einer Prozedur gesetzt werden soll, geben Sie den Variablennamen im Attribut "Variable für Rückgabewert" des Prozeduraufrufs an. In der Prozedur selbst müssen Sie als letzten auszuführenden Knoten einen Return Knoten einfügen, der den betreffenden Wert zurückgibt.

Die Prozedur in der folgenden Abbildung liest den Rabattwert aus dem SUT und gibt den Wert an den aufrufenden Testfall zurück. Dort heißt die empfangende Variable `Rabatt` und ist als lokale Variable deklariert. Dieses Beispiel ist nicht in der Übungstestsuite enthalten.

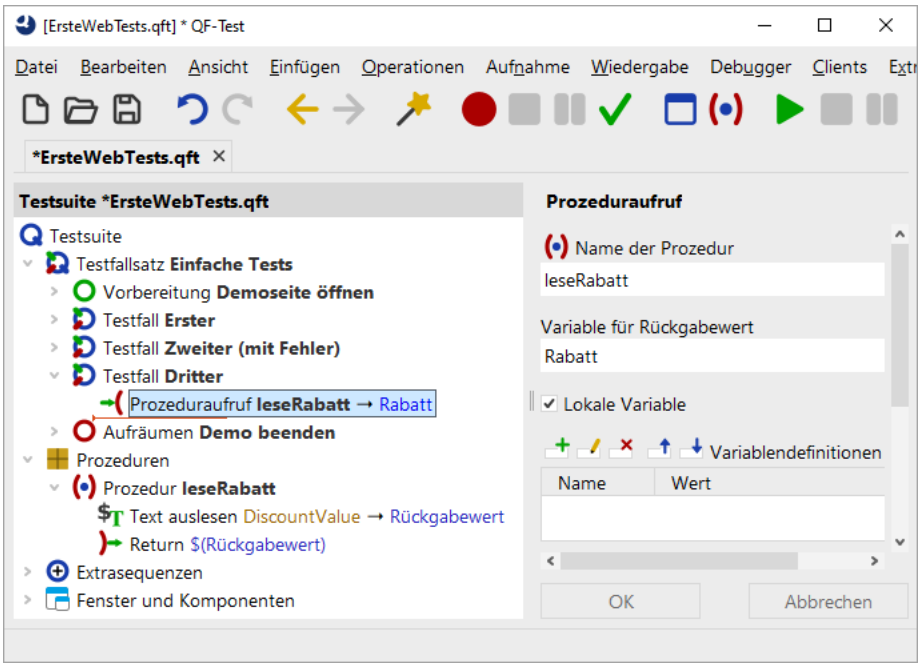

Abbildung 15.11: Prozedur mit Rückgabewert

Der Text auslesen Knoten in der obigen Abbildung ist einer der QF-Test Knoten, die direkt den Wert einer Variablen setzen. Dabei wird der Variablenname in dem entsprechend benannten Attribut eingetragen. Sie haben wiederum die Wahl, ob es eine lokale oder globale Variable werden soll.

Es gibt eine Reihe von Knoten, die eine Variablendefinitionen-Tabelle besitzen. Dort können Sie lokale Variablen setzen. Falls sich der betreffende Knoten in einer Prozedur befindet, wird die Variable als lokale Variable an die Prozedur gebunden, ansonsten als lokale Variable an den Testfall. Variablen, die mittels dieser Tabelle an den Testsuite Knoten gebunden sind, können von allen Knoten der Testsuite referenziert werden.

Alle Knoten, an die Variablen gebunden werden können, werden im Debugger-Modus im Variablen Definitionen Fenster rechts unten angezeigt, wenn sie gerade ausgeführt werden.

Variablen können auch über die Kommandozeile spezifiziert werden. Hierzu verwenden Sie den Kommandozeilenparameter `-variable`. Beispiel: `qftest -batch -variable "browser"="ie" test.qft`. Weitere Informationen hierzu finden Sie im Handbuch, Kapitel 'Kommandozeilenargumente'.

15.5 Ebenen für Variablendefinitionen

Hinweis

Dieser Abschnitt gibt Antworten auf die Frage, auf welcher Ebene eine Variable definiert werden sollte. Wenn Sie diese Frage momentan nicht interessiert, können Sie direkt zum nächsten Kapitel springen.

Variablen können auf unterschiedlichen Ebenen gebunden werden:

- Im Testsuite Knoten,
- in Testfällen und Prozeduren als Standard- oder als lokale Variablen,
- als Parameter in einem Prozeduraufruf,
- als globale Variable und
- als Kommandozeilenparameter.

Die Ebene, auf welcher eine Variable am sinnvollsten definiert wird, hängt vom Verwendungszweck der Variablen ab:

Prozedurparameter
Übergeben Sie einen Wert als Parameter an eine Prozedur, wenn die gleiche Prozedur mehr als einmal und mit unterschiedlichen Werten ausgeführt werden soll. Prozedurparameter werden in der Variable Definitionen Tabelle eines Prozeduraufruf Knoten angegeben.

Lokale Variablen in einer Prozedur
Lokale Variablen werden innerhalb der Prozedur definiert und sind nur dort gültig. Wenn die Prozedur beendet wird, werden sie gelöscht. Verwenden Sie eine lokale Variable, wenn diese nicht außerhalb der Prozedur benötigt wird. Sie sind das Mittel der Wahl für Zwischenergebnisse.

Lokale Variablen in einem Testfall
In einem Testfall können lokale Variable entweder während der Durchführung des Testfall angelegt werden oder über die entsprechende Tabelle in den Details des Testfall Knotens. Wenn Sie in einem Testfall mehrfach den gleichen Wert verwenden, ist es sinnvoll, diesen einmalig einer Variablen zuzuweisen und dann die Variable zu verwenden. Dies erhöht die Wartbarkeit. Auch für Zwischenergebnisse sollte man lokale Variablen verwenden.

Globale Variablen
Wenn globale Variablen einmal erstellt wurden, existieren sie, bis sie entweder explizit gelöscht werden oder bis QF-Test beendet wird. Auch Stopps und die erneute Ausführung von Tests ”überleben” sie. Verwenden Sie sie für

Werte, die in mehreren Testfällen genutzt werden. Ein Beispiel ist die Variable `client`, die im Vorbereitung Knoten beim Start der Applikation angelegt wird. Um sie wieder loszuwerden, wählen Sie den Menüpunkt Wiedergabe→Globale Variablen löschen. Auch beim Beenden von QF-Test werden sie gelöscht.

Kommandozeilenparameter

Variablen, die über Kommandozeile gesetzt werden, sind im Batch-Modus sinnvoll, wenn Sie mehrere Batch-Läufe mit unterschiedlichen Werten durchführen wollen. Kommandozeilenparameter gelten während der gesamten Laufzeit des Batch-Laufs. Ein typisches Beispiel ist die variablengesteuerte Ausführung auf verschiedenen Browsern. Variablen können über den Kommandozeilen-Parameter `-variable` spezifiziert (vgl. Kapitel 'Kommandozeilenargumente' im Handbuch).

Testsuite-Variablen

Testsuite-Variablen können von allen Testfällen verwendet werden. Ihr Verwendungszweck entspricht dem von globalen Variablen, nur dass sie im Batch-Modus durch Variablen in der Kommandozeile überschrieben werden können.

Standardwerte (Sekundärstapel)

Sie können Standardwerte für die Variablen von Prozeduren, Testfällen und Testfallsätzen definieren. Diese kommen zum Zug, wenn keine Variable mit dem gleichen Namen auf einer höheren Ebene definiert wurde.

Kapitel 16

Die Standardbibliothek (Web)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Die Standardbibliothek'
https://www.qftest.com/de/yt/tutorial-7.html

QF-Test stellt eine gewisse Anzahl an Knotentypen bereit, die für die Testerstellung genutzt werden können. Wenn Sie Funktionalität benötigen, die darüber hinausgeht, können Sie diese mittels Skript-Knoten implementieren. Um Ihnen die Arbeit zu erleichtern, wurden viele Funktionen, die häufig benötigt werden, bereits in Prozeduren implementiert und werden in einer Standard-Prozeduren-Bibliothek mit QF-Test ausgeliefert.

Wenn Sie also eine Aufgabenstellung haben, die nicht über die bereitgestellten Knoten gelöst werden kann, sollten Sie zunächst einmal in der Standardbibliothek forschen, ob Sie dort eine passende oder ähnliche Lösung finden. Wenn Sie eine ähnliche Lösung finden, kopieren Sie einfach die vorhandene Standardprozedur und passen sie Sie gemäß Ihrer Bedürfnisse an. Informationen zum Arbeiten mit Skripten erhalten Sie im Handbuch, Kapitel 12 "Skripting".

Die Bibliothek ist in der Datei `qfs.qft` enthalten und ist Teil der QF-Test Distribution. Da sie mit jeder QF-Test Version weiterentwickelt wird, ist es nicht ratsam, Änderungen in der ausgelieferten Standardbibliothek vorzunehmen, sondern bei Bedarf eine Prozedur in eine eigene Testsuite zu kopieren und dort anzupassen.

Um die Bibliothek `qfs.qft` verwenden zu können, muss sie im "Testsuite" Wurzelknoten Ihrer Suite in den "inkludierten Dateien" eingebunden werden. Bei neuen Testsuiten ist dies automatisch der Fall.

Aktion

- Wählen Sie den 'Testsuite'-Wurzelknoten Ihrer Testsuite aus.
- Überprüfen Sie in den Details des 'Testsuite'-Wurzelknoten, dass `qfs.qft` im Attribut "Inkludierte Dateien" aufgeführt ist.
- Fügen Sie `qfs.qft` zu dieser Liste dazu, falls es noch nicht enthalten ist.

Hinweis Eine Pfadangabe ist nicht notwendig, da das `include` Verzeichnis von QF-Test automatisch im Bibliothekspfad (siehe auch Referenzteil des Handbuchs) enthalten ist.

Im Folgenden beschreiben wir eine Auswahl der am häufigsten benötigten Standardprozeduren. Eine vollständige HTML-Dokumentation der Standardbibliothek finden Sie unter dem Menüpunkt Hilfe→Standardbibliothek qfs.qft...

16.1 Erforschen der Standardbibliothek

Zusätzlich zum Einfügen von Prozeduraufrufen aus der Standardbibliothek ist es hilfreich, einen Blick darauf zu werfen, wie Funktionen implementiert und organisiert sind.

Aktion

- Öffnen Sie die Bibliothek selbst, also die Suite `qfs.qft`, die sich im Verzeichnis `qftest-9.0.0/include` Ihrer QF-Test Installation befindet.

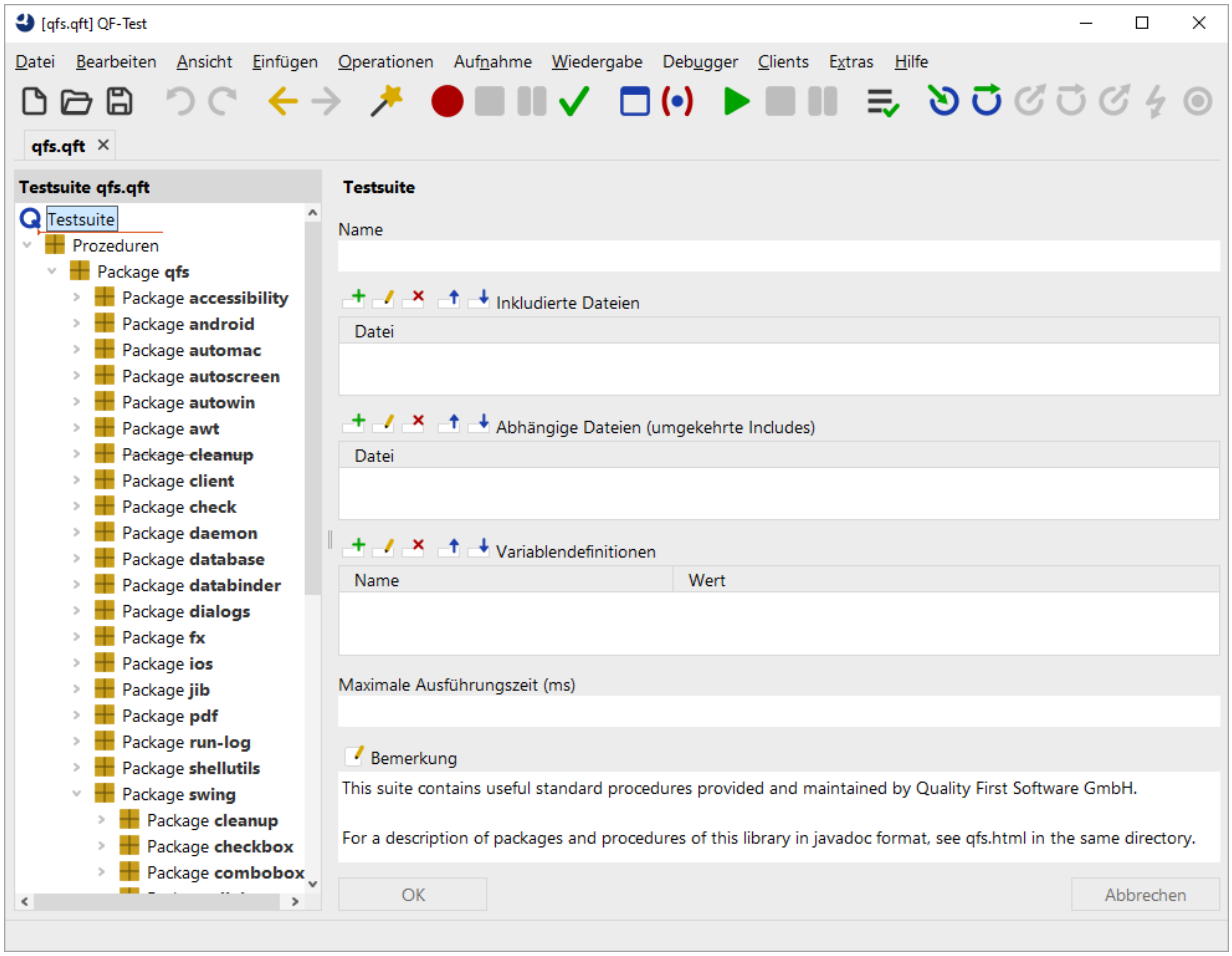

Abbildung 16.1: Die Standardbibliothek

Sie sehen ein Haupt-Package `qfs`, das die spezifischen Packages umschließt. Das `qfs` Package hilft dabei, die Prozeduren leicht als solche der Standardbibliothek zu identifizieren.

In fast allen Prozeduren unserer Bibliothek werden Sie die Verwendung der Variable `$(client)` bemerken. Dies ist ein Standardmechanismus, um Testsuiten unabhängig von einem spezifischen SUT zu gestalten. Für die Benutzung der Standardbibliothek wird vorausgesetzt, dass ein gültiger Wert für `$(client)` gesetzt wird, bevor eine ihrer Prozeduren verwendet werden kann.

16.2 Ausgewählte Packages und Prozeduren

Wir werfen nun einen genaueren Blick auf ein paar ausgewählte Packages und Prozeduren der Standardbibliothek.

Wir werden mit Packages beginnen, die den Zugriff auf Komponenten abhängig von der verwendeten GUI Technologie, hier Web, ermöglichen.

16.2.1 Das Checkbox Package

Wir beginnen nun mit der genaueren Betrachtung des Packages `qfs.web.checkbox`.

Hier sind einige Prozeduren aus diesen Packages:

- **select** Selektiert ein Kontrollkästchen. Wenn sich das Kontrollkästchen bereits im ausgewählten Zustand befindet, wird keine Aktion ausgeführt.
- **deselect** Deselektiert ein Kontrollkästchen. Wenn sich das Kontrollkästchen bereits im nicht-ausgewählten Zustand befindet, wird keine Aktion ausgeführt.
- **set** Setzt ein Kontrollkästchen auf den angegebenen Zustand (true oder false).

Für jede dieser Prozeduren wird die QF-Test ID der Kontrollkästchenkomponente als variables Argument übergeben. Die Bibliothek kümmert sich um die Überprüfung, dass der Zustand des Kontrollkästchens wie erwartet gesetzt wurde.

Die anderen Prozeduren in diesem Package folgen demselben Muster.

16.2.2 Das Select Package

Das Package `qfs.web.select` enthält Prozeduren, um Werte in einer Combobox oder eines anderen Select-Elements zu selektieren.

Die wichtigsten Prozeduren sind:

- **setValue** Selektiert einen Wert in der Liste der Combobox.
- **getItemCount** Liefert die Anzahl der Einträge zurück.

16.2.3 Das General Package

Das Package `qfs.web.general` enthält allgemeine Prozeduren für GUI-Elemente.

Die wichtigsten Prozeduren sind:

- **setLocation** Setzt die Position der Komponente mittels angegebenen Koordinaten.
- **setSize** Setzt die Größe der Komponente.

16.2.4 Das Table Package

Das Package `qfs.web.table` enthält Hilfsprozeduren für Tabellen.

- **getRowCount** Liefert die aktuelle Zeilenanzahl einer Tabelle zurück. Diese Prozedur verwendet technologiespezifische Methoden um an die Anzahl zu kommen.
- **getColumnCount** Liefert die aktuelle Spaltenanzahl einer Tabelle zurück. Diese Prozedur verwendet technologiespezifische Methoden um an die Anzahl zu kommen.
- **selectCell** Selektiert eine angegebene Tabellenzelle.

16.2.5 Das Cleanup Package

Die Packages `qfs.fx.cleanup`, `qfs.swing.cleanup` und `qfs.swt.cleanup` bieten eine gute Unterstützung für das Aufräumen der SUT Umgebung, wenn unerwartet eine Exception auftritt. Stellen Sie sich zum Beispiel vor, dass eine Exception geworfen wird, während auf ein Menü des SUTs zugegriffen wird. Die Exception bewirkt, dass der Ausführungspfad innerhalb Ihrer Testsuite zu einem Exception Handler umgeleitet wird - oder zu einem ”impliziten” Exception Handler. Das bedeutet, dass der normale Ausführungspfad, der das geöffnete Menü in der Regel wieder ordnungsgemäß geschlossen hätte, unterbrochen wurde. Ohne eine entsprechende Aktion kann dieses Menü geöffnet bleiben und somit andere Ereignisse an das SUT blockieren.

Hier sehen Sie die wichtigsten Prozeduren innerhalb des Packages:

- **closeAllModalDialogs** bewirkt, dass modale Dialoge des SUTs geschlossen werden. **Nur für Swing und FX verfügbar**!
- **closeAllDialogsAndModalShells** bewirkt, dass alle Dialoge und modalen Shells geschlossen werden. **Nur für Eclipse/SWT verfügbar**!
- **closeAllMenus** Schließt alle offenen Menüs des SUT.

Das Konzept zur Behandlung von impliziten Exceptions ist von großer Bedeutung, denn eine Exception in einem einzigen Testfall soll nicht zum Beenden des gesamten Testlaufs führen. Lediglich der aktuelle Testfall soll abgebrochen werden, dann sollte es mit dem nächsten Testfall weitergehen.

Aus diesem Grund wird eine Exception innerhalb eines Testfalls auf dieser Ebene gefangen und nicht nach oben propagiert. Dies verhindert den Abbruch des gesamten Testlaufs. Der Fehlerstatus wird jedoch stets korrekt im Protokoll und Report festgehalten.

Benutzt der Testfall eine Abhängigkeit, wird die Exception an den Catch Knoten derselben übergeben, falls ein solcher vorhanden ist. Diese Art der Behandlung von Exceptions (und Fehlern) wird im Kapitel Abhängigkeiten des Handbuchs erklärt.

16.2.6 Das Run-log Package

Das Package `qfs.run-log` enthält Prozeduren, um Meldungen in das Protokoll zu schreiben.

Hier sehen Sie die Liste von verfügbaren Prozeduren innerhalb des Packages:

- **logError** Schreibt eine Fehlermeldung ins Protokoll.
- **logWarning** Schreibt eine Warnung ins Protokoll.
- **logMessage** Schreibt eine Meldung ins Protokoll.

16.2.7 Das Run-log.Screenshots Package

Das `qfs.run-log.screenshots` Package enthält Prozeduren, die Bildschirmabbilder ins Protokoll schreiben und einige Hilfsprozeduren.

Hier sehen Sie die Liste von verfügbaren Prozeduren innerhalb des Packages:

- **getMonitorCount** Liefert die Anzahl der an den Computer angeschlossenen Monitore.

- **logScreenshot** Schreibt ein Bildschirmabbild des aktuellen Monitors ins Protokoll.
- **logImageOfComponent** Schreibt ein Bildschirmabbild einer Komponente ins Protokoll.
- **logScreenshotOfMonitor** Schreibt ein Bildschirmabbild eines angegebenen Monitors ins Protokoll.

16.2.8 Das Shellutils Package

Das `qfs.shellutils` Package beinhaltet Prozeduren für die wichtigsten Shell-Kommandos.

Hier sehen Sie die Liste von verfügbaren Prozeduren innerhalb des Packages:

- **copy** Kopiert eine angegebene Datei oder ein Verzeichnis an eine neue Stelle.
- **deleteFile** Löscht eine angegebene Datei.
- **exists** Prüft, ob eine angegebene Datei oder ein Verzeichnis existiert.
- **getBasename** Gibt den Dateinamen einer Datei zurück.
- **getParentDirectory** Gibt die Verzeichnisstruktur einer Datei zurück.
- **mkdir** Erzeugt ein Verzeichnis. Noch nicht existierende Verzeichnisse werden angelegt.
- **move** Verschiebt eine angegebene Datei oder ein Verzeichnis.
- **touch** Erzeugt eine Datei.
- **removeDirectory** Löscht ein angegebenes Verzeichnis.

16.2.9 Das Utils Package

Das Package `qfs.utils` enthält nützliche Prozeduren für häufig auftretende Anforderungen der Testentwicklung.

Hier sehen Sie einige Prozeduren des Packages:

- **getDate** Gibt einen String zurück, der ein Datum enthält. Standardmäßig wird das aktuelle Datum zurückgegeben. (Andere Daten sind konfigurierbar.)
- **getTime** Gibt einen String zurück, der eine Zeit enthält. Standardmäßig wird die aktuelle Zeit zurückgegeben. (Andere Zeiten sind konfigurierbar.)

- **logMemory** Schreibt den aktuellen Speicherverbrauch ins Protokoll.
- **printVariable** Gibt den Inhalt einer spezifizierten Variable auf der Konsole aus.
- **printMessage** Gibt den Inhalt einer angegebenen Nachricht auf der Konsole aus.
- **writeMessageIntoFile** Schreibt einen angegebenen String in eine angegebene Datei.

16.2.10 Das Database Package

Das Package `qfs.database` enthält nützliche Prozeduren, um mit Datenbanken zu interagieren.

Bitte beachten Sie, dass die jar-Datei mit dem Datenbanktreiber vor dem Start von QF-Test ins `qftest` Pluginverzeichnis kopiert werden muss.

Für weitere Informationen über den Aufbau einer Datenbankverbindung kontaktieren Sie bitte einen Entwickler oder werfen Sie einen Blick auf www.connectionstrings.com.

Die wichtigsten Prozeduren sind:

- **executeSelectStatement** Führt einen angegebenen SQL-Select-Befehl aus. Das Ergebnis wird zum einen in die globale Variable ”resultRows” des Jython Variablenstacks geschrieben und ist somit in jedem Jython Skript verfügbar. Zum anderen wird das Ergebnis auch in eine Gruppenvariable mit dem Standardnamen ”resultGroup” geschrieben und ist somit direkt von QF-Test Knoten aus ansprechbar.
- **executeStatement** Führt einen angegebenen SQL Befehl aus. Hier kann jedes beliebige SQL Kommando ausgeführt werden.

16.2.11 Das Check Package

Das `qfs.check` Package enthält Prozeduren, die Checks ausführen.

Die wichtigsten Prozeduren sind:

- **checkEnabledStatus** Überprüft, ob eine Komponente en- bzw. disabled ist. Im Fehlerfall wird von der Prozedur ein entsprechender Fehler ins Protokoll geloggt.
- **checkSelectedStatus** Überprüft, ob eine Komponente selektiert bzw. nicht selektiert ist. Im Fehlerfall wird von der Prozedur ein entsprechender Fehler ins Protokoll geloggt.
- **checkText** Überprüft den Text einer Komponente. Im Fehlerfall wird von der Prozedur ein entsprechender Fehler ins Protokoll geloggt.

16.2.12 Das Databinder Package

Das Package `qfs.databinder` enthält Prozeduren zur Ausführung innerhalb eines Datentreiber Knotens, um Daten für datengetriebenes Testen zu binden.

Die wichtigsten Prozeduren sind:

- **bindList** Bindet eine Liste von Werten an eine Variable. Die Werte sind durch Leerzeichen oder das als Parameter übergebene Trennzeichen getrennt.
- **bindSets** Bindet Sätze von Werten an einen Satz von Variablen. Die Sätze von Werten sind durch Zeilenumbrüche getrennt, die Werte innerhalb eines Satzes durch Leerzeichen oder das als Parameter übergebene Trennzeichen.

Kapitel 17

Ablaufsteuerung (Web)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Ablaufsteuerung'
https://www.qftest.com/de/yt/tutorial-8.html

Die zwei wichtigsten Kontrollstrukturen von QF-Test sind Schleifen und die bedingte Ausführung von Knoten. Schleifen können über zwei verschieden Knoten implementiert werden: While und Schleife Knoten. If, Elseif und Else Knoten stehen für die bedingte Ausführung von Knoten zur Verfügung.

17.1 If - else

If Knoten kennen Sie bereits aus der Vorbereitung Sequenz im Kapitel Starten des Browsers[109]. Sehen wir uns diesen nun etwas genauer an.

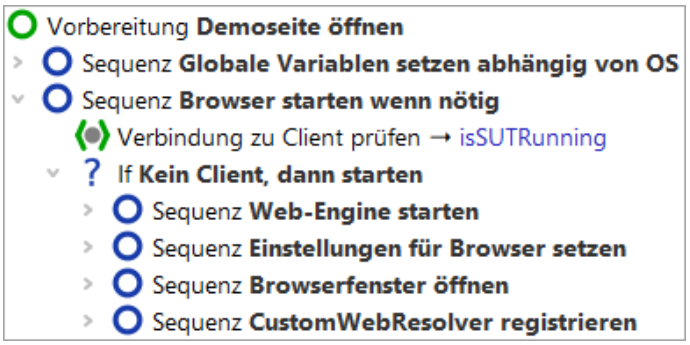

Abbildung 17.1: Setup Sequenz mit If/Elseif Knoten

Über einen If Knoten können Sie steuern, ob bestimmte Knoten ausgeführt werden

oder nicht. In unserem Fall geht es um den Start des SUT. Zunächst müssen wir herausfinden, ob die Applikation bereits läuft. Dies geschieht über den Warten auf Client Knoten, der als Ergebnis entweder `true` (wahr) oder `false` (falsch) in die Variable `isSUTRunning` schreibt.

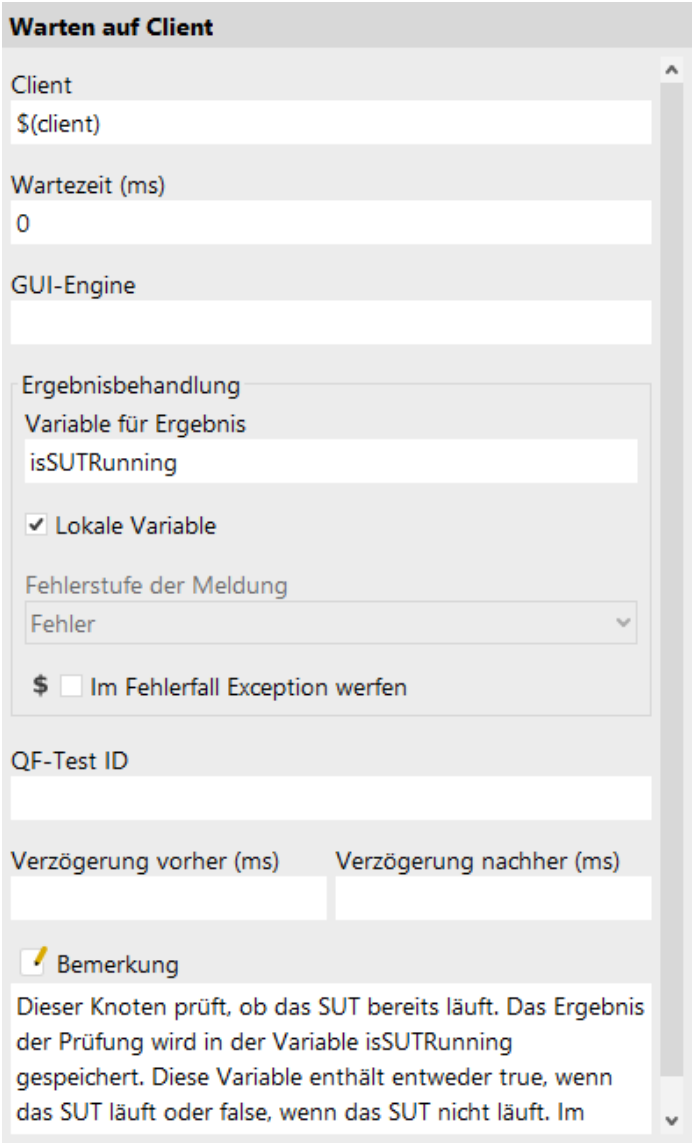

Abbildung 17.2: Warten auf Client setzt die Variable "isSUTRunning" mit dem Ergebnis

Der If Knoten wertet die Ergebnisvariable `isSUTRunning` im 'Bedingung' Attribut aus. Da auf den Wert der Variablen zugegriffen werden soll, wird die Syntax `$()` verwendet (vgl. Hinweis zu Variablensyntax in Kapitel Abschnitt 15.1 (172)).

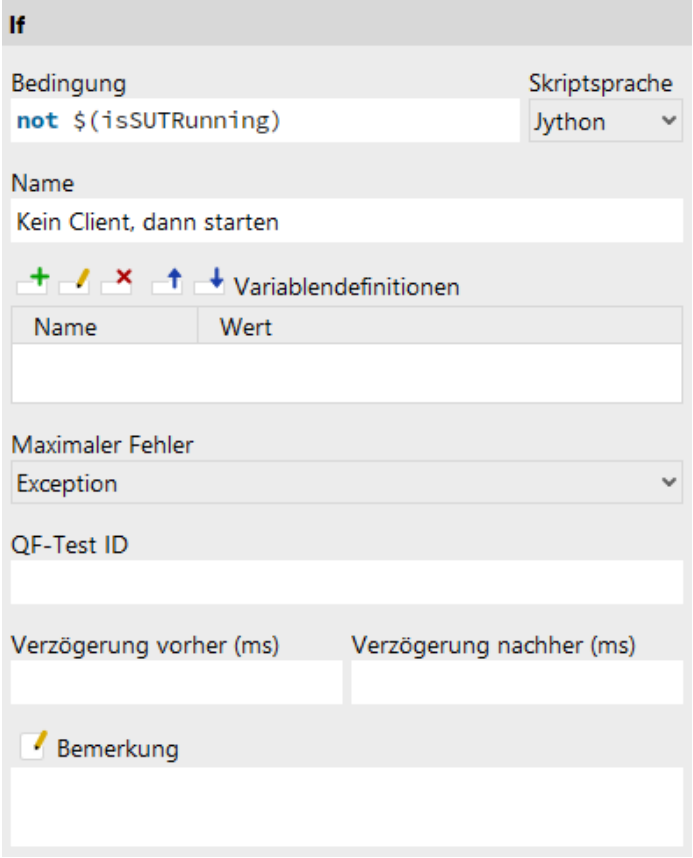

Abbildung 17.3: Der If Knoten wertet die Variable aus

Je nachdem, ob die Applikation bereits läuft, führt QF-Test die im If Knoten enthaltenen Knoten aus oder nicht.

Aktion

- **Beenden Sie das CarConfig Demo**, falls es läuft.
- **Führen Sie den Vorbereitung Knoten mit Einzelschritten aus.**
- **Führen Sie den Vorbereitung Knoten nochmals mit Einzelschritten aus** während das CarConfig Demo läuft.

In der Variablendefinition-Tabelle können Sie den Wert der Variablen `isSUTRunning` prüfen. Beim ersten Mal ist der Wert `false` und damit die Bedingung `not $(isSUTRunning)` wahr, d.h. die Knoten für den SUT-Start werden ausgeführt. Beim zweiten Mal ist der Wert `true` und damit die Bedingung `false`. Die Knoten im If Knoten werden übersprungen.

Hinweis

Im ersten Knoten der Vorbereitung werden If Knoten verwendet, um abhängig vom Be-

triebssystem den zu verwendenden Web-Browser in einer globalen Variable zu hinterlegen. Der besseren Lesbarkeit halber werden hier nur If Knoten verwendet. Es wäre ebenso möglich mit Elseif und Else Knoten zu arbeiten. In einem Elseif Knoten wird eine weitere Bedingung formuliert, die dann geprüft wird, wenn die Bedingung im If Knoten nicht zutrifft. Die Kindknoten des Else Knoten werden nur dann ausgeführt, wenn weder die Bedingung des If noch die der Elseif Knoten zutreffen.

Bei der Prüfung des Betriebssystems wird direkt auf eine QF-Test Variable zugegriffen. QF-Test speichert die Betriebssysteminformation in einer Gruppenvariable ab, wobei die Gruppe `qftest` und die Variablen `linux`, `macos` oder `windows` heißen. Die Syntax für den Zugriff auf Gruppenvariablen ist `${group:varname}`, z.B. `${qftest:windows}`.

17.2 Schleifen

QF-Test stellt zwei Knotentypen für die Implementierung von Schleifen zur Verfügung:

- Schleife Knoten führen ihre Kindknoten so oft aus, wie angegeben ist. Man kann die Schleife jedoch über einen Break Knoten jederzeit verlassen.
- While Knoten führen ihre Kindknoten so oft aus, bis die angegebene Bedingung nicht mehr gegeben ist. Derartige Schleifen können ebenfalls über einen Break Knoten jederzeit verlassen werden.

Hinweis

Schleife Knoten enden auf jeden Fall nach der angegeben Anzahl von Wiederholungen. Bei While Knoten muss man jedoch selbst dafür sorgen, dass die Ausführung irgendwann endet, indem die Bedingung falsch wird. Ansonsten kommt es zur Endlosschleife. Im interaktiven Modus können Sie in so einem Fall einfach die Pausetaste drücken. Im Batch-Modus, d.h. wenn Sie QF-Test mit dem Kommandozeilenparameter `-batch` starten um die angegebene Testsuite ohne die QF-Test Benutzeroberfläche auszuführen, müssen Sie dann jedoch den QF-Test Prozess "abschießen".

In der folgenden Übung wollen wir einen Testfall implementieren, der prüft, ob eine bestimmte Zeile in der Tabelle des CarConfig Demos angezeigt wird.

Die im Testfall durchgeführten Aktionen sind:

- Anzahl Tabellenzeilen bestimmen.
- Über alle Zeilen iterieren und prüfen, ob die Zeile passt.
- Wenn die Zeile gefunden wurde, die Schleife abbrechen.
- Falls die Zeile nicht gefunden wurde, einen Fehler ins Protokoll schreiben.

Bitte beginnen Sie mit der Aufnahme eines Checks auf die zu suchende Zeile:

Aktion

- **Aktivieren Sie den Check-Aufnahmemodus** über "Checks aufnehmen" ✓ .
- **Führen Sie einen Rechtsklick auf eine Tabellenzeile** im CarConfig Demo aus und wählen Sie den Menüpunkt Zeile aus dem Popup-Menü.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .
- **Ändern Sie den Namen der aufgenommenen Sequenz** z.B. in `Zeile prüfen`.
- **Wandeln Sie die Sequenz in einen Testfall um:** Rechtsklick auf den Sequenz Knoten und Auswahl des Untermenüpunkts Knoten konvertieren in→Testfall im Popup-Menü.

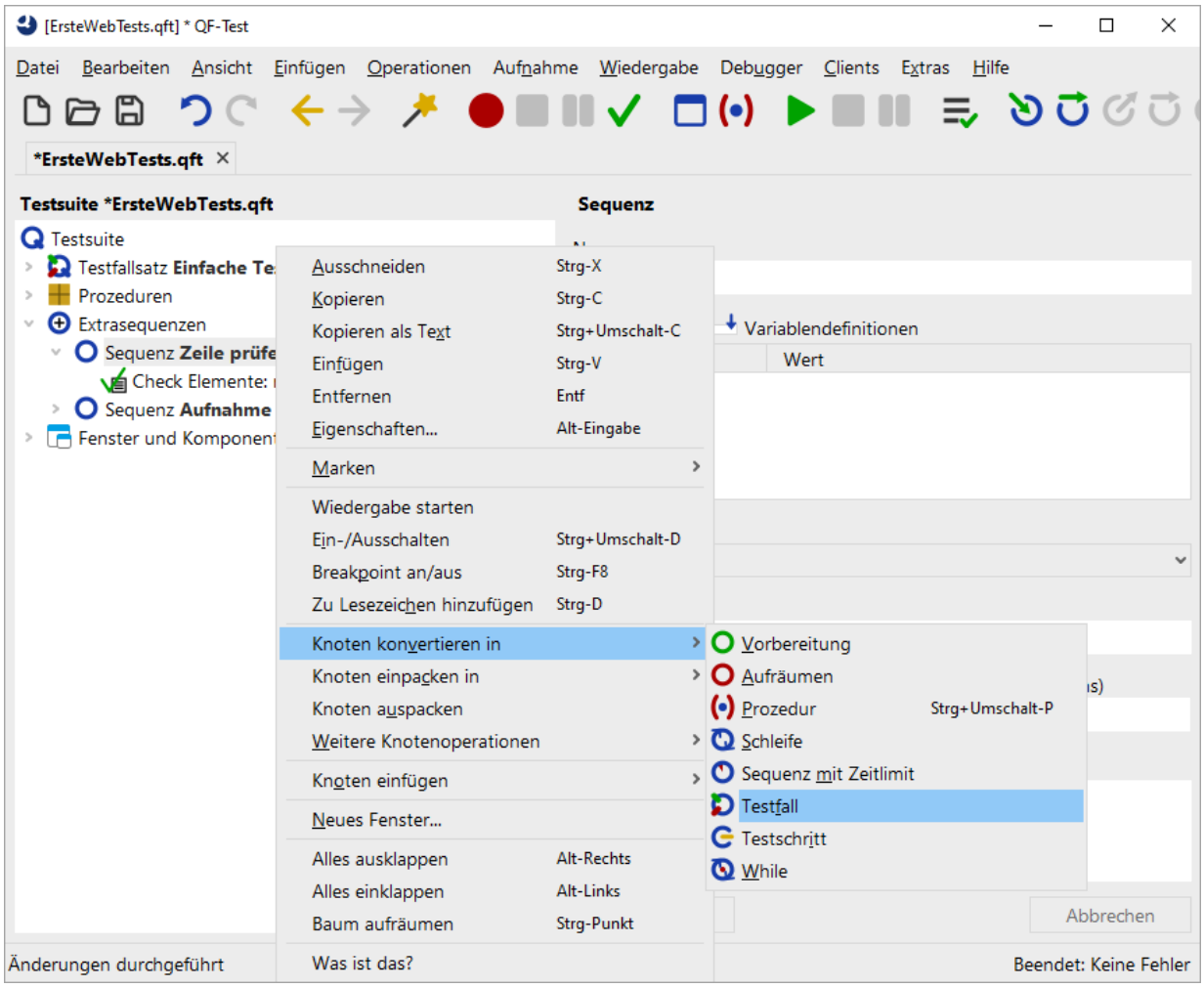

Abbildung 17.4: Knoten konvertieren

In QF-Test können Sie sehr effizient Knoten hinzufügen, indem Sie einen Knoten in einen anderen einpacken:

Aktion

- Öffnen Sie den Testfall Knoten und **packen Sie den aufgenommenen Check Knoten in eine Schleife** indem Sie rechts auf den Knoten klicken und in dem sich öffnenden Popupmenü den Punkt Knoten einpacken in→Schleife auswählen.

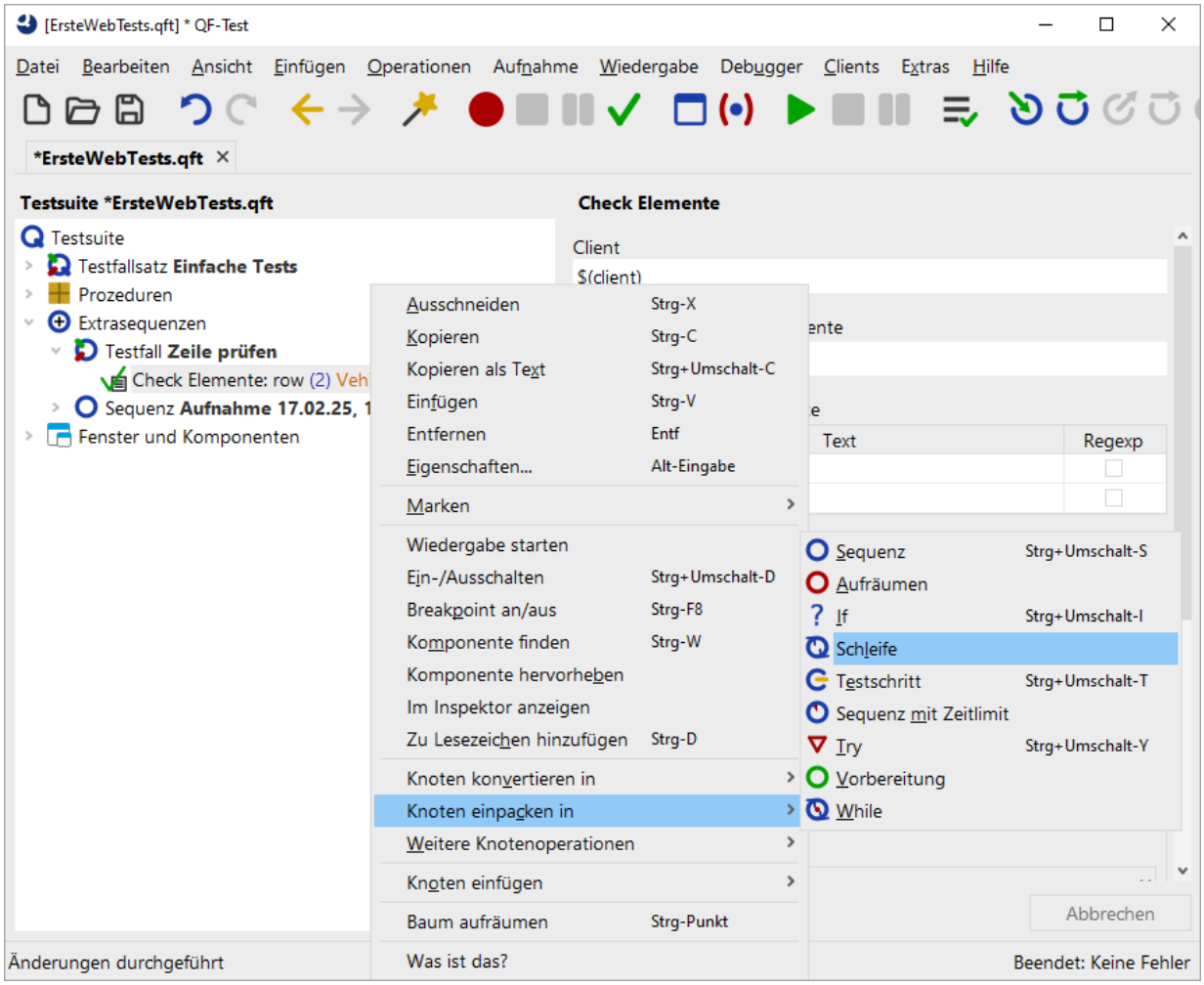

Abbildung 17.5: Knoten einpacken

QF-Test ermittelt dynamisch, in welche Knoten ein Knoten eingepackt werden kann und bietet nur diese zur Auswahl an. Entsprechend kann es passieren, dass Sie ”Schleife” im Untermenü nicht finden. Sie sollten dann prüfen, ob Sie den Rechtsklick auf den richtigen Knoten ausgeführt haben. Dasselbe gilt für die Operationen ”Knoten konvertieren in” und ”Knoten einfügen”.

Als nächstes setzen wir den Wert für das Attribut 'Anzahl Wiederholungen' des Schleife Knoten. Dazu müssen wir bestimmen, wie viele Zeilen die Tabelle hat. Es gibt keinen Knoten, der diese Operation direkt ausführen kann. Allerdings gibt es eine derartige Prozedur in der im letzten Kapitel besprochenen Standardbibliothek. Diese befindet sich im Package `qfs.web.table` und heißt `getRowCount`.

Aktion

- **Selektieren Sie den Testfall Knoten und drücken Strg-A**.
- **Klicken Sie die Schaltfläche "Prozedur auswählen"** links neben der Überschrift 'Name der Prozedur'.
- **Wählen Sie den Reiter 'qfs.qft'** im 'Prozedur auswählen' Dialog.
- **Navigieren Sie zu 'getRowCount' im Package 'qfs.web.table'**
- **Drücken Sie 'OK'** um die Prozedur auszuwählen.
- **Drücken Sie 'OK'** um den 'Prozeduraufruf' Dialog zu schließen.

Das Hinzufügen einer Prozedur über Strg-A wurde in Manuelle Erstellung von Prozeduren[139] beschrieben. Dort finden Sie auch Screenshots zur Aktion.

Aktion

- **Fügen Sie eine Variable** mit dem Namen `Zeilen` **im Attribut 'Variable für Rückgabewert' ein**.
- **Ändern Sie den Standardwert für `id` in** der Variablendefinitionen-Tabelle auf die QF-Test Komponenten ID der Tabelle **`VehicleTable`**.
- Klicken Sie **OK**.
- **Wählen Sie den 'Schleife'-Knoten**.
- Im Attribut 'Anzahl Wiederholungen' des Schleife Knotens **tragen Sie eine Referenz auf die Variable `$(Zeilen)` ein**.
- **Tragen Sie den Namen der Zählervariable**, z.B. `i`, im entsprechenden Attribut des Schleife Knotens **ein**.
- Klicken Sie **OK**.

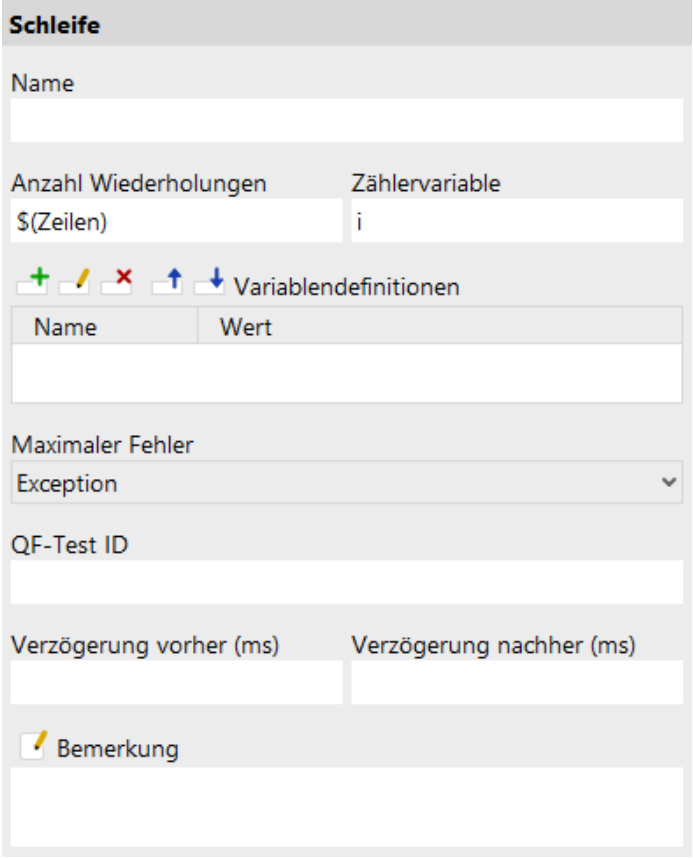

Abbildung 17.6: Details eines Schleife Knotens

In den Details des Check Knotens tragen wir nun in der QF-Test ID der Komponente statt des aufgenommenen Zeilenindex eine Referenz auf die Zählervariable ein und setzen eine Ergebnisvariable. Außerdem fügen wir einen If Knoten unter dem Check Knoten hinzu, der das Ergebnis auswertet und die Schleife über einen Break Knoten verlässt, wenn die entsprechende Zeile gefunden wurde.

Aktion

- **Öffnen Sie den Schleife Knoten**.
- **Selektieren Sie den Check Knoten**.
- **Ändern Sie den aufgenommenen Zeilenindex** der QF-Test ID der Komponente in Zählervariable `$(i)`. Die QF-Test ID der Komponente sollte nun `VehicleTable@Modell&$(i)` lauten.
- **Tragen Sie den Variablennamen `ZeileGefunden` in das Attribut 'Variable für Ergebnis' ein** und klicken **OK**.

- Führen Sie einen **Rechtsklick auf den Check Knoten** aus und wählen Sie aus dem Popup-Menü den Unterpunkt Knoten einfügen→Ablaufsteuerung→Break aus.
- **Drücken Sie 'OK'** im 'Break' Dialog.
- **Packen Sie den Break Knoten in einen If Knoten** mittels des Tastaturkürzels Strg-Umschalt-I (Sie können natürlich auch über das Menü gehen).
- In den Details des 'If'-Knotens **tragen Sie `$(ZeileGefunden)` im Attribut 'Bedingung' ein** und klicken **OK**.

Die Variable `ZeileGefunden` wird vom Check Knoten entweder auf den Wert 'true' oder auf den Wert 'false' gesetzt, so dass wir im Attribut 'Bedingung' des If Knoten nur die Referenz auf die Variable `$(ZeileGefunden)` einzutragen brauchen.

In den nächsten Schritten wollen wir einen Else Knoten als letzten Knoten im Schleife Knoten einfügen. Er wird nur ausgeführt, wenn die Schleife so oft wie angegeben ausgeführt wurde, was in unserem Fall bedeutet, dass die Variable `ZeileGefunden` nie wahr wurde, weil die Zeile nicht gefunden wurde.

Aktion

- **Schließen Sie den If Knoten**, falls dies nicht bereits der Fall ist. Dies ist wichtig, da sonst der Else Knoten zum If Knoten und nicht zum Schleife Knoten gehören würde.
- Führen Sie einen Rechtsklick auf den If Knoten aus und wählen Sie auf dem Popup-Menü den Unterpunkt **Knoten einfügen→Ablaufsteuerung→Else**.
- **Klicken Sie im 'Else' Dialog 'OK'**.
- **Öffnen Sie den Else Knoten**.
- **Fügen Sie** aus der Standardbibliothek **die Prozedur `logError`** aus dem Package `qfs.run-log` wie oben beschrieben **ein**.
- In der 'Variablendefinitionen' Tabelle **tragen Sie `Zeile nicht gefunden`** als Wert der Zeile `message` **ein.**
- **Tragen Sie `true`** als Wert der Zeile `withScreenshots` **ein.**
- Drücken Sie **OK**.

Wenn Sie die Tests im Batch-Modus ausführen, sind Screenshots eine gute Unterstützung bei der Fehleranalyse. Da aber eine große Zahl Screenshots sehr große Protokolldateien erzeugen würden, ist der Standardwert für `withScreenshots false`.

Nun bleibt nur noch, den Testfall mit Vorbereitung und Aufräumen Knoten zu vervollständigen und ihn in den oberen Teil der Testsuite zu verschieben.

Aktion

- **Kopieren Sie die Vorbereitung und Aufräumen Knoten** aus 'Testset: Einfache Tests' in den neuen Testfall als ersten und letzten Knoten.
- **Verschieben Sie den Testfall** aus dem Bereich Extrasequenzen in den oberen Bereich der Testsuite hinter den Knoten 'Testset: Einfache Tests'.

Damit würde der neue Testfall wie folgt aussehen:

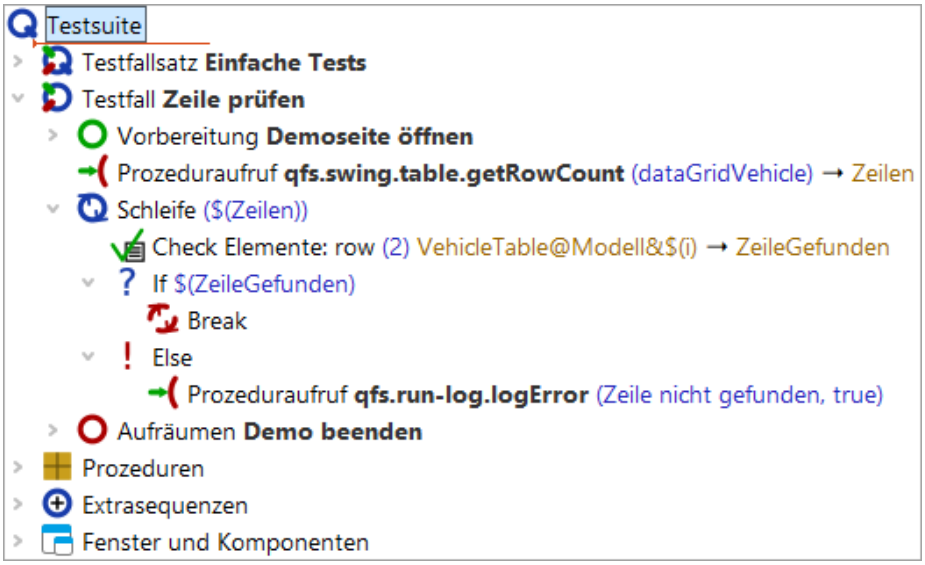

Abbildung 17.7: Der neue Testfall

Aktion

- **Führen Sie den neuen Testfall aus.**

Er sollte ohne Fehler laufen.

Aktion

- **Ändern Sie** nun in den Details des Check Elemente Knotens **den Namen des Fahrzeugs zum Beispiel auf `Falscher Wert`**.

Check Elemente

Client
$(client)

QF-Test ID der Komponente
VehicleTable@Modell&$(i)

Elemente

	Text	Regexp
0	Falscher Wert	☐
1	15.000,00 €	☐

Name des Check-Typs
row

Wartezeit (ms)

Ergebnisbehandlung
Variable für Ergebnis
ZeileGefunden

☐ Lokale Variable

Fehlerstufe der Meldung
Fehler

$ ☐ Im Fehlerfall Exception werfen

Name

QF-Test ID

Verzögerung vorher (ms) Verzögerung nachher (ms)

Bemerkung

Abbildung 17.8: Details eines Check Elemente Knoten

Aktion

- **Führen Sie den Testfall nochmals aus.**

Nun sollte der Test den Else Knoten der Schleife ausführen und eine Fehlermeldung anzeigen.

Kapitel 18

Nun ist es Zeit, Ihre eigene Anwendung zu starten (Web)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Nun ist es Zeit, Ihre eigene Anwendung zu starten'
https://www.qftest.com/de/yt/tutorial-9.html

Nachdem wir so viel Zeit mit all den Beispielanwendungen verbracht haben, sind Sie nun wirklich bereit, Ihre eigene Applikation zu starten (falls Sie dies nicht schon zwischendurch getan haben).

Der Schnellstart-Assistent, welcher über das Menü Extras→Schnellstart-Assistent... erreichbar ist, hilft Ihnen bei dieser Aufgabe. Folgen Sie einfach den Schritten innerhalb des Assistenten, um eine passende Startsequenz zu erzeugen. Bitte schauen Sie auch ins Handbuch Kapitel 3 "Schnellstart".

Es ist an der Zeit, das Gelernte in die Tat umzusetzen - kurze Sequenzen von Events und Checks aufzunehmen, Prozeduren zu erzeugen etc., um eine eigene Testbibliothek aufzubauen.

Damit endet der Basisteil in diesem Tutorial.

Teil III

Native Windows GUIs testen mit QF-Test

Dieser Teil III des Tutorials soll Ihnen die Basiseigenschaften und -arbeitsabläufe von QF-Test erläutern. Er fokussiert sich auf das Testen von nativen Windows-Anwendungen.

Wenn Sie Java- oder Web-Anwendungen testen wollen, empfehlen wir Teil I(2) beziehungsweise Teil II(106). Alle Basisteile vermitteln die gleichen Schulungsinhalte, nutzen für die Beispiele jedoch eine jeweils passende Testanwendung.

Im Teil V(305) werden weiterführende Funktionalitäten von QF-Test erklärt, die für Tests sowohl von Java-, Web- und nativen Windows-Anwendungen genutzt werden können.

Kapitel 19

Bearbeiten einer Beispiel-Testsuite (Win)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Bearbeiten einer Beispiel-Testsuite'
https://www.qftest.com/de/yt/tutorial-1.html

In diesem ersten Kapitel werden wir uns die Struktur einer einfachen Testsuite anschauen, die wesentlichen Bestandteile erklären, sie ausführen und das Ergebnis auswerten.

19.1 Laden der Testsuite

Hinweis

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

Nach dem Starten von QF-Test laden Sie bitte unser erstes Beispiel:

Aktion

- Drücken Sie den Knopf , um den Dateiauswahl-Dialog zu öffnen.
- Wechseln Sie in das Unterverzeichnis `qftest-9.0.0/doc/tutorial` Ihrer QF-Test Installation.
- Dort wählen Sie bitte die Datei `ErsteWinTests.qft` aus und öffnen diese.

QF-Test präsentiert Ihnen die Testsuite wie im folgenden Bild dargestellt:

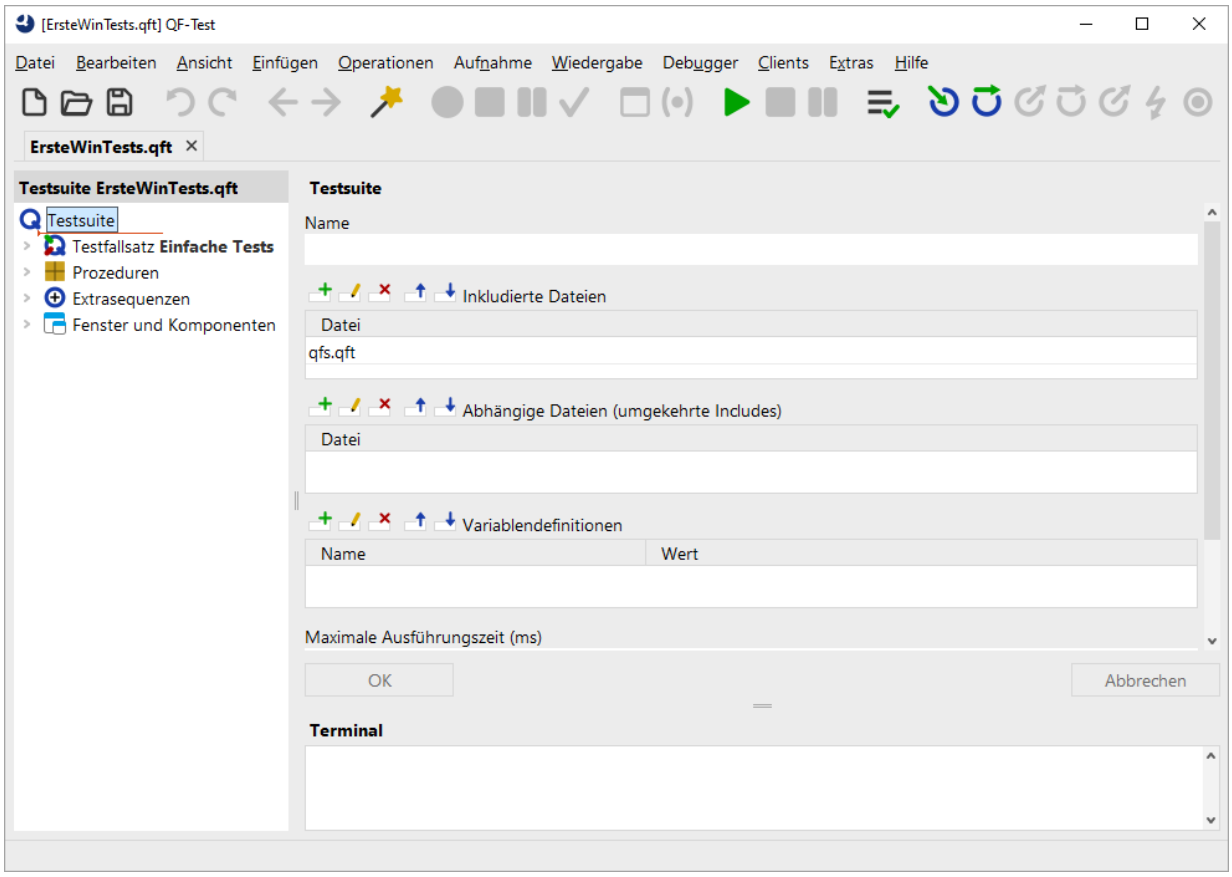

Abbildung 19.1: Das Fenster der Testsuite `ErsteWinTests.qft`

Der **linke Bereich** des Hauptfensters enthält die Testsuite, die in einer Baumstruktur dargestellt wird.
Rechts befindet sich die Detailansicht des Knotens, der im Baum gerade markiert ist. (Falls die Detailansicht bei Ihnen nicht zu sehen sein sollte, aktivieren Sie diese bitte über das Menü Ansicht→Details anzeigen.)
Im Bereich **unten rechts** befindet sich das Terminal, welches die Ausgaben von QF-Test und dem zu testenden Client protokolliert.

Mit Hilfe des Baumes können Sie durch die Testsuite navigieren und einzelne Knoten auswählen, für die dann jeweils die Details im rechten Fensterbereich eingeblendet werden.

Aktion

- **Doppelklicken** Sie bitte den Knoten **Testfallsatz: Einfache Tests** um ihn zu expandieren und die darin liegenden Knoten sehen zu können.

Der Testfallsatz enthält primär zwei Testfälle, umgeben von einem "Vorbereitung"/"Aufräumen" Knotenpaar, das im Wesentlichen die Testanwendung

startet bzw. beendet.

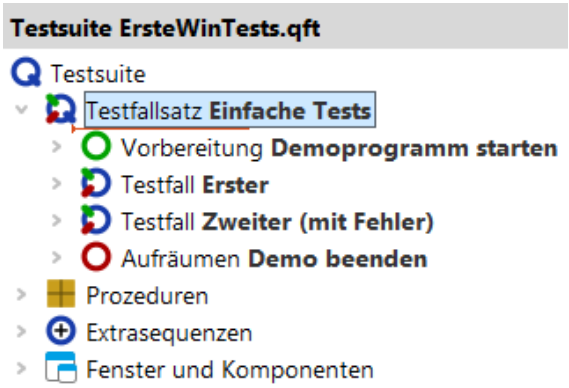

Abbildung 19.2: Der Inhalt des Testfallsatz Knotens

In den folgenden Abschnitten werden wir Funktion und Zweck der einzelnen Knoten erklären.

19.2 Starten der Anwendung

Zuerst wollen wir die Vorbereitung genauer unter die Lupe nehmen:

Aktion

- **Expandieren** Sie den Knoten **Vorbereitung: Demo starten**, wie im folgenden Bild gezeigt.

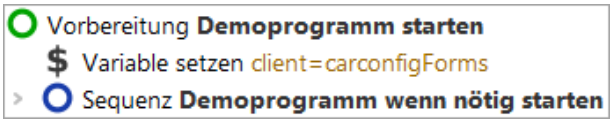

Abbildung 19.3: Der Knoten ”Vorbereitung”

Es werden zwei Kindknoten sichtbar:

1. **Variable setzen** - der Variablen ’client’, wird der Verbindungsname für das zu startende SUT zugewiesen, der für jeden Zugriff auf die Applikation benötigt wird.
2. **Sequenz: Starte Client wenn nötig** - startet das zu testende System (SUT), wenn es nicht schon läuft.

Lassen Sie uns noch einen kurzen Blick in die **Sequenz: Starte Client wenn nötig** werfen:

Abbildung 19.4: Die Sequenz zum Starten des Client

Zu Beginn steht ein **Warten auf Client** Knoten, der prüft, ob der Client bereits läuft. Nur wenn dies nicht der Fall ist, wird er gestartet.

Der **"Windows Client starten"**-Knoten führt die Anwendung aus und stellt die Verbindung zwischen dem Client und QF-Test her. Um unabhängig vom absoluten Verzeichnispfad der Applikation zu sein, verwenden wir einen relativen Pfad, ausgehend vom QF-Test Installationsverzeichnis, dass über die Variable `${qftest:dir.version}` angegeben wird (siehe Handbuchkapitel Variablen).

Wir wollen nun die Anwendung wirklich starten:

Aktion

- **Markieren** Sie dazu bitte den Knoten **Vorbereitung: Demo starten**, doch belassen Sie ihn aufgeklappt.
- **Klicken Sie** den Knopf **Wiedergabe**. Dies führt den aktuellen ausgewählten Knoten aus.

Während der Ausführung wird der gerade aktive Knoten durch "**->**" markiert.

Nach Abschluss der Startsequenz sollte die Demoapplikation "CarConfiguratorNet Form" am Bildschirm erscheinen. Da QF-Test nach Ende der Wiedergabe den Fokus zurückerhält, kann die Demoapplikation dadurch auch wieder verdeckt worden sein.

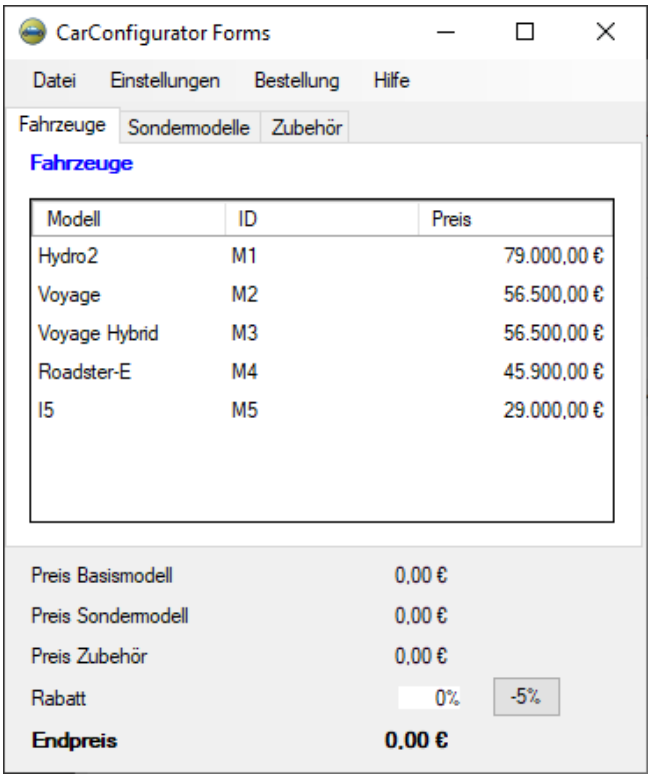

Abbildung 19.5: Das Windows CarConfigurator Demo

19.3 Ein erster Testfall

Als nächstes wollen wir einen Blick auf den ersten Testfall werfen. Er besteht aus vier Testschritten:

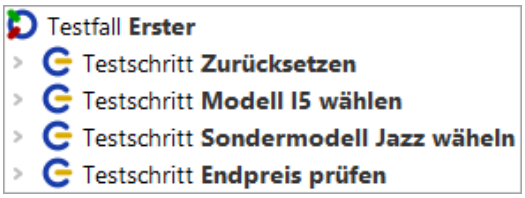

Abbildung 19.6: Der ”Erste” Testfallknoten

1. **Zurücksetzen** - stellt den Anfangszustand der Anwendung über das Menü Datei->Zurücksetzen wieder her und selektiert den Tab Fahrzeuge.
2. **Modell I5 wählen** - Wählt das letzte Modell I5 in der Fahrzeugetabelle aus.

3. **Sondermodell Jazz wählen** - Wechselt zum Tab Sondermodelle und wählt dort Jazz.
4. **Endpreis prüfen** - Überprüft, dass der berechnete Wert dem Feldes Endpreis unten rechts einem vorgegebenen Wert entspricht.

Testschritte sind oft hilfreich, um einen Testfalls zu strukturieren und dadurch lesbar und verständlicher zu gestalten. Dies erleichtert später eine eventuelle Fehlersuche oder Anpassungen des Testfalls.

Aktion

- Bitte **expandieren** Sie die vier **Testschritt Knoten**.

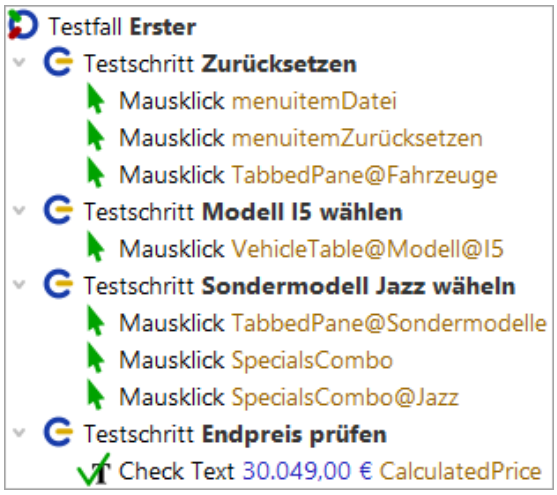

Abbildung 19.7: Die Details des ersten Testfalls

Sie sehen diverse Mausklicks sowie einen Check. Zur besseren Lesbarkeit des Testfalls wurden sie mittels Testschrittknoten strukturiert. Neben der Aktionsart (Mausklick, Check) wird angezeigt, auf welche Anwendungskomponente sich die Aktion bezieht, also wohin z.B. der Mausklick geht. Diese Knoten können direkt über die Aufnahmefunktion von QF-Test erzeugt werden. Näheres hierzu erfahren Sie im nächsten Kapitel Erstellen einer eigenen Testsuite (Java)[19].

Wir wollen uns nun die Ausführung des ersten Testfalls anschauen.

Aktion

- **Markieren** Sie dazu den **Testfall: Erster** Knoten.
- **Drücken Sie** anschließend den Wiedergabeknopf ▶ .

Die Testschritte werden nun der Reihe nach abgespielt.

Das aktuelle Testergebnis wird während und nach dem Testlauf in der Statuszeile am unteren rechten Rand des QF-Test Hauptfensters angezeigt und sollte "Beendet: Keine Fehler" lauten. Daneben zeigt QF-Test verschiedene Zähler an. Der erste Zähler bezieht sich auf die Anzahl der ausgeführten Testfälle, der zweite auf die Zahl der ausgeführten Testfälle ohne Fehler. In unserem Fall wurde ein Testfall fehlerfrei ausgeführt, was einer Erfolgsquote von 100% entspricht.

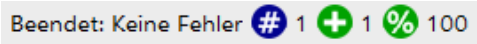

Abbildung 19.8: Die Ergebnisanzeige in der Statusleiste

Wenn Sie den Mauszeiger auf dem Symbol eines Testfallzählers ruhen lassen, wird Ihnen eine entsprechende Beschreibung angezeigt. Eine Auflistung aller Testfallzähler finden Sie im Kapitel Aufnahme und Wiedergabe des Handbuchs.

19.4 Ein zweiter Testfall - mit Fehler

Der zweite Testfall wird uns zeigen, was passiert, wenn ein Fehler bei der Testausführung auftritt.

Aktion

- Bitte **expandieren** Sie den Knoten **Testfall: Zweiter (mit Fehler)**.

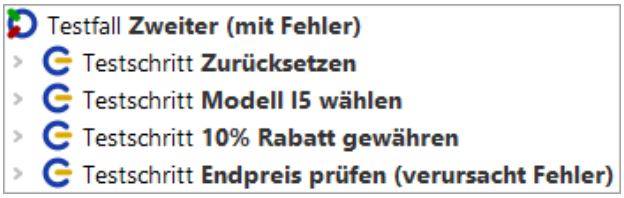

Abbildung 19.9: Der "Zweite" Testfallknoten

Bis auf den dritten Testschritt sieht es bekannt aus. Was tut der Unbekannte?

Testschritt: 10% Rabatt gewähren - Schreibt den Wert 10 in das Rabattfeld

Die Texteingabe ist eine weitere Basisaktion. Eingabe-Knoten kann man ebenfalls direkt über die Aufnahmefunktion generieren lassen. Den Wert 10 sieht man im Feld "Text" rechts und auch direkt im Text des Baumknotens.

Aktion

- **Expandieren** Sie den Knoten **Testschritt: 10% Rabatt gewähren**.

Abbildung 19.10: Die Details des zweiten Testfalls

Wir wollen uns die Ausführung des zweiten Testfalls anschauen.

Aktion

- **Markieren Sie** dazu den **Testfall: Zweiter (mit Fehler)** Knoten.
- **Drücken Sie** anschließend den Wiedergabeknopf ▶ .

Diesmal erscheint ein Dialog mit der Information, dass ein Fehler aufgetreten ist.

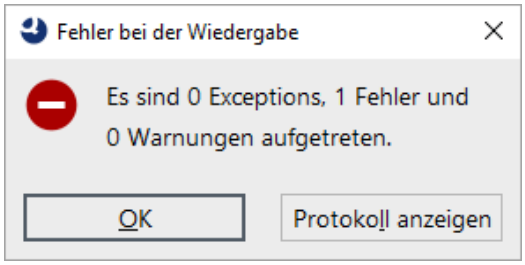

Abbildung 19.11: Fehler im zweiten Testfall

Was ist passiert? Fast immer wenn so ein Fall auftritt, ist es sinnvoll das Protokoll zu Rate zu ziehen.

Alternativ könnte man den Testfall zur Fehlersuche nochmal im Debug-Modus ausführen. Diese Vorgehensweise wird in Kapitel Benutzen des Debuggers (Java)[54] erläutert.

19.5 Das Protokoll zur Fehlerdiagnose

QF-Test protokolliert detaillierte Informationen für jede Testausführung.

Aktion

- **Öffnen** Sie nun bitte das **letzte Protokoll** über eine der folgenden Möglichkeiten:
  - den **Protokoll anzeigen** Knopf im Fehlerdialog

  oder falls Sie den Dialog bereits geschlossen haben

  - den **Button** in der Werkzeugleiste oder
  - über die Tastenkombination **Strg-L**.

Hinweis

Die Protokolle der letzten Testläufe können auch über die unteren Einträge im Menü 'Wiedergabe' aufgerufen werden.

Das Protokoll öffnet sich in einem separaten Fenster und zeigt die protokollierten Aktionen des zweiten Testfalls, den Sie soeben ausgeführt haben:

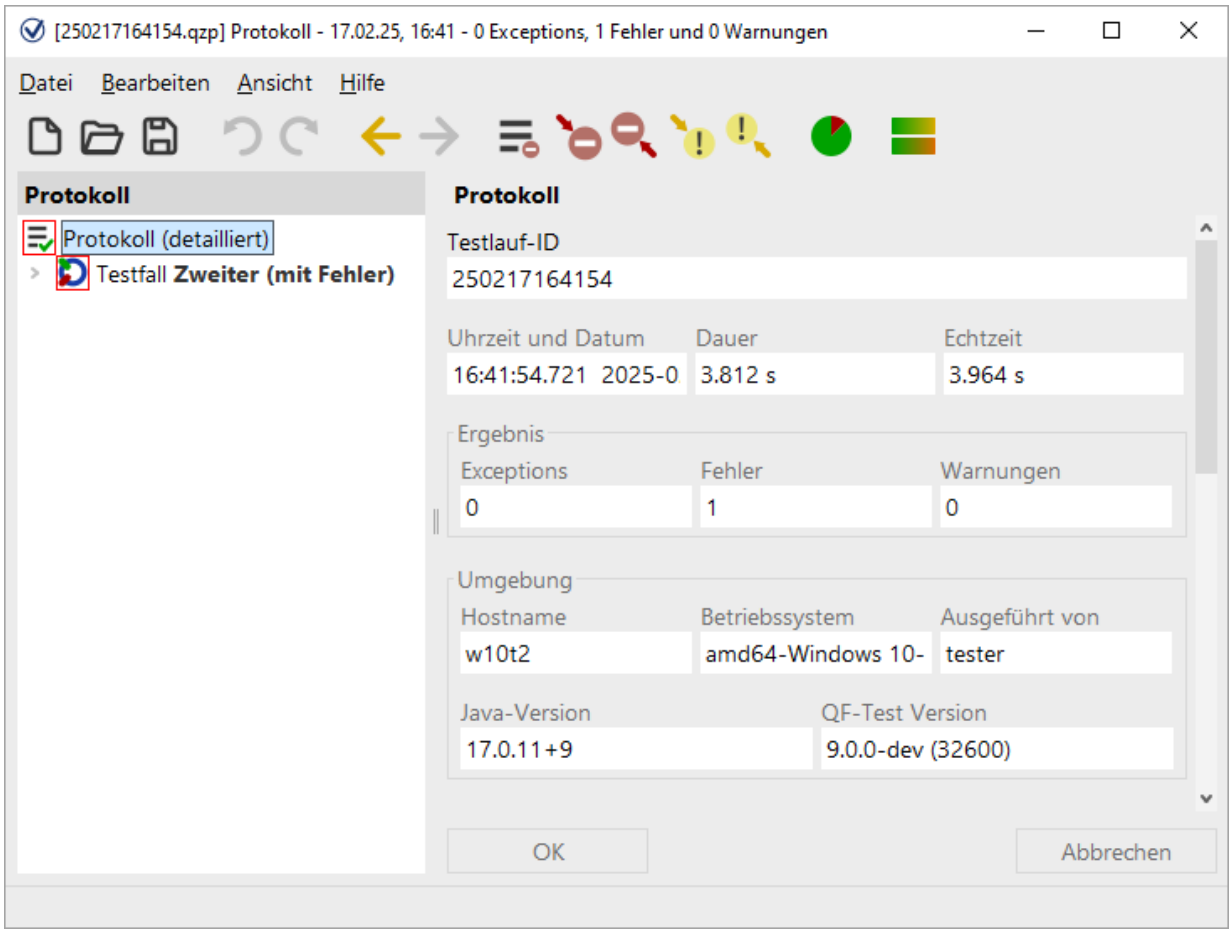

Abbildung 19.12: Protokoll des zweiten Testfalls

Das Protokoll ist in seinem Aufbau ähnlich zu dem der Testsuite. Der Baum links enthält wieder die bekannten Knoten, jedoch dieses Mal in der zeitlichen Abfolge des Testlaufs. Wenn man einen Knoten anwählt, sieht man rechts die Details inklusive Zeitstempel und Ausführungsdauer.

Im Baum links werden Ihnen **rote Rahmen** um einige Knoten auffallen. Diese zeigen an, dass sich darunter Fehler befinden. Wenn man den rot umrandeten Knoten Ebene für Ebene folgt, erreicht man irgendwann den Fehler.

Aktion

- Schneller und bequemer geht es über den Button **Nächsten Fehler finden** in

der Werkzeugleiste oder auch die Tastenkombination **[Strg-N]**.

Alle rot markierten Knoten werden expandiert und der Knoten mit dem eigentlichen Fehler wird selektiert:

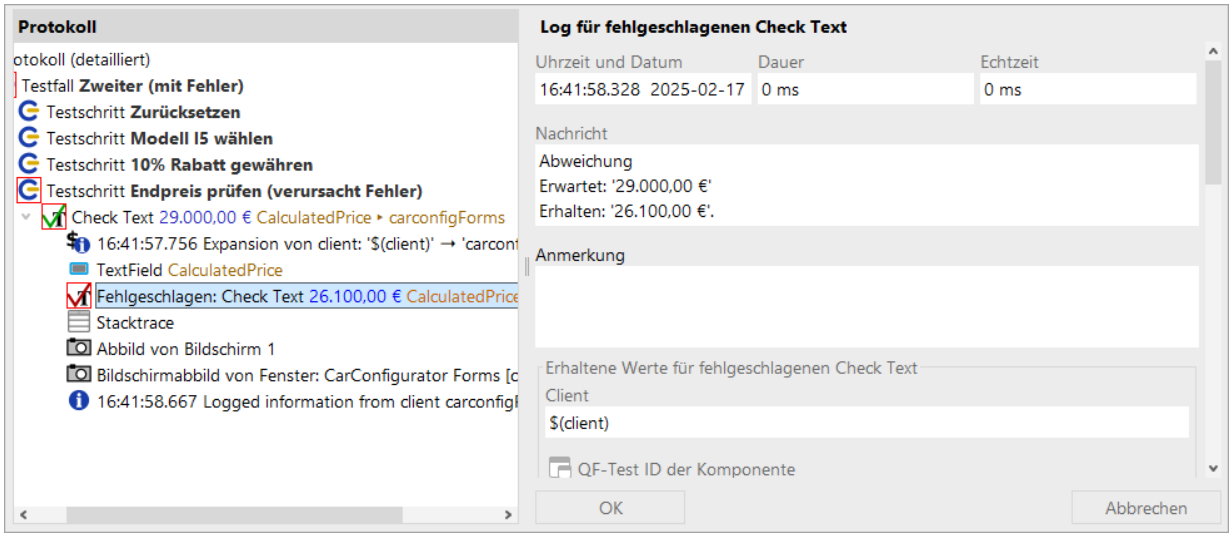

Abbildung 19.13: Fehlerdiagnose für den zweiten Testfall

Die Fehlermeldung auf der rechten Seite gibt an, dass der erhaltene Wert des Endpreis Feldes nicht dem erwarteten entspricht. Dieser Fehler wurde natürlich mit Absicht eingebaut, um zu zeigen, wie man bei der Analyse vorgehen kann.

Hilfreich bei der Fehleranalyse ist üblicherweise auch der übernächste Protokollknoten **Bildschirmabbild**. Seine Detailansicht enthält ein vollständiges Abbild des Bildschirms zum Zeitpunkt des Fehlers. Dies ist sehr nützlich, um den Zustand des SUTs zu sehen und daraus eventuell die Fehlerursache ableiten zu können. Die folgende Grafik zeigt den Knoten:

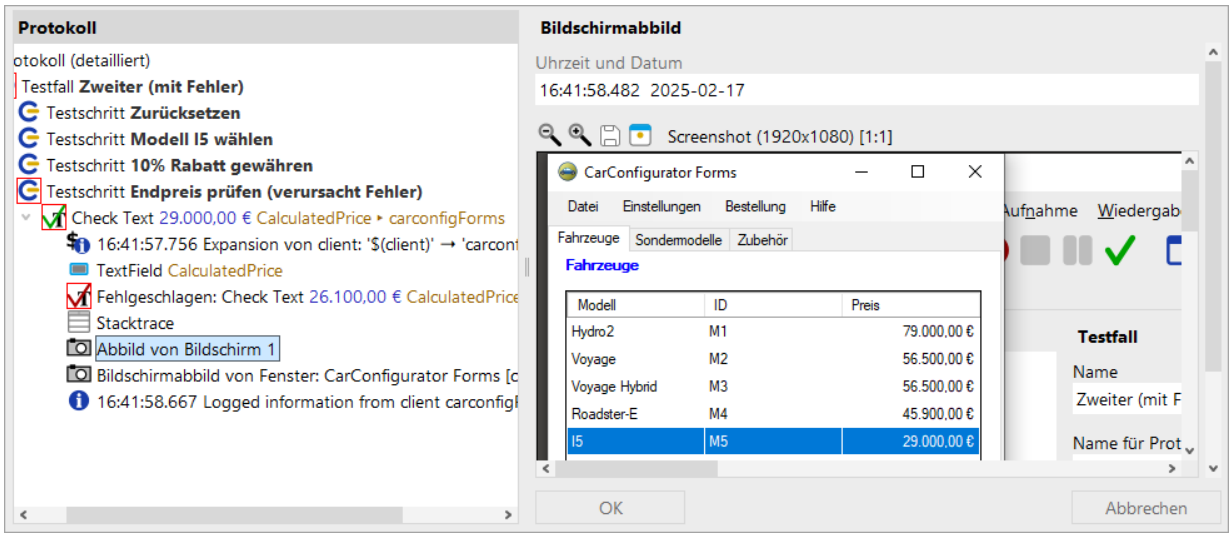

Abbildung 19.14: Knoten mit Bildschirmabbild der Fehlersituation

Neben dem Abbild aller Bildschirme speichert QF-Test auch Bilder der einzelnen Fenster des SUT zum Fehlerzeitpunkt. Dies erlaubt Ihnen deren Inhalt zu analysieren, auch wenn diese eigentlich durch andere Fenster oder Dialoge verdeckt sind.

Hinweis Die in einem längeren Testlauf im Protokoll gesammelten Informationen können große Mengen an Arbeitsspeicher verbrauchen. Deshalb ist QF-Test so voreingestellt, dass es kompakte Protokolle erstellt, wobei nur die für Fehlerdiagnose und Reportgenerierung wichtigen Informationen erhalten bleiben.
Diese Funktion ist mit der Option "Kompakte Protokolle erstellen" über Bearbeiten→Optionen...→Protokoll→Inhalt konfigurierbar. Der Typ eines Protokolls wird in seinem Wurzelknoten angezeigt. Auch die Anzahl der Bildschirmabbilder, die im Protokoll gespeichert werden, ist konfigurierbar.

19.6 Wo finde ich Hilfe?

In diesem Abschnitt machen wir eine kleine Pause, um einige allgemeine Hinweise zu geben.

Es gibt verschiedene Möglichkeiten, um Hilfe oder Antworten zu finden:

Die umfassendste Suche kann man über **Hilfe→Online Suche...** anstoßen. Dies bringt Sie auf die Suchfunktionalität unserer Homepage und erlaubt Ihnen die **Abfrage aller verfügbarer Dokumentation** (Handbuch https://www.qftest.com/qf-test-handbuch.html, Tutorial https://www.qftest.com/qf-test-tutorial.html, Standardbibliothek

https://www.qftest.com/qf-test-support/dokumentation/standardbibliothek.html, Blog https://www.qftest.com/blog.html und unsere Videos https://www.qftest.com/los-gehts-mit-qf-test/videos.html). Die angezeigten Suchergebnisse können **passend gefiltert** werden.

Wenn Sie **offline** arbeiten und nach einem Thema suchen wollen, können Sie die **PDF Versionen von Handbuch und Tutorial** nutzen, die über das **Hilfe** Menü verfügbar sind. Offline HTML Versionen haben keine übergreifende Inhaltssuche. Jedoch gibt es auf jeder HTML Seite in Kopf- und Fußzeile einen Link auf die PDF Version, so dass der Wechsel dorthin einfach möglich ist.

QF-Test bietet eine **kontextsensitive Hilfe** für alle Baumknoten und deren Detailattribute an. Um diese zu nutzen, klicken Sie einfach mit der **rechten Maustaste** auf den gewünschten Knoten oder das Attribut in der Detailansicht. Im Kontextmenü wählen Sie dann den Eintrag **Was ist das?**. Dieser bringt Sie direkt zur passenden Referenzbeschreibung ins Handbuch.

Neben der Hilfestellung in der Dokumentation haben Sie auch die Möglichkeit unser Support-Team zu kontaktieren. Während Ihrer Evaluationsphase und anschließend als Kunde mit einem gültigen Pflegevertrag können Sie Ihre Fragen direkt an unsere Supportexperten richten über das Support-Formular im QF-Test Hilfe-Menü Support-Team kontaktieren oder über unsere Webseite.

19.7 Beenden der Anwendung

Wir haben noch nicht die Aufräumsequenz angeschaut und wollen dies nun tun:

Aktion

- **Expandieren** Sie den **Aufräumen: Demo beenden** Knoten.

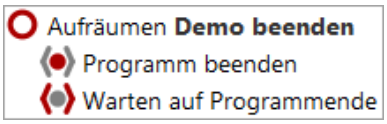

Abbildung 19.15: Die Aufräumsequenz

Unsere Aufräumsequenz stoppt "hart" den Client-Prozess und wartet anschließend, bis sich dieser vollständig beendet hat. Dies ist eine sehr einfache Variante aber für den Moment ausreichend.

Aktion

Führen Sie die **Aufräumsequenz aus** und lassen damit den CarConfigurator verschwinden.

19.8 Ein vollständiger Testlauf

Nachdem wir uns Schritt für Schritt durch den Testfallsatz gearbeitet haben, wollen wir nun alles in einem Rutsch ausführen.

Aktion

- **Schließen Sie bitte das "CarConfigurator Demo"**, falls es noch läuft.
- **Markieren** Sie den **Testfallsatz "Einfache Tests"**.
- Führen Sie ihn aus mittels ▶ .

Der Testlauf endet mit dem bekannten Fehler.

Aktion

- Wenn Sie nun bitte mittels das **Protokoll öffnen**, sehen Sie, wie QF-Test den Test abgearbeitet hat.

Abbildung 19.16: Das Protokoll des gesamten Testfallsatzes

Man sieht, dass die Vorbereitungs- und Aufräumenknoten vor bzw. nach **jedem Testfall** ausgeführt werden. Dies ist eine Eigenschaft, die diese im Zusammenspiel mit einem Testfallsatzknoten entwickeln. Dadurch wird für jeden Testfall immer ein sauberer Ausgangszustand hergestellt.

Hinweis

Das SUT nach jedem Testfall zu beenden ist nicht die eleganteste Art, einen sauberen Ausgangszustand zu erreichen. Elegantere Wege zur Herstellung einer definierten Testausgangssituation und Durchführung der notwendigen Aufräumarbeiten werden in Kapitel (Kapitel 29(316)) dieses Tutorials erklärt.

19.9 Reportgenerierung

Im Qualitätssicherungsprozess ist es wichtig, Testergebnisse zu dokumentieren und auch zu archivieren. QF-Test bietet die Möglichkeit, aus Protokollen Testreports zu generieren. Wir wollen dies für das gerade aufgezeichnete Protokoll beispielhaft durchführen.

Aktion

- **Öffnen** Sie bitte das **Protokoll** und
- **wählen** im Menü **Datei→Report erstellen...**.

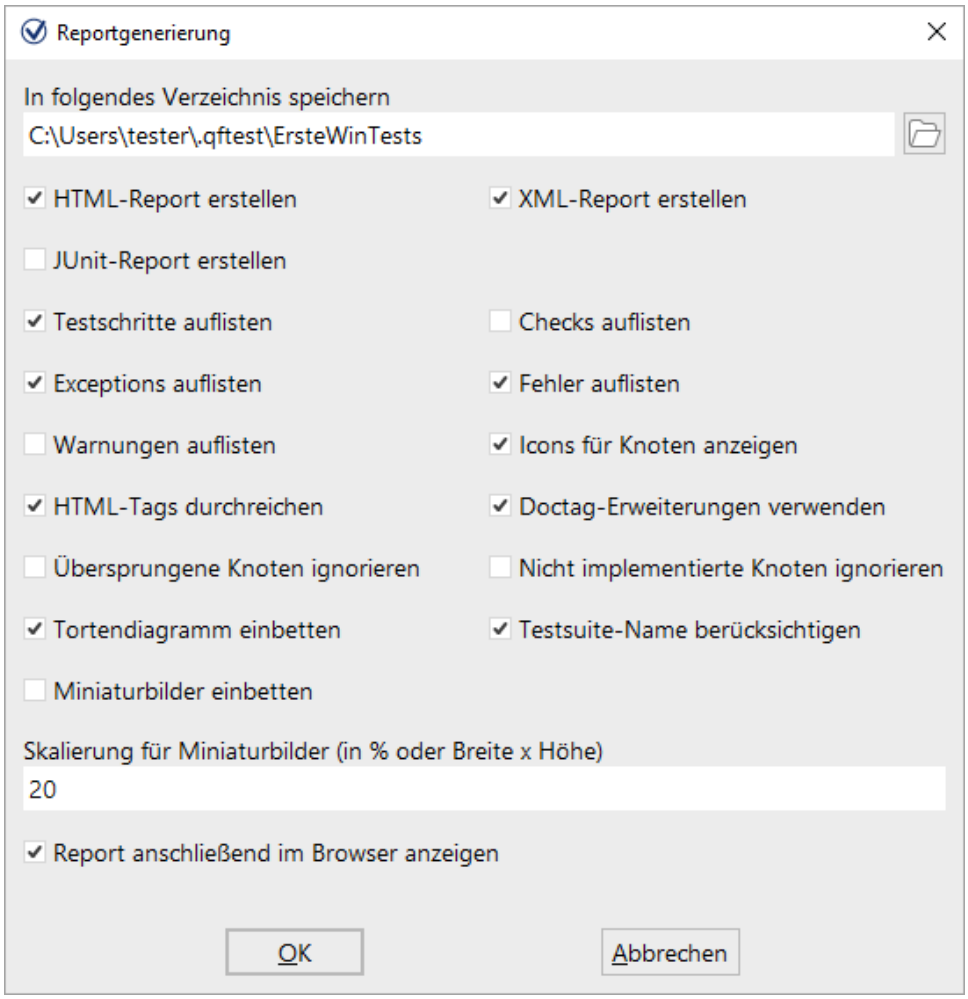

Abbildung 19.17: Auswahldialog für die Reportgenerierung

Im ersten Feld können Sie den Dateinamen des Reports festlegen. QF-Test bietet drei Arten von Reports - HTML, XML und JUnit Format. Das XML Format können Sie verwenden, wenn Sie die Reports zum Beispiel mit Hilfe eigener XSLT Stylesheets selbst gestalten wollen. JUnit-Reports erweisen sich als hilfreich, wenn es darum geht, Resultate in Build- oder Testmanagement-Tools zu importieren.

Wir wollen uns nun einen einfachen HTML Report zu unserem letzten Testlauf erzeugen lassen.

Aktion

- Lassen Sie bitte die vorgegebenen **Optionen unverändert**.

- Bestätigen Sie den Reportdialog mit **OK**.

Anschließend sollte sich Ihr Browser automatisch mit einem Ergebnis äquivalent zum folgenden Bild öffnen:

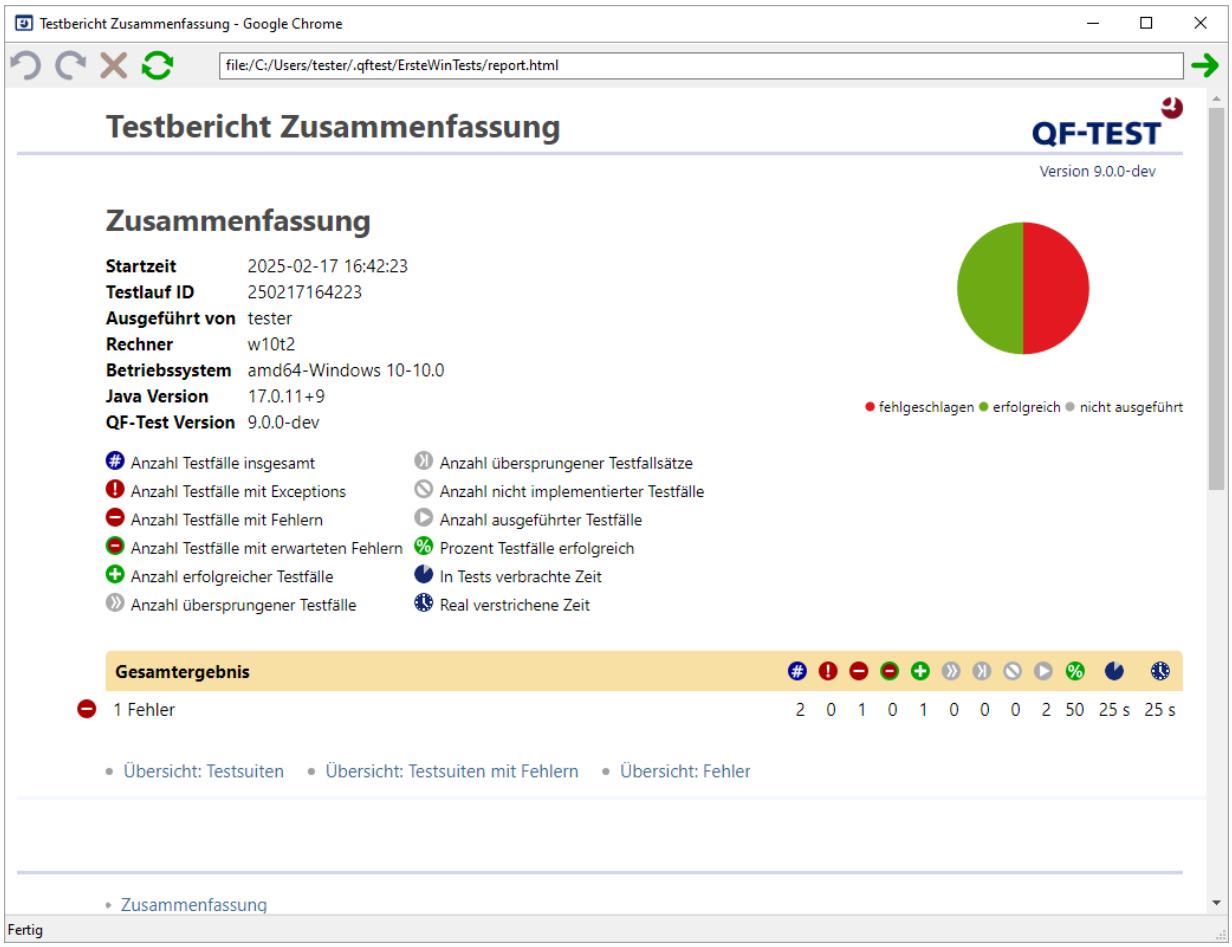

Abbildung 19.18: Ein HTML Report

Der Testbericht beginnt mit einer Zusammenfassung mit allgemeinen Systeminformationen im linken Bereich, einer Legende der verwendeten Symbole rechts, einem Überblicks-Tortendiagramm in der Mitte und dem Gesamtergebnis darunter. In unserem Fall bei einem fehlerhaften von zwei ausgeführten Testfällen eine Erfolgsquote von 50%.

Auf die Zusammenfassung folgen drei Übersichten:

1. Testsuiten, die in diesem Testlauf ausgeführt wurden.
2. Testsuiten, in denen Fehler aufgetreten sind.

3. Fehler, mit Ihrem genauen Ort und Fehlermeldung

Die Reporterstellung in QF-Test ist ein praktisches Hilfsmittel, um einen Überblick über einen Testlauf zu gewinnen und ein Dokument zu Präsentations- und Archivierungszwecken zu erstellen.

Kapitel 20

Erstellen einer eigenen Testsuite (Win)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Erstellen einer eigenen Testsuite'
https://www.qftest.com/de/yt/tutorial-2.html

In diesem zweiten Kapitel des Windows Tutorials werden wir selbst Sequenzen zum Starten und Beenden des SUT erstellen. Zusätzlich wollen wir Aktionen und Checks aufnehmen und damit einen einfachen Testfall aufbauen.

20.1 Starten der Anwendung

Zu Beginn muss die zu testende Anwendung aus QF-Test heraus gestartet werden. Es gibt einen **Schnellstart-Assistenten**, der uns hilft, eine passende Startsequenz zu erzeugen.

Aktion

- Öffnen Sie bitte eine neue, leere Testsuite mittels Datei→Neue Testsuite...
- Öffnen Sie den Schnellstart-Assistenten über das **Menü** **Extras→Schnellstart-Assistent...**.

Der Assistent startet mit einem Willkommen und allgemeinen Informationen.

Aktion

- Nach einem kurzen Hallo drücken Sie bitte den **"Weiter" Knopf**.

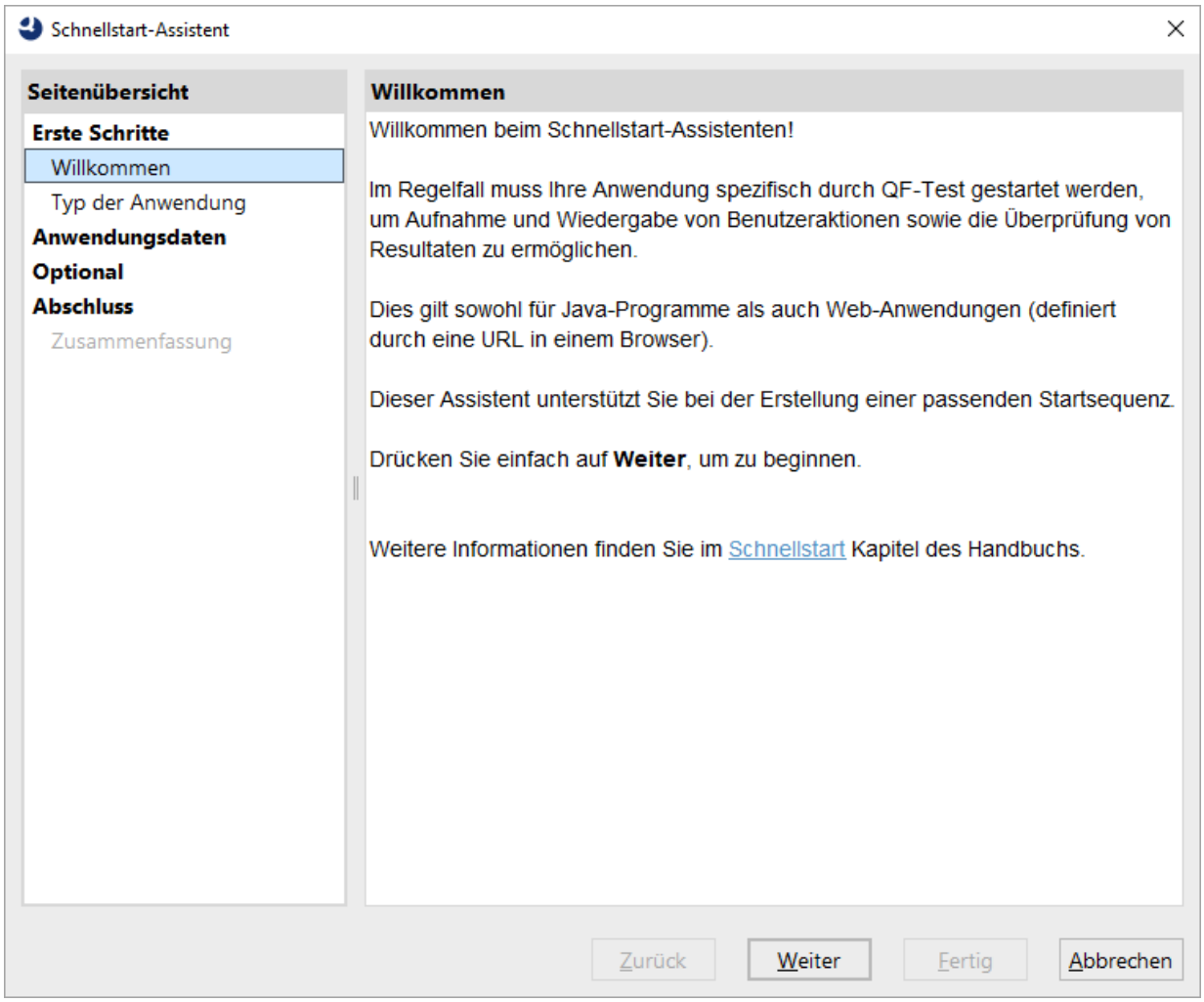

Abbildung 20.1: Der Schnellstart-Assistent

Im zweiten Schritt werden Sie dazu aufgefordert, die Art der zu testenden Applikation auszuwählen.

Aktion

- Wählen Sie bitte die vierte Option **Eine native Windows-Anwendung**.
- Drücken Sie **Weiter**.

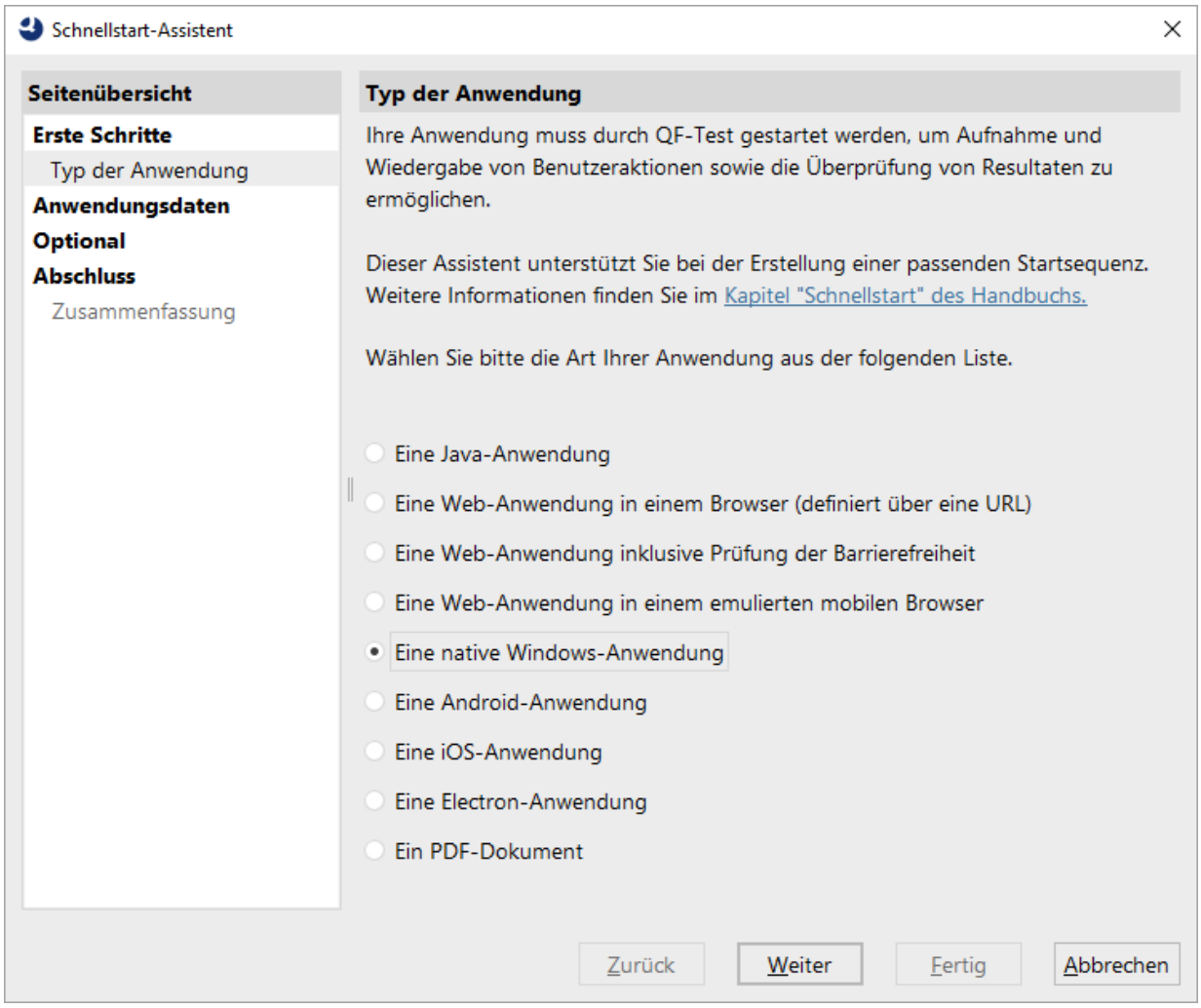

Abbildung 20.2: Auswählen der SUT Art

Nun werden Sie nach dem Windows Programm gefragt.

Aktion

- Nutzen Sie hierzu den **Programm auswählen** Knopf auf der rechten Seite.
- Wechseln Sie in das **Verzeichnis** `.../qftest-9.0.0/demo/carconfigForms/` in Ihrer QF-Test Installation.
- **Wählen** Sie dort die Datei `CarConfigForms.exe`.

Das zweite Feld kann in unserem Fall leer bleiben. Jedoch soll erwähnt werden, dass es auch möglich ist QF-Test mit einer bereits laufenden Windows-Anwendung zu verbinden. Dies geschieht durch Angabe des Fenstertitels im besagten zweiten Feld. Auch reguläre Ausdrücke für Fenstertitel sind hier möglich.

Hinweis Im Bild sieht man eine weitere Möglichkeit: Die Verwendung der Variablen `${qftest:dir.version}` am Beginn, die automatisch zum versionsspezifischen Installationsverzeichnis von QF-Test expandiert. Details zu speziellen QF-Test Variablen finden Sie im Handbuch Kapitel Variablen.

Aktion

- Drücken Sie den **Fertig** Knopf, da wir die weiteren optionalen Schritte für unser einfaches Demo nicht benötigen.

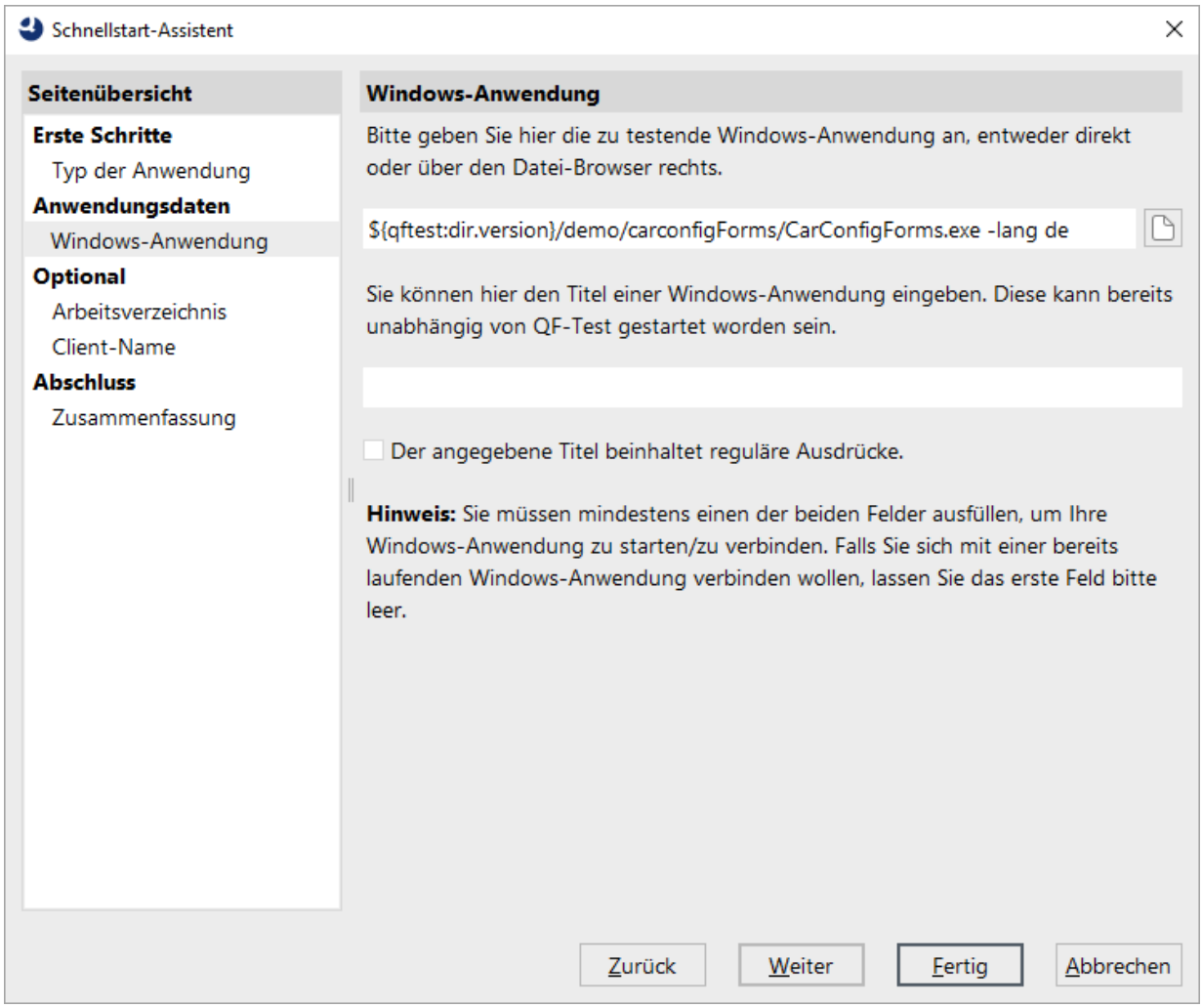

Abbildung 20.3: Auswahl der Programm Datei

Wir gelangen direkt zur Zusammenfassung, die beschreibt, wie es nach dem Beenden des Schnellstart-Assistenten weiter geht.

Aktion

- Drücken Sie den **Fertig** Knopf, um den Assistenten zu beenden.

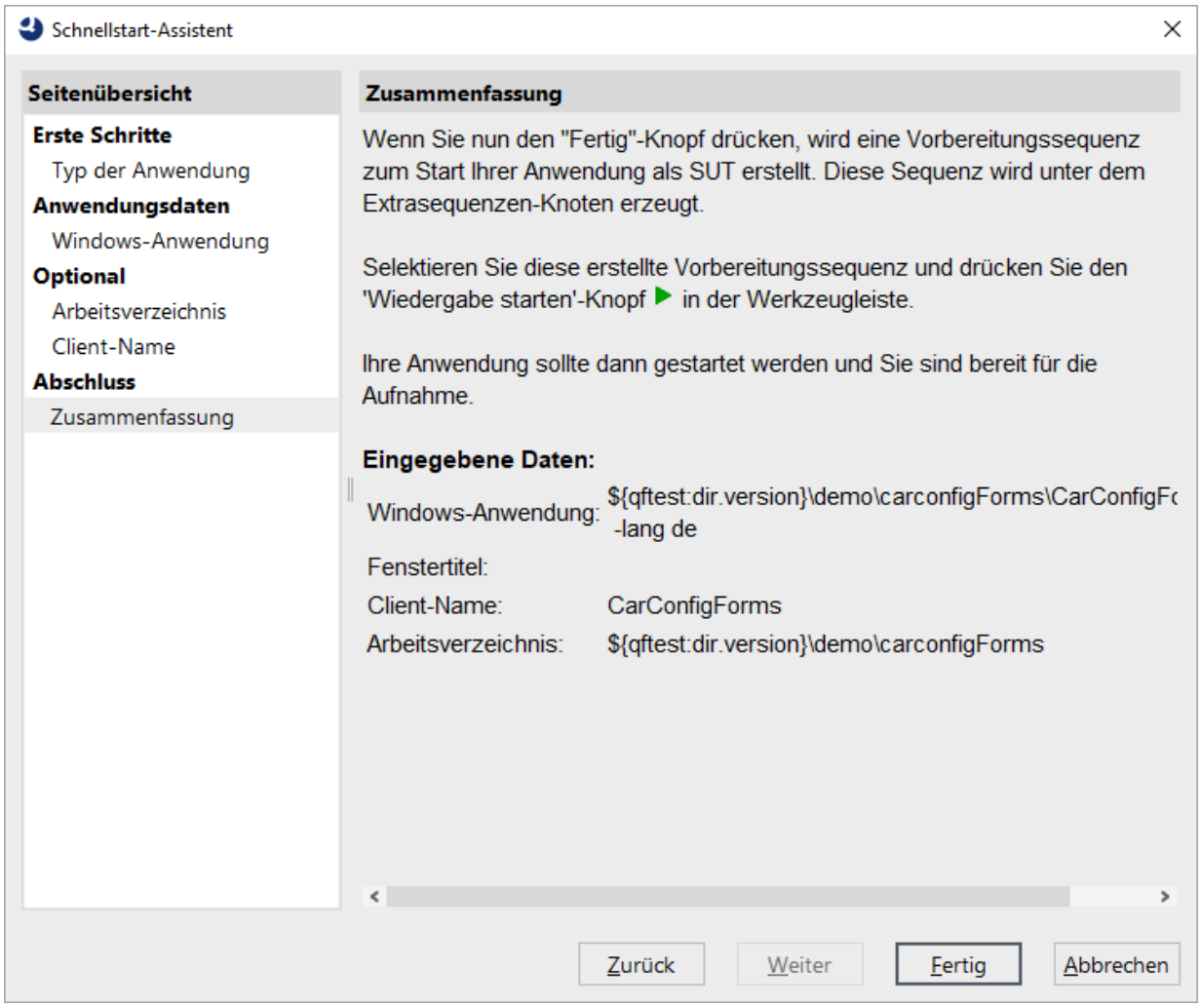

Abbildung 20.4: Zusammenfassung

Die generierte Startsequenz **Starte CarConfigForms** erscheint in den ”Extrasequenzen” der Testsuite und enthält drei Schritte:

- **Variable setzen** - definiert die globale Client Variable, die durchweg in der Testsuite benutzt wird.
- **Warten auf Client** - prüft, ob der Client bereits läuft.
- **SUT starten, wenn notwendig** - falls der Client noch nicht läuft, wird er mit Hilfe eines Windows-Anwendung starten Knotens als zu testendes System (SUT) gestartet und gewartet bis dies erfolgt ist.

Hinweis Die Information, ob der Client bereits läuft wird im ersten Warten auf Client-Knoten einer

Variable ”isSUTRunning” gespeichert und in der folgenden ”If”-Bedingung ausgewertet. Sie können dies in den entsprechenden Knotendetails sehen. Diese Art der bedingten Ausführung wird später noch im Detail erklärt.

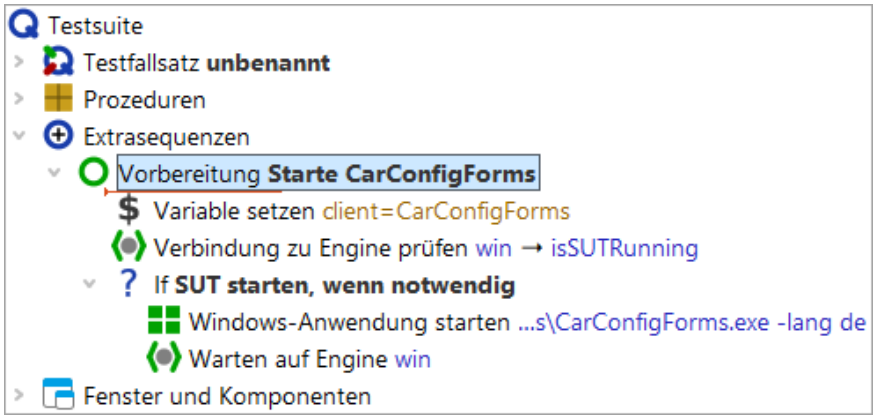

Abbildung 20.5: Generierte Startsequenz

Probieren wir sie aus:

Aktion

- Stellen Sie bitte sicher, dass der Knoten **Vorbereitung: Starte CarConfigForms** ausgewählt ist.
- **Drücken Sie** ▶ oder betätigen Sie einfach die Eingabe Taste.

Im folgenden Bild ist das Fenster des SUT-Client dargestellt, das nun erscheinen sollte. Da nach dem Abspielen der Fokus zurück zu QF-Test wandert, kann es sein dass das Fenster der Testsuite die Demoanwendung dann verdeckt.

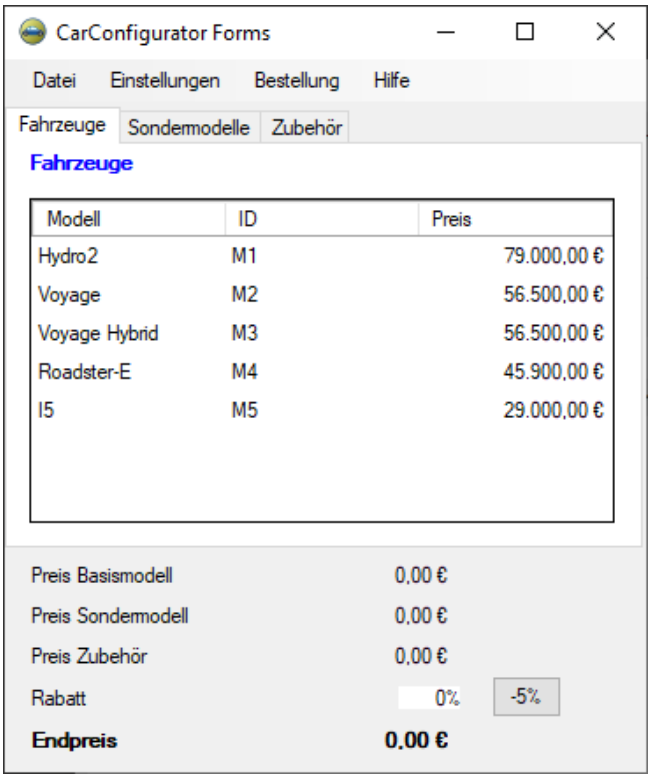

Abbildung 20.6: Das Fenster des ”CarConfigurator”

Am Ende dieses Abschnitts wollen wir unsere Testsuite speichern.

Aktion

- Drücken Sie den Knopf in der Werkzeugleiste oder nutzen Sie die Datei→Speichern Menüaktion bzw. das Tastenkürzel Strg-S.
- Im Datei-Explorer navigieren Sie in ein passendes Verzeichnis, in dem Sie Schreibrechte besitzen, z.B. `Dokumente` in Ihrem Benutzerverzeichnis.
- Geben Sie einen Namen ein z.B. `MeineErstenTests.qft`.
- Beenden Sie die Speicheraktion über den Speichern-Knopf.

20.2 Aufnehmen von Aktionen

Wir werden nun erste Aktionen in unserem Demo aufnehmen:

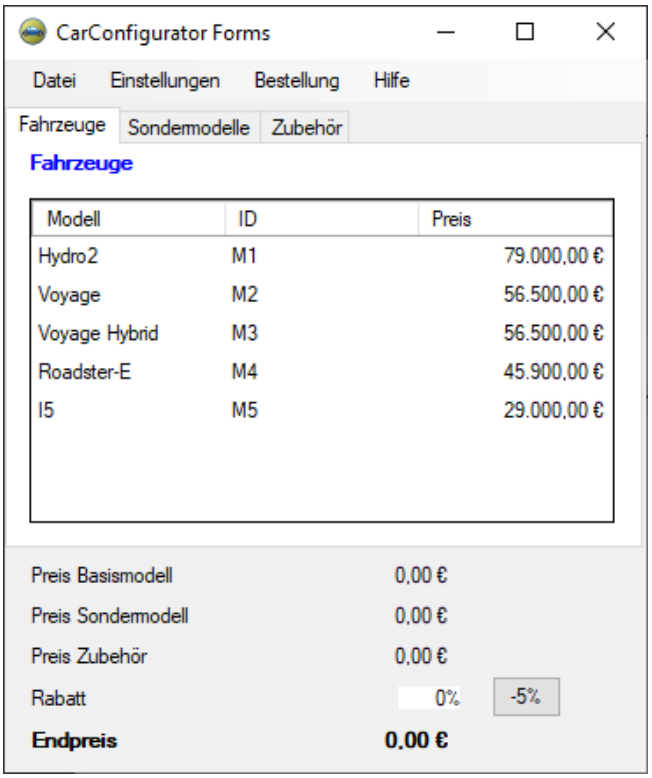

Abbildung 20.7: Aktionen im CarConfigurator Demo aufnehmen

Aktion

- Drücken Sie dazu den ● **Aufnahmeknopf**
- **Wechseln** Sie zum SUT Anwendungsfenster. Von jetzt ab wird jede Maus- oder Tastaturaktion aufgenommen. Es gibt jedoch einen Indikator in Form eines roten Quadrats unter dem Mauszeiger. Erst wenn diese verschwindet, ist QF-Test bereit für die Aufnahme der Aktion auf die Komponente.
- Wählen Sie mit der Maus das Modell **I5** ganz unten in der Tabelle aus.
- Wechseln Sie zum Tab **Sondermodelle**.
- Wählen Sie dort das Sondermodell **Jazz** über das Dropdown-Menü.
- Zum Schluss klicken Sie wieder auf den ersten Tab **Fahrzeuge**.
- **Beenden Sie die Aufnahme**, indem Sie zurück zum QF-Test Fenster wechseln und dort den Knopf für "Aufnahme beenden" ■ drücken.

Hinweis

Ein rotes Quadrat unter dem Mauszeiger zeigt durch sein Verschwinden an, dass nun Aktionen auf die Komponente aufgenommen werden können.

Sie finden die aufgenommene Sequenz unter dem **"Extrasequenzen"** Knoten, wie im folgenden Bild dargestellt.

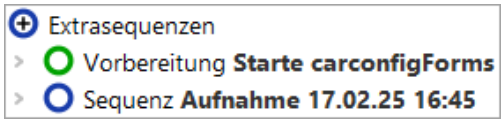

Abbildung 20.8: Der Baum nach Aufnahme der Sequenz

Als Sequenzname wird standardmäßig Datum und Zeit der Erstellung verwendet. Dieser kann anschließend in den Details rechts beliebig angepasst werden.

Aktion

- **Ändern** Sie den Sequenznamen bitte ab zu "Modell I5 Jazz wählen"
- **Öffnen** Sie die Sequenz um die enthaltenen Knoten zu sehen. Es sollten die erwarteten Mausklicks sein. Sie sollten sogar in der Lage sein, die angesprochenen Komponenten zuordnen zu können.

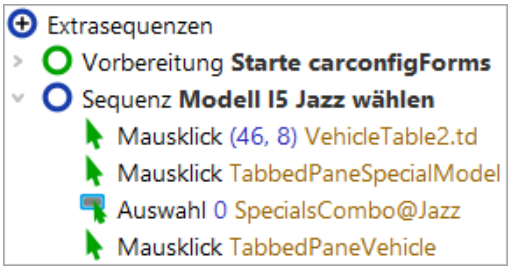

Abbildung 20.9: Die umbenannte Sequenz

Wir wollen nun die aufgenommene Sequenz abspielen.

Aktion

- Markieren die Sequenz **Modell I5 Jazz wählen**.
- Drücken Sie ▶ **Wiedergabe**.

Sie sollten die exakt gleichen Aktionen sehen, die Sie zuvor aufgenommen haben.

Den aufgenommenen Ablauf sollten Sie auch wiederholt ohne Fehler abspielen können. Rechts unten im Fenster der Testsuite sollte "Beendet: Keine Fehler" zu sehen sein.

20.3 Aufnahme von Checks

Um das Verhalten des Clients zu überprüfen, verwenden wir Check-Knoten, mit denen man Zustand und Eigenschaften von Elementen abfragen kann. Auch Checks können aufgezeichnet werden.

Aktion

- Zum Aufnehmen eines Checks drücken Sie die den ✓ **"Check aufnehmen"** Knopf.
- Wechseln Sie zum Fenster des SUT. Es erscheint ein Rahmen um die Komponente, über der sich der Mauszeiger befindet.
- Bewegen Sie den Mauszeiger über das Wertfeld des **Endpreises** und warten Sie bis der Check-Rahmen erscheint.
- Klicken Sie nun mit der **rechten Maustaste**. Das erscheinende Kontextmenü erlaubt Ihnen die Auswahl eines Checks. Die Liste der verfügbaren Checks hängt vom Typ der Komponente ab.
- Wählen Sie den **ersten Eintrag "Text"**, um den textuellen Wert des Feldes zu überprüfen.
- Beenden Sie die Aufnahme durch Drücken des **Stoppknopfs** ■ .

Hinweis

Warten Sie das Erscheinen des Check-Rahmens bevor Sie klicken, um den Check aufzunehmen. Anderen falls kann es vorkommen, dass der Check in ordentlich aufgezeichnet wird.

Wieder taucht die neue Aufnahme unter den "Extrasequenzen" auf.

Aktion

- **Benennen** Sie den Sequenzknoten um auf den Namen **"Endpreis prüfen"**.
- **Öffnen** Sie anschließend den Sequenzknoten, um den Checkknoten zu sehen.

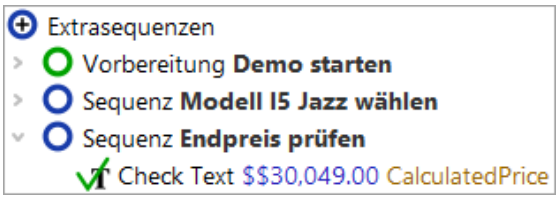

Abbildung 20.10: Die aufgenommene Check-Sequenz

In den Details des "Check Text" Knotens sieht man ebenfalls den erwarteten Wert des Endpreis Feldes.

Aktion

- Auch diese Sequenz können Sie wieder **selbst ausführen**, um die Wiedergabe zu testen.

Im nächsten Schritt wollen wir aus den beiden Sequenzen einen richtigen Testfall aufbauen.

20.4 Erstellen einer Testsuite

Die Basisstruktur unterhalb des Wurzelknotens einer Testsuite ist durch folgende Knoten festgelegt:

- Eine beliebige Anzahl von ”Testfallsatz” und ”Testfall” Knoten, um funktionale Tests zu spezifizieren und zu strukturieren.
- ”Prozeduren” - hier können wiederverwertbare Sequenzen in Prozeduren organisiert werden
- ”Extrasequenzen” - unsere Spielwiese für Aufnahmen etc.
- ”Fenster und Komponenten” - das eigentliche Herz der Testsuite. Hier sind alle aufgenommenen Fenster und Komponenten des SUT mit ihren Eigenschaften enthalten

Funktionale Testfälle werden durch ”Testfall” Knoten repräsentiert und mittels ”Testfallsatz” Knoten gruppiert bzw. strukturiert.

”Vorbereitung” und ”Aufräumen” Knoten können Aktionen enthalten, um einen wohldefinierten Zustand vor und nach einem Testfall sicherzustellen.

Aktion

- Wir beginnen mit dem **Umbenennen** des ”Testfallsatz” Knotens von ”unbenannt” in ”Demo Tests”.
- Falls ein **Dialog** bzgl. der Aktualisierung von Verweisen erscheint, können wir diesen einfach mit **”Ja”** beantworten.
- Der nächste Schritt ist, den vom Schnellstart-Assistenten erzeugten Knoten **”Vorbereitung”** in den **”Testfallsatz” zu verschieben** und zwar an die **erste Position** vor den enthaltenen Testfall. Das Verschieben kann mit Hilfe der Maus (Drag&Drop), des Kontextmenüs (rechte Maustaste Ausschneiden/Einfügen) oder der Tastenkombination Strg-X und Strg-V durchgeführt werden.

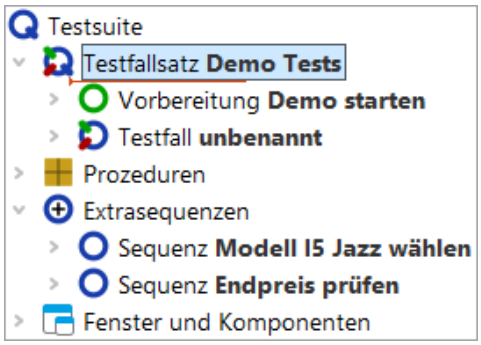

Abbildung 20.11: Beginn der Strukturierung

Als Nächstes gilt es, aus den beiden vorher aufgezeichneten Sequenzen einen Testfall zu machen.

Aktion

- **Benennen** Sie dazu den Testfall Knoten von ”unbenannt” in ”Erster” um.
- **Öffnen** Sie den Testfall Knoten durch einen Klick auf das ’>’ Symbol.
- **Verschieben** Sie die beiden Sequenzen aus den ”Extrasequenzen” in den Testfall.

Wenn Sie den Testfall Knoten nicht öffnen, versucht QF-Test die Sequenzknoten hinter dem Testfall Knoten auf der gleichen Ebene einzufügen. Dies ist jedoch für Sequenzknoten nicht zulässig.

QF-Test nimmt immer Sequenzen auf. Diese haben die gleiche Funktion wie Testschritte. Testschritte werden jedoch im Bericht aufgeführt. Man kann sie ineinander umwandeln, was wir Ihnen in den nächsten Schritten informationshalber zeigen wollen.

Aktion

- Öffnen Sie das **Kontextmenü** für den **ersten** der beiden Sequenzknoten mit der rechten Maustaste.
- Wählen Sie Knoten konvertieren in...→Testschritt
- Führen Sie dasselbe für den **zweiten** Sequenzknoten durch.

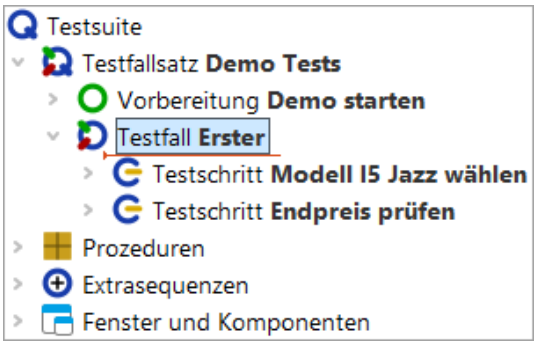

Abbildung 20.12: Der Baum nach der Neustrukturierung

Damit haben wir die wichtigsten Schritte zur Strukturierung unserer Testsuite abgeschlossen.

20.5 Beenden der Anwendung

Was uns als Basiselement noch fehlt, ist eine Aufräumsequenz, die das SUT sauber beendet.

Es gibt verschiedene Wege eine Anwendung zu beenden, z.B. über den "Fenster schließen" Knopf rechts oben, durch Drücken von Alt-F4 oder das Menü Datei→Beenden. Alle diese Varianten lassen sich direkt aufzeichnen. Wir werden die letzte Möglichkeit nutzen, so dass Sie die folgenden Schritte durchführen können:

Aktion

- **Aufnahme starten** ● .
- Menüaktion **Datei→Beenden** durchführen.
- Das Fenster der Demoanwendung verschwindet.
- **Aufnahme beenden** ■ .
- Benennen Sie die aufgenommene Sequenz **in "Demo beenden" um**.
- Öffnen Sie das **Kontextmenü** für den Sequenzknoten und wählen Sie den Menüpunkt **Knoten konvertieren in...→Aufräumen** .
- Zuletzt **verschieben** Sie den Aufräumenknoten nach oben, so dass er **der letzte Knoten im Testfallsatz** ist.

Hinweis

Der Aufräumknoten kann nur per Drag and Drop in den Testfallsatz verschoben wer-

den, wenn dessen letzter Kindknoten eingeklappt ist. Um einen Knoten während einer Drag and Drop Operation ein- oder auszuklappen, verweilen Sie einen Moment mit dem Mauszeiger über dem Dreieck neben dem Knoten.

Sie sollten folgendes Resultat erhalten:

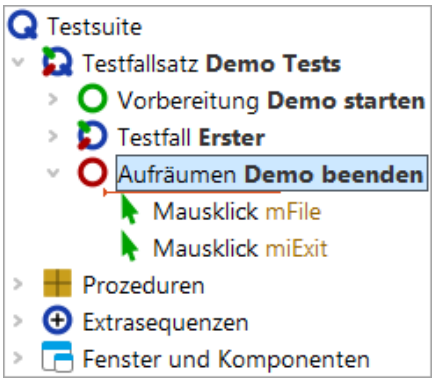

Abbildung 20.13: Die einfache Aufräumsequenz

Damit haben wir die wichtigsten Schritte zur Strukturierung unserer Testsuite abgeschlossen.

20.6 Gesamte Suite ausführen

Als Abschluss wollen wir unsere neue Suite ausführen:

Aktion

- **Beenden** Sie dazu nun bitte den SUT Client, falls er läuft.
- **Markieren** Sie den "Testsuite" **Wurzelknoten**.
- Führen Sie diesen durch Drücken von "Wiedergabe" ▶ oder der Eingabe Taste aus.

Das SUT sollte erscheinen, der Testfall aufgeführt und das SUT wieder beendet werden.

Wie wir wissen, wird das Ergebnis des Testlaufs im Protokoll festgehalten:

Aktion

- Um dieses anzuschauen, können wir den Button **"Protokoll anzeigen"** in der Werkzeugleiste oder alternative die Tastenkombination Strg-L nutzen.

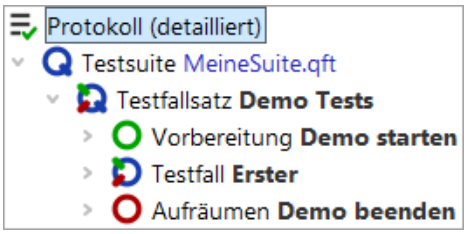

Abbildung 20.14: Der Protokollbaum der eigenen Testsuite

Wir hatten bereits im ersten Kapitel gesehen, wie das Protokoll für die Fehleranalyse genutzt werden kann.

Wir möchten damit dieses Kapitel beenden und einen Schritt weiter in Richtung Fehlerdiagnose gehen. Hierbei stellt der Debugger ein wichtiges Werkzeug innerhalb QF-Test dar. Seine Handhabung und Möglichkeiten werden im nächsten Kapitel beschrieben.

Kapitel 21

Eine Prozedur erstellen (Win)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Eine Prozedur erstellen'
https://www.qftest.com/de/yt/tutorial-3.html

In den beiden vorherigen Kapiteln haben Sie gelernt wie man eine Applikation über QF-Test startet, so dass Maus- und Tastatureingaben aufgenommen werden können, auch wie man Checks aufnimmt und wie man das Ergebnis zu einem Testfall zusammenfasst. Diese Herangehensweise ist ausreichend, solange die Tests einfach und nicht allzu viele sind. Sobald jedoch die Zahl der Tests zunimmt, ist es wichtig, sogenannte "Prozeduren" einzusetzen.

Prozeduren sind ein Mittel um Sequenzen wiederverwendbar zu machen und damit Doppelvorkommen zu vermeiden. Dies ist wichtig, um eine einfache und effiziente Wartbarkeit von Tests über die Zeit zu erreichen.

Prozeduren können in Packages gruppiert werden. Prozeduren und Packages sind die Basis für die Modularisierung der Tests.

21.1 Wiederverwendbare Abschnitte identifizieren

In diesem Abschnitt werden wir die Testsuite ErsteWinTests.qft, die Sie bereits aus Kapitel 19 kennen, weiterentwickeln.

Aktion

- **Kopieren Sie ErsteWinTests.qft** aus dem Unterverzeichnis `qftest-9.0.0/doc/tutorial` der QF-Test Installation in ein Arbeitsverzeichnis und
- **öffnen Sie ErsteWinTests.qft**.

- Wenn Sie die Änderung, die wir an der Demo-Testsuite vornehmen werden, sichern wollen, so **speichern Sie diese in einem Arbeitsverzeichnis** wie am Ende von Abschnitt 20.1 (228) beschrieben.

Bitte sehen Sie sich den Testschritt "Zurücksetzen" in den beiden Testfällen an. Die beiden Testschritte sind identisch.

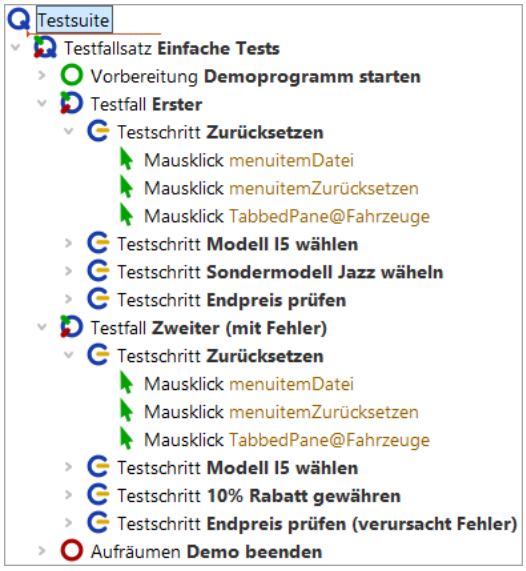

Abbildung 21.1: Zwei identische Testschritte

Gemäß obiger Überlegungen wäre es also sinnvoll, den Testschritt in eine Prozedur umzuwandeln.

21.2 Manuelle Erstellung von Prozeduren

Es gibt mehrere Methoden Prozeduren zu erstellen und Prozeduraufrufe einzufügen. Wir fangen mit der manuellen an, bei der ein (leerer) Prozedurknoten eingefügt wird, in den dann die entsprechenden Aktionen verschoben werden. Danach erstellen wir den zugehörigen Prozeduraufruf.

Es ist gut, wenn man diese grundlegenden Schritte kennt. Es gibt jedoch eine zweite, elegantere Methode Prozeduren zu erstellen, die wir im Anschluss zeigen werden.

Also los, fügen wir eine Prozedur von Hand ein: Wir beginnen mit dem Anlegen des Prozedurknotens, dem wir einen geeigneten Namen geben.

Aktion

- **Öffnen Sie den Prozedur Knoten** und achten Sie darauf, dass er auch selektiert (blau markiert) ist.

- Wählen Sie **Einfügen→Prozedurknoten→Prozedur**.
- Tragen Sie als Name "**zurücksetzen**" ein. Die anderen Felder brauchen nicht befüllt zu werden.
- Drücken Sie **OK** um die Erstellung der Prozedur abzuschließen.
- **Öffnen Sie** die neu erstellte "zurücksetzen" Prozedur.

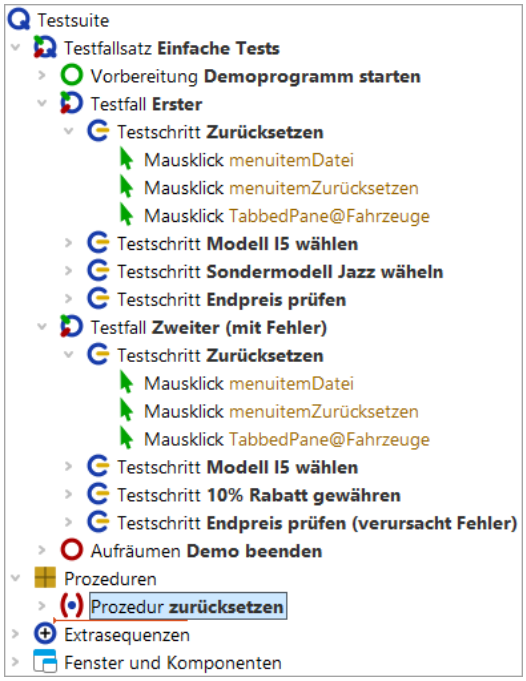

Abbildung 21.2: Prozedurknoten erstellen

Im zweiten Schritt befüllen wir die Prozedur mit den entsprechenden wiederverwendbaren Aktionen.

Aktion

- **Selektieren Sie** die drei "Mausklick" Knoten im Testschritt. Um mehr als einen Knoten zu selektieren, klicken Sie den ersten der Knoten an, dann drücken Sie die Shift Taste und klicken den letzten der zu selektierenden Knoten während Sie die Shift Taste gedrückt halten.
- **Verschieben** Sie diese nach unten in die Prozedur, z.B. mit der Maus (Drag and drop) oder über Ausschneiden/Einfügen im Menü Bearbeiten oder über das Kontextmenü.

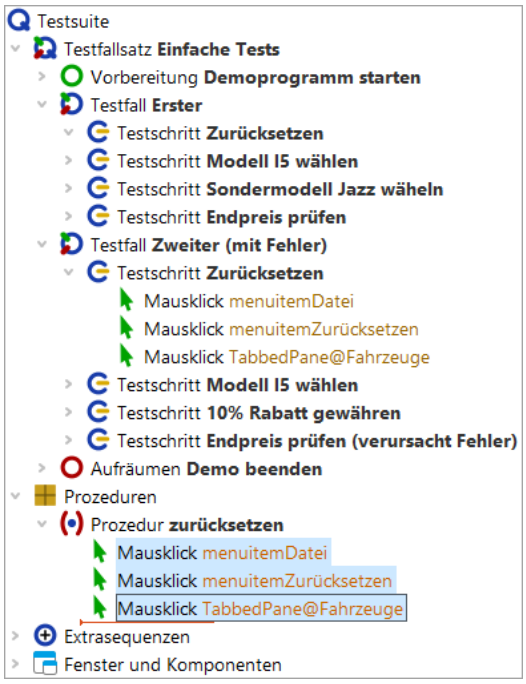

Abbildung 21.3: Prozedur mit Inhalt befüllen

Im dritten Schritt fügen wir einen Prozeduraufruf an Stelle der drei verschobenen Mausklicks ein.

Aktion

- **Selektieren** Sie den **Testschritt "Zurücksetzen"**, der geöffnet sein sollte.
- **Wählen Sie** den Menüpunkt Einfügen→Prozedurknoten→Prozeduraufruf oder verwenden Sie das Tastaturkürzel Strg-A.

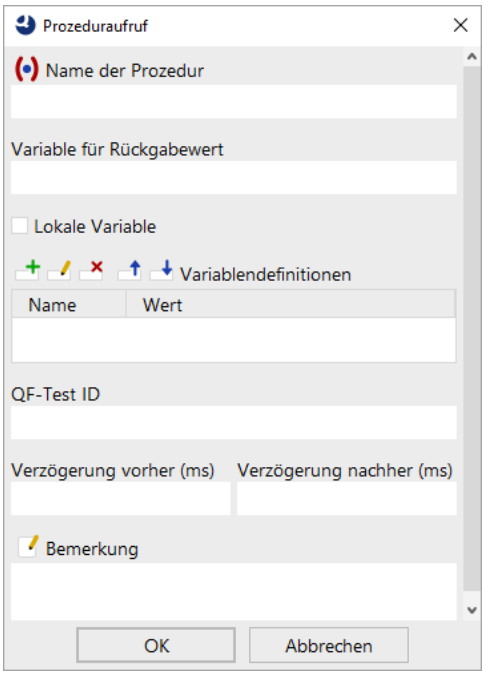

Abbildung 21.4: Prozeduraufruf einfügen

Aktion

- **Drücken Sie** den Prozedur-Auswahlknopf in dem Dialog links neben der Beschriftung "Name der Prozedur".
- **Wählen Sie "zurücksetzen"** aus der Prozedurliste. Weitere Eingaben sind nicht nötig.
- Drücken Sie **OK** in beiden Dialogen um die Erstellung des Prozeduraufrufs abzuschließen.

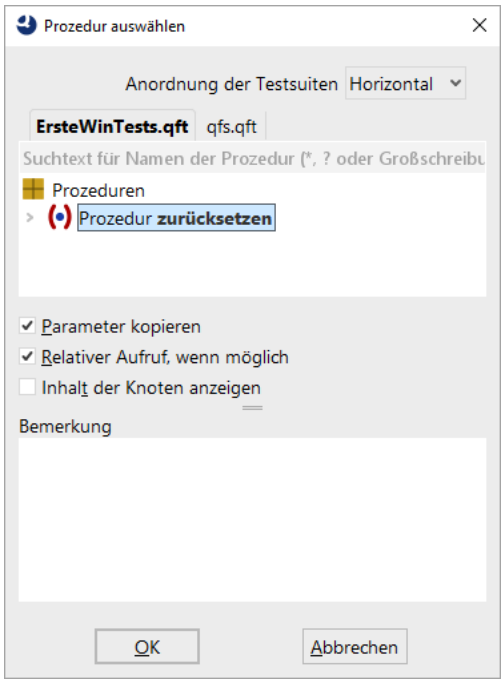

Abbildung 21.5: Prozedur auswählen

Um wirklich einen Mehrwert durch die Prozedur zu erlangen, müssen wir nun den Inhalt des Testschritts im zweiten Testfall ebenfalls durch einen Aufruf der Prozedur ”zurücksetzen” ersetzen.

Sie können dies wie oben beschrieben tun oder Sie führen folgende **alternative Schritte** zur Erstellung des Prozeduraufrufs aus:

Aktion

- **Öffnen Sie** den Testschritt ’Zurücksetzen’ des zweiten Testfalls.
- **Löschen Sie** die drei Mausklick Knoten daraus.
- **Selektieren Sie** den Prozedurknoten ”zurücksetzen”.
- **Ziehen Sie** den Prozedurknoten ”zurücksetzen” mit der Maus in den Testschrittknoten. Kopieren/einfügen kann ebenfalls verwendet werden. Dadurch wird der Prozedurknoten nicht verschoben, sondern ein entsprechender Prozeduraufruf erzeugt.

Die Testsuite sollte anschließend wie folgt aussehen:

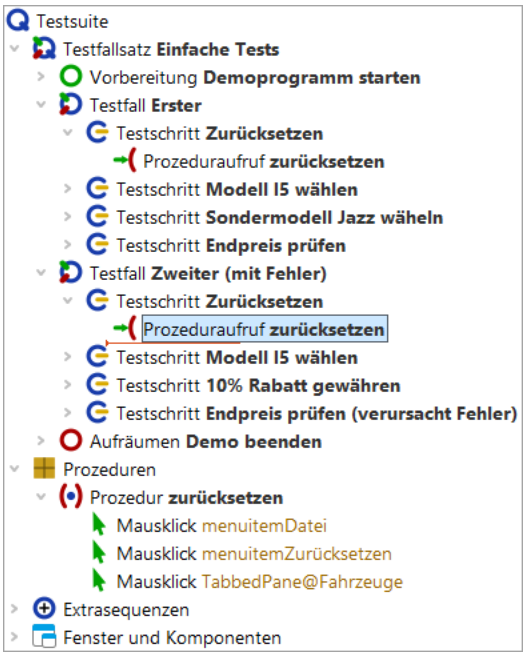

Abbildung 21.6: Testsuite mit Prozedur

Wenn Sie nun die Testfälle ausführen, sollten diese funktionieren wie zuvor. Im Protokoll sind jetzt auch die Prozeduraufrufe und deren Ausführung zu sehen.

21.3 Knoten in Prozedur konvertieren

Wie bereits am Anfang des letzten Abschnitts erwähnt, bietet QF-Test eine Alternative um Prozeduren wesentlich schneller zu erstellen.

Aktion

- **Markieren Sie den Testschritt oder Sequenz-Knoten**, der die wiederverwendbaren Schritte enthält, die zur Prozedur umgewandelt werden sollen.
- Wählen Sie den Menüpunkt **Operationen→Knoten konvertieren in→Prozedur** aus oder verwenden Sie das Tastaturkürzel Strg-Umschalt-P.

Wie Sie sehen, ist der Testschritt bzw. der Sequenzknoten verschwunden. Anstelle dessen befindet sich ein Prozeduraufruf. Außerdem wurde eine Prozedur mit dem Namen des ehemaligen Testschritts bzw. der Sequenz im Abschnitt "Prozeduren" erstellt. Sie enthält genau die gleichen Kindknoten wie zuvor der Testschritt bzw. die Sequenz.

Bei der Aufnahme einer Sequenz in QF-Test hat sich das Vorgehen bewährt, der Sequenz sofort einen Namen zu geben und sie anschließend in eine Prozedur zu konvertieren. Auch wenn man nur eine Vermutung hat, dass sich die aufgenommenen Schritte irgendwo wiederholen könnten.

Kapitel 22

Komponenten (Win)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Komponenten'
https://www.qftest.com/de/yt/tutorial-4.html

Werfen wir nun einen Blick auf den letzten verbleibenden Bereich des Testsuite Fensters, den Fenster und Komponenten Knoten. Zuvor möchten wir Ihnen jedoch zeigen, wie Unterelemente von Komponenten wie Tabellen, Bäumen und Listen adressiert werden.

22.1 Adressierung von Unterelementen von Tabellen, Bäumen und Listen

Unterelemente von Tabellen, Bäumen und Listen werden über Indizes angesprochen. Die wichtigsten beiden Indextypen sind der numerische und der Textindex. Zur Demonstration nehmen wir als nächstes einen Mausklick auf eine Tabellenzelle auf und sehen uns die aufgenommene QF-Test ID der Komponente näher an.

Aktion

- **Starten Sie das CarConfig Demo**, falls dieses nicht bereits läuft. Führen Sie dazu den Vorbereitung Knoten in der Testsuite aus.
- **Aktivieren Sie den Aufnahmemodus** über "Aufnahme starten" ● .
- **Klicken Sie auf eine Tabellenzelle**, z.B. das erste Modell.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .

Den aufgenommenen Mausklick finden Sie im Bereich Extrasequenzen.

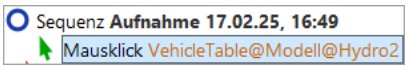

Abbildung 22.1: Adressierung einer Tabellenzelle

Die aufgenommene QF-Test ID der Komponente ist `VehicleTable@Modell&0`. Sie setzt sich aus den folgenden Teilen zusammen:

- `VehicleTable` ist die QF-Test ID der Komponente der Tabelle selbst.
- `@` und `&` trennen die einzelnen Teile voneinander. Gleichzeitig definieren sie den Typ des darauf folgenden Index: auf `@` folgt ein Textindex, auf `&` ein numerischer Index.
- `Modell` ist der Textindex für die Spalte mit der Überschrift 'Modell'.
- `0` ist der numerische Index für die erste Tabellenzeile.

Hinweis Numerische Indizes beginnen immer mit 0.

Sie können beide Indextypen für Zeilen und Spalten verwenden. Dabei ist nur wichtig, dass das Trennzeichen und der Typ des zugehörigen Index zusammenpassen.

Aktion

- **Ändern Sie die QF-Test ID der Komponente** so, dass das dritte Preisfeld adressiert wird. Verwenden Sie dafür numerische Indizes.

Die Lösung hierfür lautet `VehicleTable&1&2`.

Um das Modell 'I5' über Textindizes anzusprechen, tragen Sie `VehicleTable@Modell@I5` ein. Das gleiche Feld kann man numerisch mittels `VehicleTable&0&4` ansprechen oder mit gemischten Indizes mittels `VehicleTable&0@I5` oder `VehicleTable@Modell&4`.

Der dritte Indextyp von QF-Test ist ein Index mit regulärem Ausdruck. Reguläre Ausdrücke werden verwendet, um Zeichenketten durch einen Ausdruck zu ersetzen, der verschiedene Zeichenketten adressieren kann. Sozusagen eine "Sternchensuche", wobei reguläre Ausdrücke wesentlich mächtiger sind und eine eigene Syntax besitzen. Eine genauere Beschreibung regulärer Ausdrücke finden Sie im Handbuch. Beispiel: Das Modell 'I5' könnte man also auch über `VehicleTable@Modell%I.*` ansprechen.

Listen werden analog zu Tabellen adressiert, nur dass sie nur einen einzigen Index benötigen.

Bäume haben ebenfalls nur einen einzigen Index. Dieser ist der Pfad durch den Baum zu dem adressierten Baumknoten. Der Pfad setzt sich aus den einzelnen Knoten zusammen, die durch Schrägstriche ("/") voneinander getrennt werden.

Aktion

- **Starten Sie das CarConfig Demo**, falls dieses nicht bereits läuft. Führen Sie dazu den Vorbereitung Knoten in der Testsuite aus.
- **Öffnen Sie das Baum-Beispiel:** Wählen Sie im CarConfig Demo den Menüpunkt Einstellungen→Sondermodelle..., selektieren Sie ein Modell und drücken die Schaltfläche 'Details'.
- **Aktivieren Sie den Aufnahmemodus** über "Aufnahme starten" ● .
- **Klicken Sie auf einen Baumknoten**, z.B. 'Beschreibung'.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .

Für den Baumknoten "Beschreibung" wird die folgende QF-Test ID der Komponente aufgenommen: `DetailsTree@/Information/Beschreibung`. Die einzelnen Bestandteile davon sind:

- `DetailsTree` ist die QF-Test ID der Komponente des Baums selbst.
- `@` trennt die QF-Test ID der Komponente des Baums vom Index. Die Syntax ist hierbei analog zu der der Tabellenindizes, d.h. `@` steht für einen Textindex, `&` für einen numerischen Index und `%` für einen Index mit regulärem Ausdruck.
- `/Information/Beschreibung` ist der Textindex für den Baumpfad zum Knoten 'Beschreibung'.

Wenn Sie den Knoten über einen numerischen Index adressieren wollen, verwenden Sie `DetailsTree&/0/1`.

Kapitel 23

Benutzen des Debuggers (Win)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Benutzen des Debuggers'
https://www.qftest.com/de/yt/tutorial-5.html

In diesem Kapitel lernen Sie, wie der in QF-Test integrierte intuitive Debugger benutzt wird. Diejenigen unter Ihnen, die bereits Erfahrungen mit anderen IDEs, wie z.B. Eclipse haben, werden hier Ähnlichkeiten in Funktion und Nutzen des Debuggers feststellen.

Wir werden uns mit den folgenden Debugger-Funktionen beschäftigen:

- Setzen eines Breakpoints[255] mittels Strg-F8 (⇧-⌘-B auf macOS).
- Testausführung pausieren[264] mittels Pausetaste ⏸ oder der Tastenkombination für das Unterbrechen der Wiedergabe ("Keine Panik"-Taste) Alt-F12.
- Schrittweise Ausführung[256] mittels "Einzelschritt ausführen" , "Gesamten Knoten ausführen" und "Bis Knotenende ausführen" .
- Knoten überspringen[258] mittels "Knoten überspringen" und "Aus Knoten herausspringen" .
- Debug-Modus bei Fehler oder Exception aktivieren[260].
- Fehlerbehebung aus dem Protokoll heraus[262].
- Den aktuellen Fehler im Protokoll direkt anspringen über Strg-J. (Ins Protokoll springen in Kapitel Abschnitt 23.5[262]).

Hinweis

Anstatt über die Schaltflächen können die Befehle auch über die Menüzeile oder Tastaturkürzel abgesetzt werden. Die Kürzel stehen neben den Optionen in den QF-Test Menüs, sofern vorhanden. Eine vollständige Übersicht der von QF-Test verwendeten

Tastaturkürzel finden Sie im Anhang Tastaturkürzel im Handbuch. Dort findet sich auch ein kleiner Helfer für die Funktionstastenbelegung von QF-Test zum Befestigen an der Tastatur.

Es gibt noch einige weitere Debugger-Funktionen wie

- "Aktuellen Knoten finden" (Aktuellen Knoten finden in Abschnitt 24.3[274]),
- "Ausführung hier fortsetzen" über das Popup-Menü des entsprechenden Knotens (Abbildung 24.9[277]),
- die "Exception erneut werfen" ,
- die Tabelle der Variablendefinitionen (Abschnitt 24.3[274]),

auf die wir in späteren Kapiteln eingehen werden.

23.1 Setzen eines Breakpoints

Zunächst einmal soll der Debugger aktiviert werden. Dies kann auf mehrere Arten erfolgen, zum Beispiel durch das Setzen eines Haltepunktes (Breakpoint) vor Start des Testlaufs. Der Zweck eines Haltepunktes ist es, den Testlauf an einer Stelle, die man näher untersuchen möchte, zu unterbrechen. Sobald QF-Test auf den Breakpoint trifft, wird die Testausführung pausiert und der Debugger-Modus aktiviert. Der Pauseknopf ist nun gedrückt.

Aktion

- Selektieren Sie einen Knoten und **drücken Sie Strg-F8 (⇧-⌘-B auf macOS)**. Der Haltepunkt wird durch das Symbol B kenntlich gemacht.

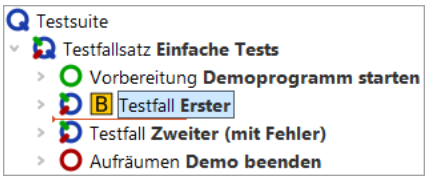

Abbildung 23.1: Breakpoint setzen

Aktion

- Selektieren Sie den Testsuite Knoten und starten Sie den Testlauf über die Taste Enter.

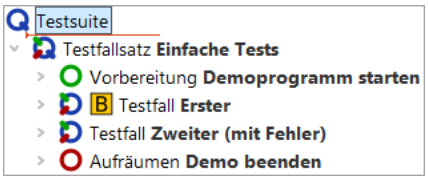

Abbildung 23.2: Testlauf starten

Aktion

- Löschen Sie den Breakpoint wieder, indem Sie nochmals **Strg-F8** (**⇧-⌘-B** **auf macOS) drücken**.

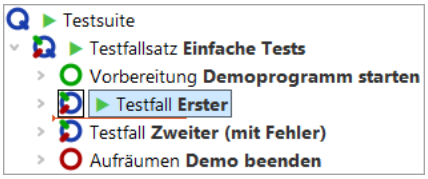

Abbildung 23.3: Breakpoint löschen

Man kann einen Breakpoint nicht nur über das Tastaturkürzel Strg-F8 sondern auch über den Menüpunkt Debugger→Breakpoint an/aus oder alternativ durch Rechtsklick auf den Knoten und Auswahl von Breakpoint an/aus im Popup-Menü setzen bzw. löschen. Für die weiteren Debugger-Funktionen werden wir hauptsächlich die jeweiligen Schaltflächen nennen, aber auch hier stehen die anderen Varianten zur Verfügung.

Wieder sehen Sie den kleinen Pfeil, der diesmal anzeigt, welcher Knoten als nächster ausgeführt wird. Dieser Knoten wird **aktueller Knoten** genannt. Bei der Aktivierung des Debug-Modus navigiert QF-Test zum aktuellen Knoten, falls dieser nicht bereits sichtbar ist, und selektiert diesen, d.h. die Zeile wird blau hinterlegt.

Das Kommando Debugger→Alle Breakpoints löschen ist ebenfalls nützlich, wenn alle Haltepunkte in allen geöffneten Testsuiten gelöscht werden sollen.

Es gibt keine Beschränkung für die Anzahl an Breakpoints, die Sie in Ihrer Testsuite setzen können. Haltepunkte werden beim Schließen der Testsuite nicht mit abgespeichert.

23.2 Schrittweise Ausführung

Nun wollen wir die Testfälle schrittweise ausführen.

Aktion

- Bitte experimentieren Sie ein wenig mit **"Einzelschritt ausführen"** , **"Gesamten Knoten ausführen"** und **"Bis Knotenende ausführen"** .

Wie Sie sicher festgestellt haben werden, öffnet **"Einzelschritt ausführen"** einen Knoten mit Kindern und macht den ersten Kindknoten zum aktiven Knoten. Dies ist wie immer an der Pfeilmarkierung des Knotens erkennbar.

Wenn Sie an dem Punkt weitergemacht haben, an dem die Ausführung der Testsuite im letzten Abschnitt pausiert war, d.h. vom Knoten "Testfall: Erster" aus, so würde nun der Testfall geöffnet werden:

Abbildung 23.4: Einzelschritt ausführen

Im Falle eines Blattknotens, d.h. eines Knotens, der keine Kinder hat, ist die Wirkung die gleiche wie die der folgenden Funktion.

Mittels der Schaltfläche **"Gesamten Knoten ausführen"** wird ein Knoten inklusive aller Kindknoten ausgeführt. Der als nächstes auszuführende Knoten auf der gleichen Ebene wird dann der aktive und erhält den Pfeil.

Abbildung 23.5: Gesamten Knoten ausführen

"Bis Knotenende ausführen" führt die verbleibenden Geschwisterknoten aus und stoppt beim nächsten auszuführenden Knoten der übergeordneten Hierarchieebene.

Abbildung 23.6: Bis Knotenende ausführen

Im Beispiel ist dies der Aufräumen Knoten. Wie bereits im ersten Kapitel Ein vollständiger Testlauf[224] erläutert, haben Vorbereitung und Aufräumen Knoten die besondere Eigenschaft, dass sie vor und nach **jedem Testfall** ausgeführt werden, um so einen definierten Anfangszustand für jeden Testfall herzustellen.

Hinweis Dieses Verhalten tritt nur auf, wenn Sie die komplette Testsuite oder den Testfallsatz gestartet hatten und sich im Debug-Modus befinden. Wenn keine Testausführung aktiv war und Sie nur den Testfall selektiert hatten, so bewirkt die Funktion "Gesamten Knoten ausführen", dass der Testfall ausgeführt wird und dann der nächste Testfall selektiert wird.

Aktion

- Führen Sie die Aufräumen und Vorbereitung Knoten aus, indem Sie mit Hilfe der Schaltfläche die **gesamten Knoten ausführen** und dann über **den zweiten Testfall öffnen**. Dies ist eine Vorbereitung für das nächste Kapitel, in dem es um das Überspringen von Knoten geht.

Hinweis Bitte beachten Sie bei der interaktiven Testerstellung bei Menüs und Comboboxen, dass diese häufig zuklappen, wenn die Applikation den Fokus verliert, was beim Wechsel in den Debug-Modus der Fall ist. In diesem Fall empfiehlt es sich, den Knoten, der ein Menü oder eine Combobox öffnet, und den Knoten, der die Auswahlaktion durchführt, gemeinsam auszuführen, also nicht zwischendurch in den Debug-Modus zu gehen. Dies kann man zum Beispiel dadurch erreichen, dass man nach dem Knoten, der die Auswahlaktion durchführt, einen Haltepunkt B setzt und bei Erreichen des Knotens, der das Menü oder die Combobox öffnet, die Testausführung durch Lösen der Pausetaste freigibt.

23.3 Knoten überspringen

Die "Überspringen" Funktionen erweitern die Fähigkeiten des Debuggers von QF-Test in einer Weise, die über den Funktionsumfang von Standardprogrammierumgebungen

hinausgeht. Wie der Name andeutet, erlauben die ”Überspringen” Operationen einen oder mehrere Knoten während des Testlaufs auszulassen, d.h. weiter zu springen ohne diese auszuführen. Dies kann aus verschiedensten Gründen sinnvoll sein. Sei es um schnell an eine gewisse Position in Ihrem Testablauf zu gelangen oder um einen aktuell zu einem Fehler führenden Knoten zu überspringen.

Am Schluss des letzten Abschnitts haben wir den ersten Testschritt im zweiten Testfall zum aktiven Knoten gemacht. Dies ist er Ausgangspunkt für unsere nächste Aktion:

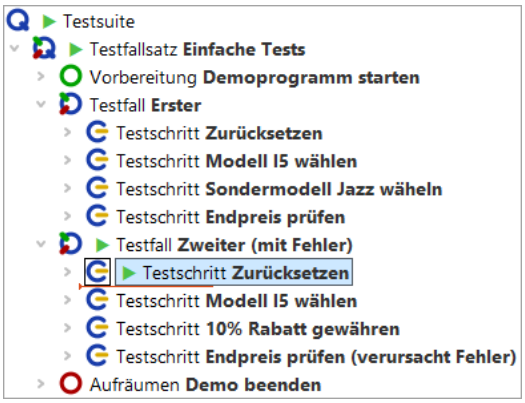

Abbildung 23.7: Testausführung am ersten Knoten des zweiten Testfalls pausiert

- Drücken Sie nun die Schaltfläche **”Knoten überspringen”** . QF-Test springt einfach über den aktiven Knoten ohne ihn oder seine Kindknoten auszuführen. Anschließend pausiert QF-Test beim nächsten auszuführenden Knoten auf der gleichen Ebene.

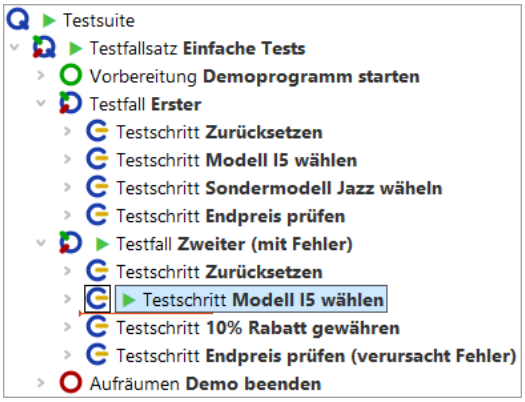

Abbildung 23.8: ”Knoten überspringen”

Aktion

- Und zuletzt die Schaltfläche **”Aus Knoten herausspringen”** . Sie sehen so-

fort, dass QF-Test aus dem Knoten, in dem Sie sich befinden, herausspringt ohne weitere Kindknoten auszuführen.

Abbildung 23.9: ”Aus Knoten herausspringen”

Hinweis Noch eine Bemerkung zu ”Knoten überspringen” und ”Aus Knoten herausspringen”: Benutzen Sie diese mit Vorsicht! Aus einer Sequenz herauszuspringen, bevor diese zu Ende gelaufen ist, kann dazu führen, dass Ihr SUT in einem Status belassen wird, mit dem andere Sequenzen oder Tests in der Suite nicht aufsetzen können.

23.4 Debug-Modus bei Fehler oder Exception aktivieren

Beim Debuggen eines Tests ist es oft hilfreich, wenn die Testausführung genau dann stoppt und in den Debug-Modus gewechselt wird, wenn ein Fehler, eine Exception oder manchmal auch nur eine Warnung auftritt.

Mittels dieser Technik werden wir in diesem Abschnitt und dem nächsten den zweiten Testfall debuggen.

Aktion

- Bitte **öffnen Sie das Debugger-Menü** und ändern Sie die Standardeinstellungen wie folgt:
- **Klicken Sie** auf den Menüpunkt **Debugger→Debugger aktivieren** um ihn zu aktivieren.
- **Klicken Sie** auf den Untermenüpunkt **Debugger→Optionen→Unterbrechen bei Fehler** um auch diese Funktion zu aktivieren.

Wenn Sie nun das Debugger-Menü und das Optionen-Untermenü wieder öffnen sollte es wie folgt aussehen:

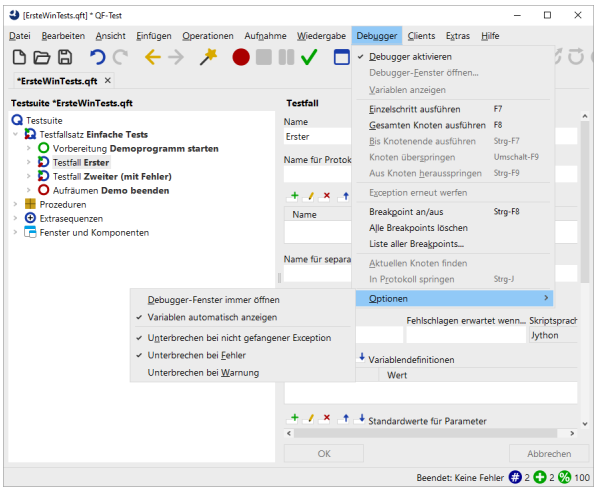

Abbildung 23.10: Debugger-Optionen: Test bei Fehler anhalten

Wir müssen die Debugger-Optionen ändern, da sonst der Test einfach durchlaufen würde, analog zu den vorherigen Beispielen aus Kapitel eins und zwei.

Aktion

- **Selektieren Sie den Testsuite Knoten** und starten Sie anschließend den Test mittels "Wiedergabe starten" ▶ .

QF-Test hält bei dem fehlerhaften Knoten an und wechselt in den Debug-Modus:

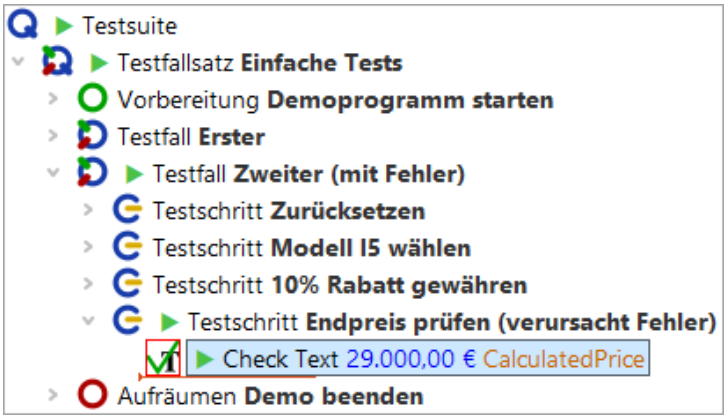

Abbildung 23.11: QF-Test pausiert bei Fehler

Der Knoten, der den Fehler verursacht hat, wird durch ein rotes Quadrat markiert. Außerdem erscheint ein Fehlerdialog, der uns Näheres zur Fehlerursache mitteilt. Über

diesen wechseln wir in das Protokoll, das wie so oft der Schlüssel zur Fehlerbehebung ist.

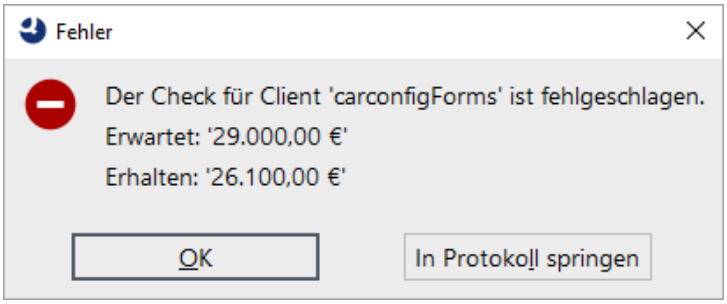

Abbildung 23.12: Fehlermeldung

Aktion

- **Klicken Sie** auf die Schaltfläche **In Protokoll springen** der Fehlermeldung.

23.5 Fehlerbehebung aus dem Protokoll heraus

Über die Schaltfläche **In Protokoll springen** (siehe Fehlermeldung in Abbildung Abbildung 23.12[262]) wird das Protokoll direkt bei dem entsprechenden Knoten geöffnet.

Neben der eigentlichen Fehlermeldung wurden etliche weitere Informationen zur Testumgebung zum Zeitpunkt des Fehlers protokolliert. Neben Bildschirmabbildern zum Fehlerzeitpunkt enthält das Protokoll unter dem Knoten, der den Fehler verursachte, eine Liste der gebundenen Variablen (Stacktrace). Auf die Nützlichkeit des Stacktrace werden wir zu einem späteren Zeitpunkt eingehen (Die Variablendefinitionen-Tabelle[270]).

Beim vorliegenden Fehler wird der falsche Wert im Check Text Knoten der Testsuite erwartet. Zur Fehlerbehebung muss dieser durch den tatsächlich angezeigten ersetzt werden. Dies geht bei einem Check mit festem Wert, um den es sich hier handelt, am einfachsten, indem Sie

Aktion

- auf den rot umrandeten Fehler-Knoten **"Fehlgeschlagen: Check Text: default ..." rechtsklicken** und
- im Kontextmenü **Check-Knoten mit erhaltenen Daten aktualisieren** **auswählen**.

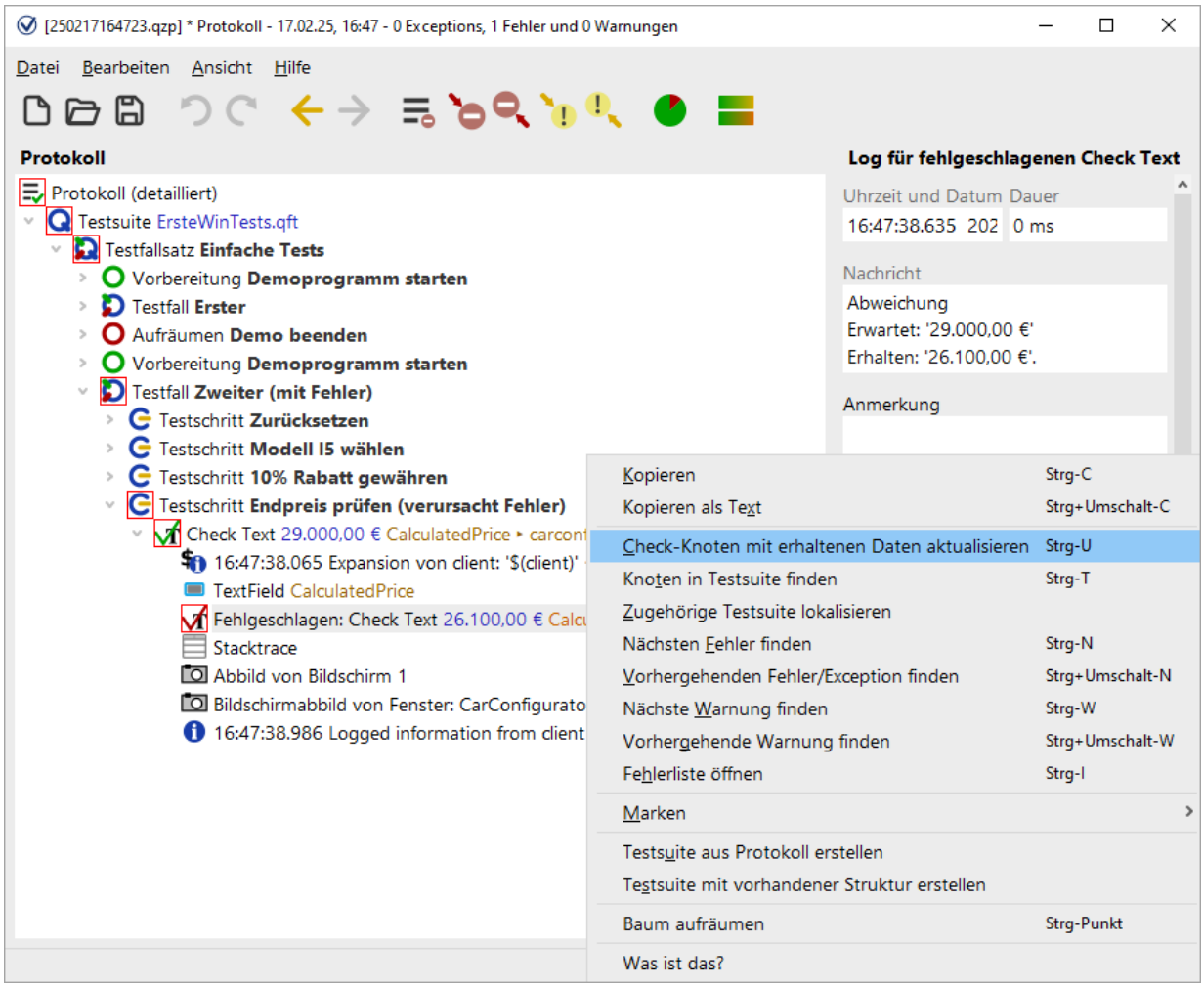

Abbildung 23.13: Check-Knoten mit erhaltenen Daten aktualisieren

QF-Test navigiert zum betroffenen Check Text Knoten in der Testsuite und aktualisiert den Wert des Attributs Text anhand der aus dem SUT ausgelesenen Daten.

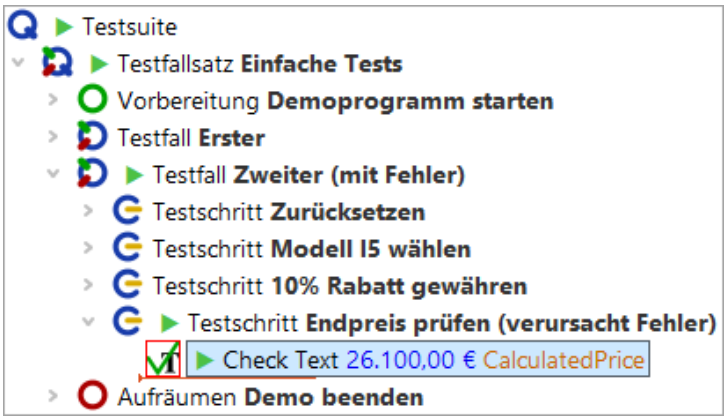

Abbildung 23.14: Korrigierter Check-Knoten

Der Knoten enthält nun zwar den korrekten Wert, ist aber immer noch rot umrandet, da er noch nicht wieder ausgeführt wurde. Dies wollen wir nun tun.

Aktion

- führen Sie den Test also fort, indem Sie auf den Pauseknopf ⏸ drücken und so **die Pause lösen**.

QF-Test führt den Rest der Testsuite aus. In unserem Fall sind das der Check Text und der Aufräumen Knoten. Anschließend informiert Sie QF-Test, dass ein Fehler aufgetreten ist. Diesen haben wir allerdings bereits während des Testlaufs behoben.

Ins Protokoll springen: Falls Sie das Protokoll an der Stelle, an der sich die Testausführung gerade befindet, öffnen wollen, brauchen Sie aus dem Debugging Modus heraus nur den Menüpunkt **Debugger→Jump to Run log** anzuklicken oder das Tastaturkürzel Strg-J zu drücken. Wenn Sie einfach nur das Protokoll öffnen wollen, ohne an die aktuelle Stelle zu springen, steht Ihnen Strg-L zur Verfügung, was auch nach Ende des Testlaufs weiterhin funktioniert.

23.6 Testausführung pausieren

Wenn ein Test gerade ausgeführt wird und Sie den Debug-Modus aktivieren wollen, so können Sie entweder schnell einen Breakpoint auf einen noch nicht ausgeführten Knoten setzen oder Sie drücken einfach die Schaltfläche "Pause" ⏸ um den Debug-Modus sofort zu aktivieren.

Um die Ausführung fortzusetzen lösen Sie den Pauseknopf ⏸ , völlig unabhängig von der Art und Weise wie der Debug-Modus aktiviert wurde.

Wir wollen die vorhandenen Testfälle im Verlauf dieses Tutorials weiter verwenden. Allerdings wurde der Fehler in dem zweiten Testfall jetzt behoben. Insofern macht es Sinn "(mit Fehler)" aus dem Namen des zweiten Testfalls zu löschen, genauso wie "(verursacht Fehler)" aus dem Namen des Testschritts.

Es gibt manchmal Situationen, in denen das SUT ständig den Fokus für sich beansprucht. Dann kann es schwierig sein, das QF-Test Fenster lange genug im Vordergrund zu halten, um die Pausetaste drücken zu können. In einem solchen Fall steht Ihnen die "Keine Panik"-Taste Alt-F12 zur Verfügung. Sie unterbricht alle laufenden Tests sofort. Zur Weiterführung des Tests können Sie diese Tastenkombination erneut drücken.

Kapitel 24

Variablen und Prozedurparameter (Win)

In diesem Kapitel lernen Sie, wie man eine Prozedur einsetzt um die gleichen Schritte auf unterschiedlichen Daten auszuführen. Außerdem sehen Sie, wie man Variablen einsetzt. Ebenso wird die Fehleranalyse in Bezug auf Variablen behandelt.

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Variablen und Prozedurparameter'
https://www.qftest.com/de/yt/tutorial-6.html

. Im Video wird eine Java-Applikation für die Erläuterungen verwendet. Bei nativen Windows-Applikationen gibt es hinsichtlich der Variablenverwendung keine Unterschiede.

24.1 Prozedur mit Variable

Sehen Sie sich den letzten Testschritt "Endpreis prüfen" in unseren beiden Testfällen an.

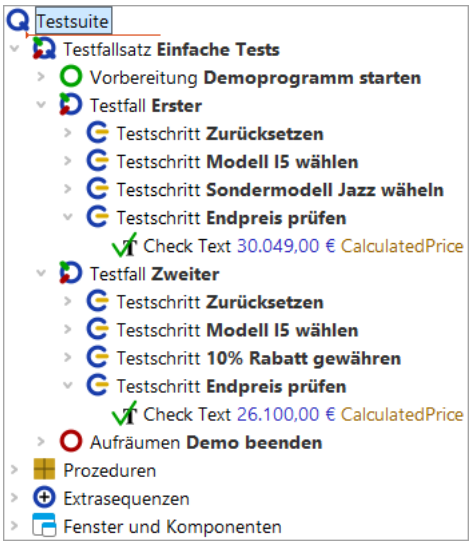

Abbildung 24.1: Zwei fast gleiche Testschritte

Es wird der gleiche Schritt ausgeführt, jedoch mit unterschiedlichen Daten. Auch wenn es sich nur um einen Schritt handelt, macht es Sinn eine Prozedur daraus zu machen. Vielleicht kommen wir später auf die Idee, die hartkodierten Werte `30.049,00 €` und `26.100,00 €` in ein anderes Format zu bringen, so dass der Check auf das Feld "Endpreis" auch für andere Währungen funktioniert. Diesen Algorithmus zweimal zu implementieren wäre auf jeden Fall nicht sinnvoll.

Aktion

- **Selektieren Sie den "Check text" Knoten im ersten Testfall.**
- Wählen Sie den Menüpunkt **Operationen→Knoten einpacken in→Sequenz** aus oder verwenden Sie das Tastaturkürzel Strg-Umschalt-S um ihn in eine Sequenz einzupacken.
- **Nennen Sie die Sequenz 'prüfeEndpreis'.** Dieser Name entspricht der Java-Konvention die Wörter zusammenzuschreiben. Andererseits erlaubt QF-Test auch Leerzeichen in Prozedurnamen, so dass Sie der Java-Konvention nicht zu folgen brauchen.
- Drücken Sie **Strg-Umschalt-P** um auf kürzestem Weg den Sequenzknoten in eine Prozedur zu konvertieren (wie aus dem letzten Kapitel bekannt). Wie Sie sehen, wurde die Sequenz durch einen Prozeduraufruf von "prüfeEndpreis" ersetzt.
- **Klicken Sie doppelt auf den Prozeduraufruf**, um zur Prozedur im Prozeduren Knoten zu springen.
- **Öffnen Sie** den neu erstellten Prozedurknoten um den Inhalt zu sehen.

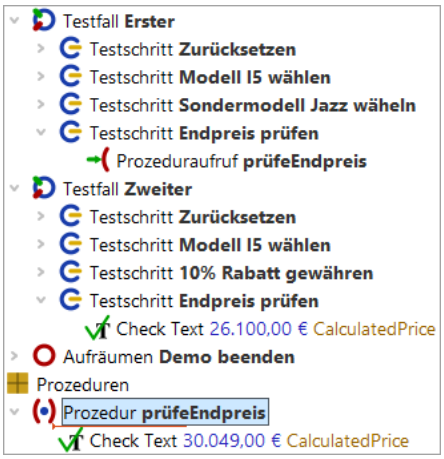

Abbildung 24.2: Prozedur mit hartkodiertem Wert

Wie erwartet befindet sich der "Check text" Knoten in der Prozedur. Er ist jedoch nur für einen einzigen Preis gültig, nämlich `30.049,00 €`. Da wir die gleiche Prozedur auch für den zweiten Testfall verwenden möchten, müssen wir den Preis durch eine Variable ersetzen. Der Wert dieser Variable sollte dann beim Prozeduraufruf mit übergeben werden.

Im nächsten Beispiel werden wir einen Parameter mit Standardwert im Prozedurknoten einfügen. Standardwerte werden häufig verwendet, wenn der entsprechende Parameter bei den meisten Prozeduraufrufen den Standardwert erhalten würde. Dann braucht man den Standardwert nicht jedes Mal zu spezifizieren, sondern kann auf den im Prozedurknoten definierten Wert zurückgreifen. Obwohl das auf den Preis als Parameter nicht zutrifft, können wir ihn gerade deshalb verwenden um zu zeigen, wie ein Standardwert funktioniert und wie man ihn bei Bedarf mit einem anderen Wert überschreiben kann.

Als erstes fügen wir also eine Variable mit Standardwert ein:

Aktion

- **Selektieren Sie die Prozedur 'prüfeEndpreis'**
- **Drücken Sie den** "Zeile einfügen" Knopf über der Tabelle "Standardwerte für Parameter".
- **Tragen Sie `preis`** als Namen des Parameter ein.
- **Tragen Sie `30.049,00 €`** als Wert ein.
- Drücken Sie **OK**.

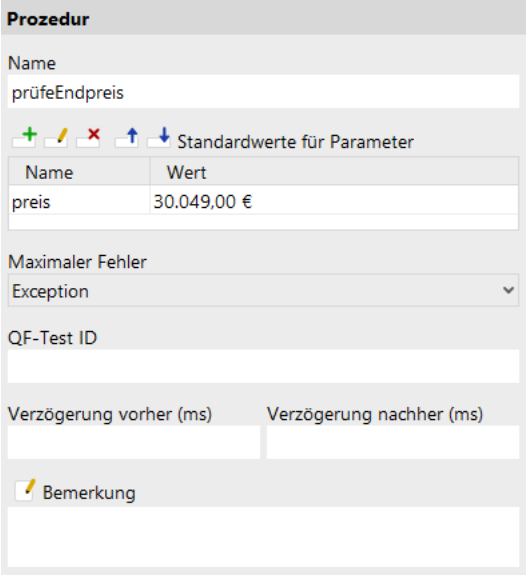

Abbildung 24.3: Die Details eines Prozedurknotens

Im nächsten Schritt ersetzen wir den Wert des Text Attributs des Check Text Knotens durch eine Referenz auf die Variable.

Hinweis

Variablensyntax: Wenn Sie mit Variablen arbeiten, ist es wichtig, sich bewusst zu sein, dass Sie an bestimmten Stellen QF-Test mitteilen wie, eine Variable heißt und an anderen, dass QF-Test auf den Wert einer Variablen zugreifen soll.

In obigem Beispiel wird QF-Test in der Namensspalte für die Standardwerte der Variablenname mitgeteilt. In diesem Fall brauchten Sie nur `preis` einzutragen.

Im Text Attribute des Check Text Knotens soll der Wert der Variablen verwendet werden. Bei QF-Test geschieht dies dadurch, dass Sie den Variablenname in `$()` setzen, hier `$(preis)`. Falls Sie den Variablennamen nicht in `$()` setzen, würde QF-Test den Preis mit der Zeichenfolge `preis` vergleichen.

Aktion

- **Selektieren Sie den Check Text Knoten** in der Prozedur ”prüfeEndpreis”.
- **Tragen Sie `$(preis)`** im Text Attribut der Check Text Knotendetails ein.
- **Drücken Sie ’OK’** in den Knotendetails.

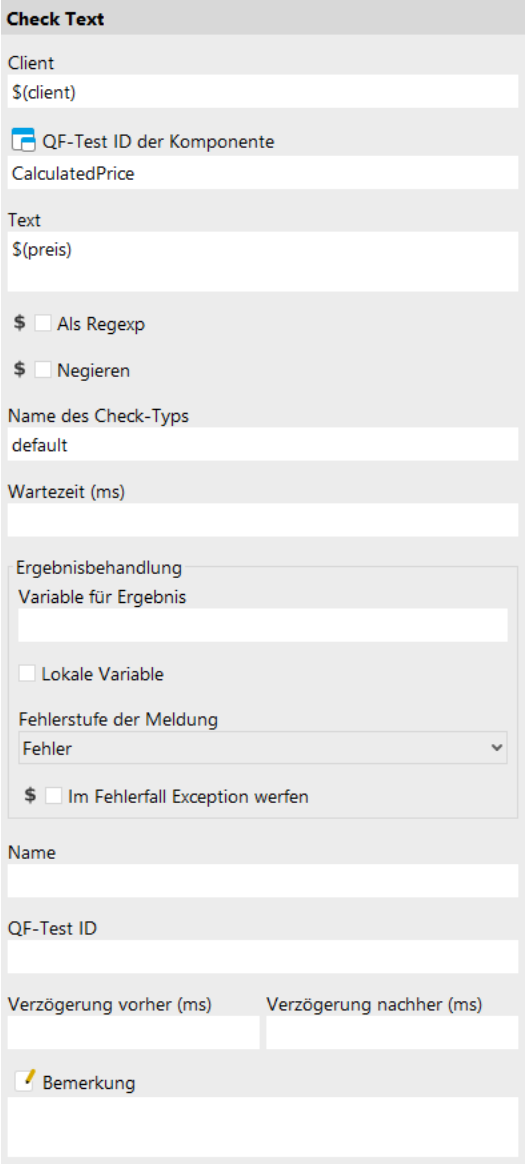

Abbildung 24.4: 'Check text'-Knoten

Aktion

- **Führen Sie den ersten Testfall aus.**

Der Testfall sollte fehlerfrei durchlaufen.

24.2 Die Variablendefinitionen-Tabelle

Im nächsten Schritt fügen wir einen Prozeduraufruf im zweiten Testfall ein.

Aktion

- **Ersetzen Sie den Check Text Knoten** des zweiten Testfalls **durch einen Prozeduraufruf von "prüfeEndPreis"**. Sie können einfach den Prozeduraufruf aus dem ersten Testfall kopieren oder den Prozeduraufruf wie oben beschrieben einfügen.

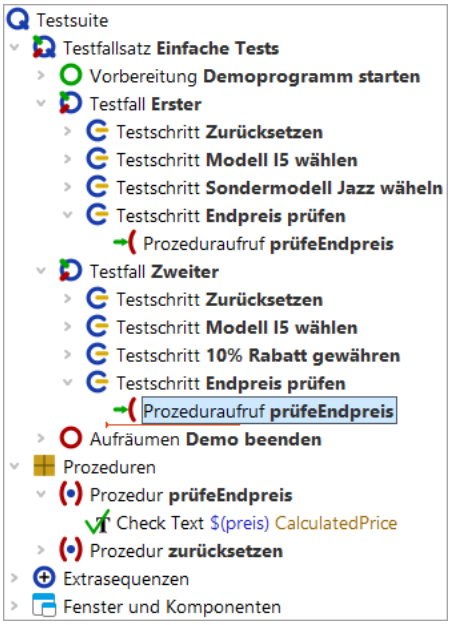

Abbildung 24.5: Prozeduraufruf von "prüfeEndPreis" in der zweiten Prozedur

Hinweis

Wenn im Prozeduraufruf bereits der Preis mit Standardwert eingetragen ist, rührt das daher, dass der Prozeduraufruf mit Hilfe der Prozedur selbst erzeugt wurde. Entweder durch Kopieren der Prozedur oder durch eine Drag-and-Drop Aktion unter Verwendung des Prozedurknoten oder über direktes Einfügen der Prozedur wie weiter oben erläutert. Aktuell geht es jedoch um den Standardwert. Daher bitten wir Sie, den evtl. vorhandenen Preis-Parameter zu löschen, wenn Sie dem Tutorial Schritt für Schritt folgen wollen. Dazu klicken Sie das rote X über der Variablendefinitionen-Tabelle.

Aktion

- Überprüfen Sie, ob der **Debugger so eingestellt ist, dass er bei Fehlern unterbricht** (siehe Debugger-Optionen: Test bei Fehler anhalten(261)).
- **Selektieren Sie den Knoten "Testfall: Zweiter"**.
- **Führen Sie ihn aus**, entweder über den Knopf ▶ oder durch Drücken der Eingabe Taste.

Eine Fehlermeldung zeigt an, dass ein anderer als der angezeigte Preis erwartet wurde. Was lief schief? Lassen Sie uns nach dem Fehler forschen. Üblicherweise würden wir ins Protokoll schauen, aber es gibt noch eine andere wichtige Informationsquelle.

Aktion

- **Klicken Sie OK**, um die Fehlermeldung zu schließen.

Im Debugging-Modus sehen Sie rechts unten im QF-Test Fenster eine Knotenliste mit Variablen, die von diesen gebunden sind.

Aktion

- Eventuell müssen Sie die Variablendefinitionen-Tabelle vergrößern, um alle Einträge sehen zu können.

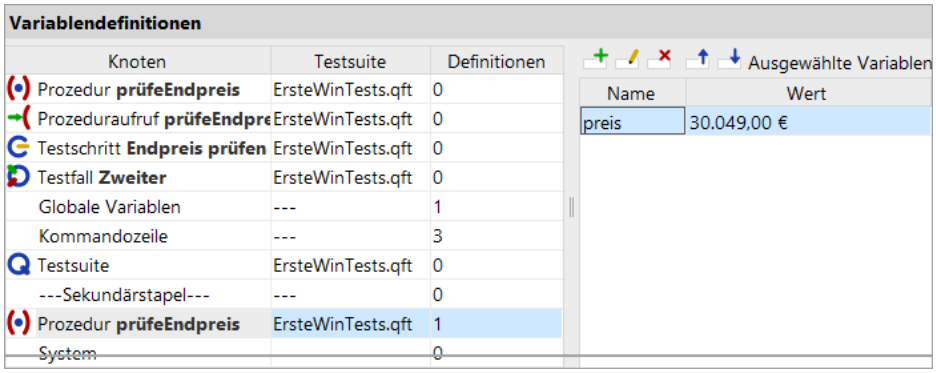

Abbildung 24.6: Variablendefinitionen

Die Variablendefinitionen-Tabelle ist beim Debuggen sehr hilfreich, da sie die aktuellen Werte der Variablen anzeigt. Sie unterstützt beim Arbeiten mit Prozeduren als auch beim Verständnis, wie QF-Test den richtigen Variablenwert ermittelt.

Hinweis

QF-Test geht die Variablendefinitionen-Tabelle immer von oben nach unten durch.

Sie sehen, dass in den ersten Zeilen keine Variablen gebunden sind. Auf der Ebene "Globale Variablen" ist eine Variable gebunden und auf dem Sekundärstapel für "Prozedur: prüfeEndpreis" eine weitere. Die globale Variable wird für die Verbindung zur SUT Applikation verwendet und wurde vor dem Anwendungsstart gesetzt. (vgl. Starten der Anwendung(214)). Die andere Variable interessiert uns im Moment mehr - sie hat jedoch den falschen Wert.

Dieser Wert auf dem Sekundärstapel ist der Standardwert, da er dann verwendet wird, wenn nirgendwo sonst einer Variablen mit dem gleichen Namen ein Wert zugewiesen wurde.

Um es richtig zu machen, müssen wir den korrekten Wert beim Prozeduraufruf an die Prozedur übergeben. Wieder gibt es mehrere Arten, dies zu tun. Ein Weg wäre, eine neue Zeile in der Variablendefinitionen-Tabelle in den Details des Prozeduraufrufs einzufügen, ähnlich wie beim Prozedurknoten im vorigen Abschnitt.

Wenn es jedoch bereits mehrere Prozeduraufrufe gibt, ist folgendes einfacher:

Aktion

- **Beenden Sie die laufende Testausführung** mittels ■ .

- Führen Sie einen **Rechtsklick auf den Prozedurknoten** aus und wählen **Weitere Knotenoperationen→Parameter von Referenzen anpassen** im Popup-Menü.

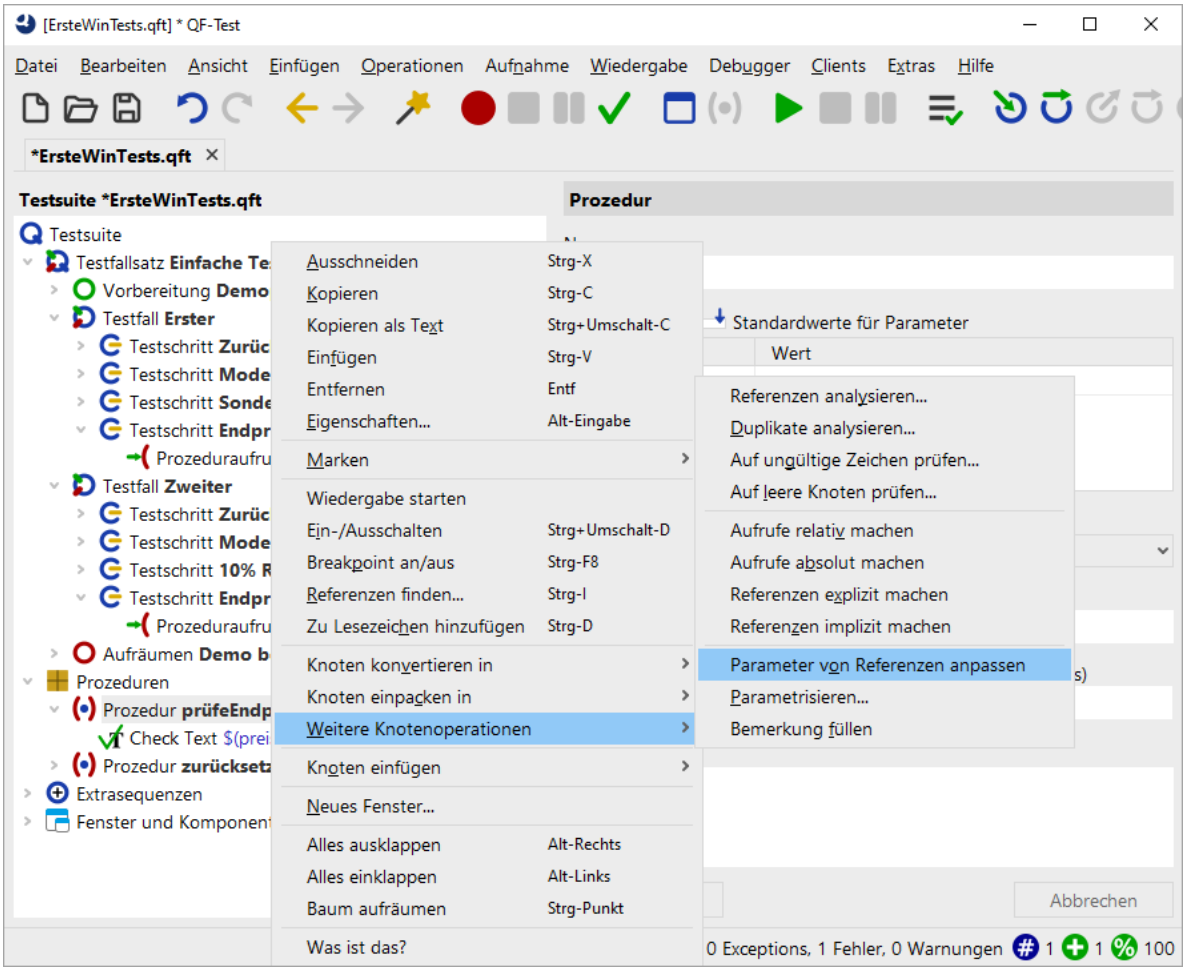

Abbildung 24.7: Popup-Menü für "Parameter von Referenzen anpassen"

- Prüfen Sie im folgenden Dialog, dass ein Häkchen bei **Fehlende Parameter beim Aufrufer hinzufügen** gesetzt ist, und **bestätigen Sie mit OK**.

Im Prozeduraufruf erzeugt QF-Test für jeden Standardwert eine Zeile in der Variablen Definitionen Tabelle. In unserem Fall wurde eine Zeile für den Parameter mit dem Namen `preis` und dem Wert `30.049,00` € hinzugefügt.

Auch damit wird es im zweiten Testfall noch nicht funktionieren, auch wenn der Wert direkt übergeben wird, weil es sich immer noch um den Standardwert handelt, der hier

nicht korrekt ist. Bitte verändern Sie den Wert noch nicht, damit wir Ihnen mittels des entstehenden Fehlers eine weitere Möglichkeit des Debuggens zeigen können.

Aktion

- **Schließen Sie den Dialog "Angepasste Knoten"**, den QF-Test anzeigt, um Sie über die vervollständigten Knoten zu informieren.

24.3 Fortgeschrittenes Debuggen mittels Variablendefinitionen-Tabelle

Als nächstes wollen wir die Variablendefinitionen-Tabelle unter die Lupe nehmen und herausfinden, wie man sie für Debugging-Zwecke einsetzen kann. Daher belassen Sie bitte den fehlerhaften Wert, der im vorigen Abschnitt im Prozeduraufruf eingefügt wurde.

Dazu soll die Ausführung des Testfalls beim Prozeduraufruf unterbrochen werden um dann mittels Einzelschritten in die Prozedur zu gehen. Dabei werden wir uns ansehen, was in der Variablendefinitionen-Tabelle passiert. Anschließend wollen wir direkt aus der Variablendefinitionen-Tabelle zum fehlerhaften Prozeduraufruf springen und dort den Parameterwert korrigieren.

Aktion

- **Setzen Sie einen Breakpoint** bei "Prozeduraufruf: prüfeEndpreis" im zweiten Testfall.
- **Führen Sie den zweiten Testfall aus**.
- Wenn QF-Test am Breakpoint anhält, **führen Sie zwei Einzelschritte in die Prozedur** mittels aus und **beobachten dabei die Variablendefinitionen-Tabelle**.

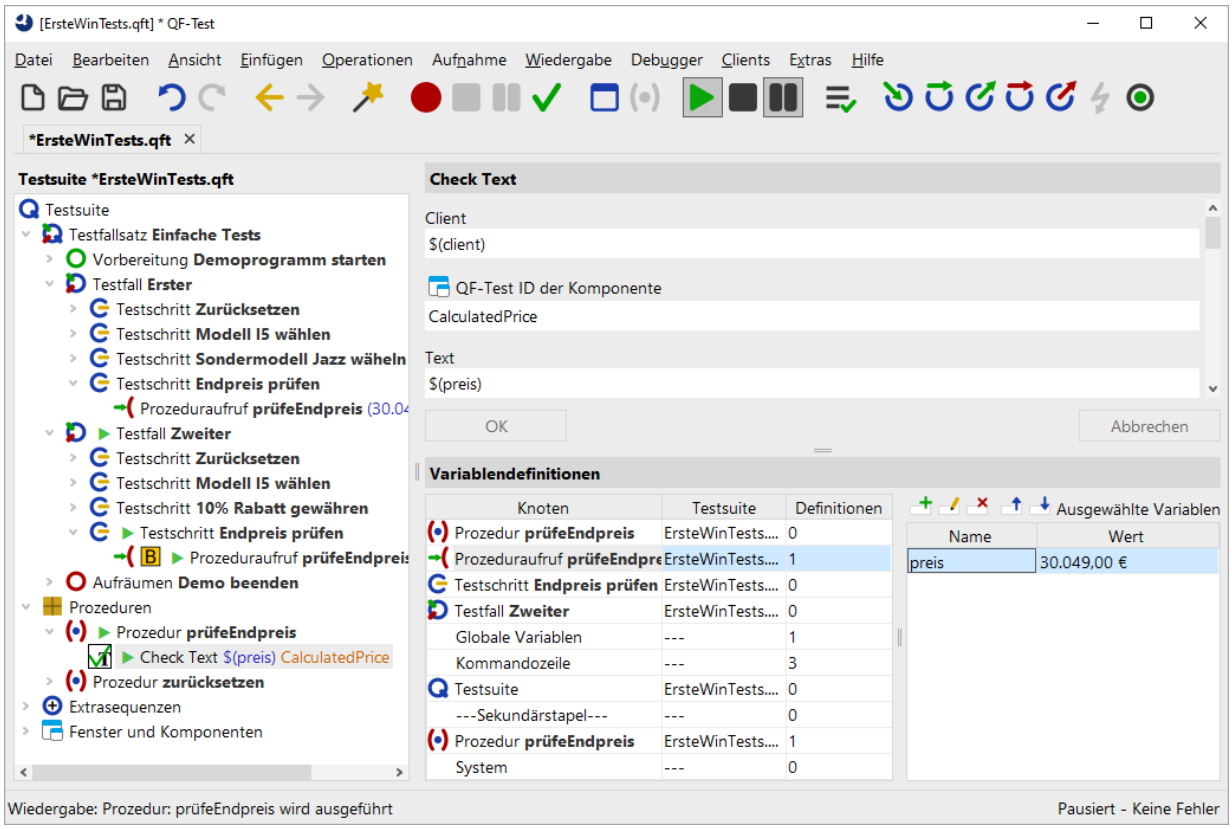

Abbildung 24.8: Variablendefinitionen-Tabelle zeigt den falschen Wert

Wenn Sie mit Einzelschritten in die Prozedur gehen, wird beim ersten eine Zeile für "Prozeduraufruf: prüfeEndpreis" oben in der Tabelle erzeugt und beim zweiten die Zeile "Prozedur: prüfeEndpreis".

Jetzt gibt es die Variable `preis` auf zwei verschiedenen Ebenen in der Variablen Definitionen Tabelle: in der Zeile für "Prozeduraufruf: prüfeEndpreis" und in der Zeile "Prozedur: prüfeEndpreis" auf dem Sekundärstapel, wobei keiner der beiden Variablenwerte der richtige ist.

In QF-Test können Sie interaktiv die Werte von Variablen in der Variablendefinitionen-Tabelle verändern, wenn Sie sich im Debugging-Modus befinden. Sie können sogar neue Variablen hinzufügen oder vorhandene löschen. Damit können Sie arbeiten, solange sich die Variablen auf dem Variablenstapel befinden, in unserem Fall solange wie die Prozedur ausgeführt wird.

Änderungen des aktuellen Variablenwertes in der Variablendefinitionen-Tabelle bewirken keine Anpassung des im Prozeduraufrufknoten eingetragenen Parameterwerts. Der Wert muss explizit im Prozeduraufruf geändert werden.

Die schnellste Methode um zum Prozeduraufruf zu gelangen ist ein Doppelklick auf die

Prozeduraufrufzeile (zweite Zeile) in der Variablendefinitionen-Tabelle. Diese Methode ist besonders hilfreich, wenn Sie umfangreiche Testsuiten debuggen und der Knoten, zu dem Sie springen wollen, nicht im Testsuite-Fenster angezeigt wird. Statt eines Doppelklicks können Sie auch einen Rechtsklick auf die Zeile ausführen und den Menüpunkt Zu Knoten in Testsuite springen wählen.

Aktion

- **Führen Sie einen Doppelklick auf die zweite Zeile mit dem Prozeduraufruf** in der Variablendefinitionen-Tabelle aus.
- **Setzen Sie den Parameterwert auf den richtigen Wert**, d.h. `26.100,00 €`.

Umgekehrt wird auch der aktuelle Wert in der Variablendefinitionen-Tabelle dadurch nicht verändert. Um dies zu erreichen müssen wir den Prozeduraufruf erneut ausführen. Allerdings ist die Testausführung über diesen Punkt bereits hinaus.

Hinweis

Daher wollen wir hier eine weitere nützliche Funktion des Debuggers zeigen, mit der man den QF-Test anweisen kann, den nächsten auszuführenden Knoten zu verändern. Dazu selektieren Sie den entsprechenden Knoten und wählen den Menüpunkt **Ausführung hier fortsetzen** oder verwenden das Tastaturkürzel Strg-,.

Also, um den neu gesetzten Wert auszuprobieren:

Aktion

- **Führen Sie einen Rechtsklick auf den Knoten "Prozeduraufruf: prüfeEndpreis"** in der zweiten Prozedur aus.
- **Wählen Sie "Ausführung hier fortsetzen"** im Popup-Menü.

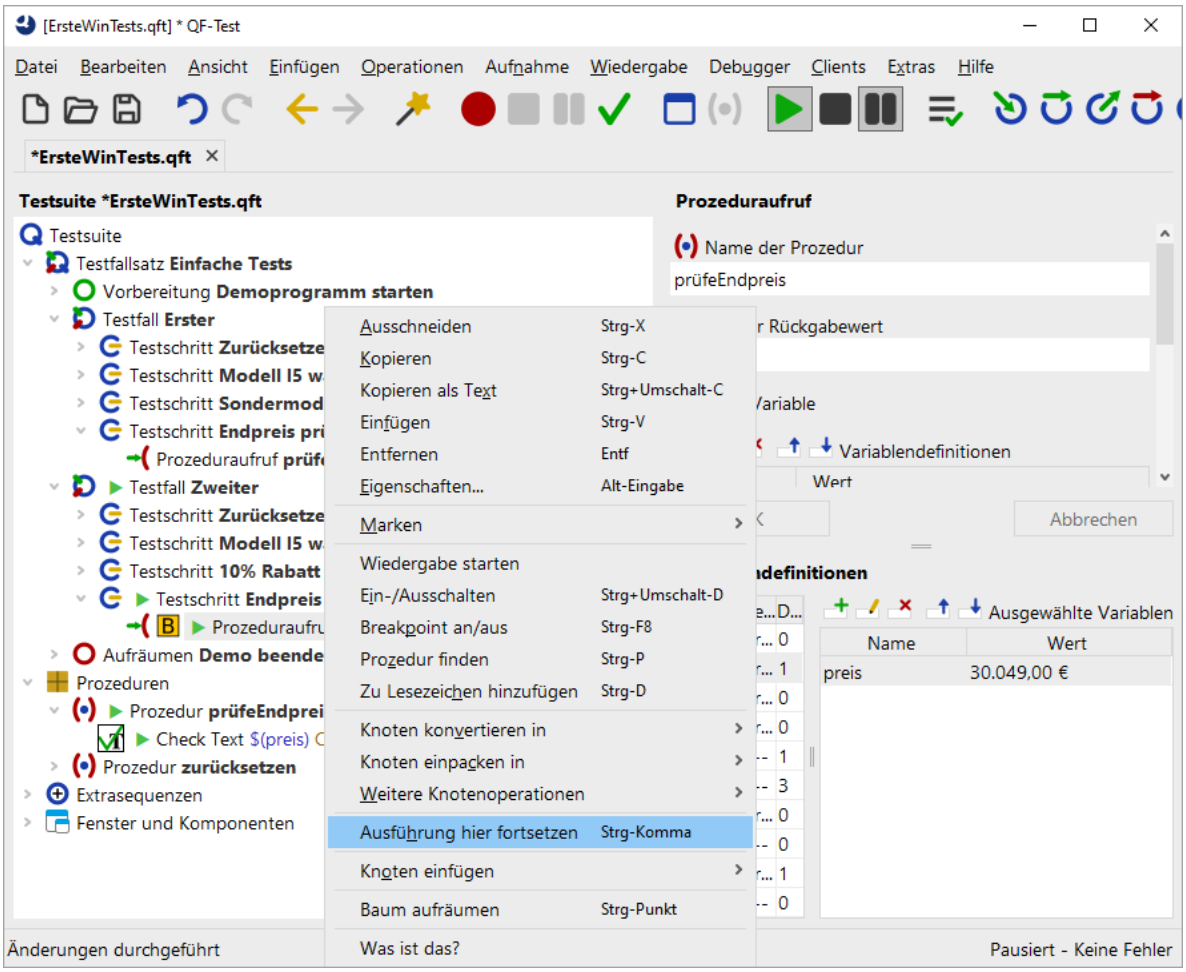

Abbildung 24.9: Ausführung hier fortsetzen

In der Variablendefinitionen-Tabelle sind die zwei obersten Zeilen verschwunden. Der Grund ist, dass Sie die Prozedur verlassen haben (wenn auch "rückwärts") und dass dadurch der Prozeduraufruf mit den daran gebundenen Variablen vom Aufrufstack genommen wurde.

Aktion

- Lösen Sie den Pauseknopf ⏸ .

Nun sollte kein Fehler mehr auftauchen.

Hinweis

Da die Variablendefinitionen-Tabelle äußerst hilfreich ist, wenn Sie nach fehlerhaften Variablenwerten fahnden, wird eine Kopie davon auch unter dem Knoten "Stacktrace" im Protokoll abgespeichert, in dem die Variablenwerte genau zum Zeitpunkt des Fehlers zu sehen sind.

Aktuellen Knoten finden: Manchmal entfernt man sich beim Debuggen ziemlich weit vom aktuellen Knoten und möchte anschließend wieder zu diesem Knoten zurückfinden.

Das geht am einfachsten indem man in der Toolbar "Aktuellen Knoten finden" drückt oder den Menüpunkt Debugger→Aktuellen Knoten finden wählt.

24.4 Variablen setzen

Zusätzlich zu den oben beschriebenen Wegen können die Variablen auch wie folgt gesetzt werden:

- Mittels Variable setzen Knoten,
- als Rückgabewert einer Prozedur,
- als Ergebnis eines QF-Test Knotens wie Text auslesen, Geometrie auslesen, Index auslesen und Check,
- in der 'Variablendefinitionen' Tabelle von Testsuite, Testfall, Testschritt, Sequenz und weiteren Knoten wie dem If oder Schleife Knoten,
- über Kommandozeilenparameter.

Informationen dazu, an welcher Stelle eine Variable am besten definiert wird, finden Sie im nächsten Abschnitt.

Ein Variable setzen Knoten kann über den Menüpunkt Einfügen→Diverse Knoten→Variable setzen eingefügt werden. In den Knotendetails können Sie angeben, ob es sich um eine lokale (Häkchen bei "Lokale Variable" setzen) oder eine globale Variable handeln soll.

Die folgende Abbildung zeigt die Details eines Variable setzen Knotens, den Sie als ersten Knoten im Vorbereitung Knoten finden. Es wird eine Variable mit dem Namen `client` definiert. Dass es sich um eine globale Variable handelt, erkennen Sie daran, dass das Attribut 'Lokale Variable' nicht gesetzt ist.

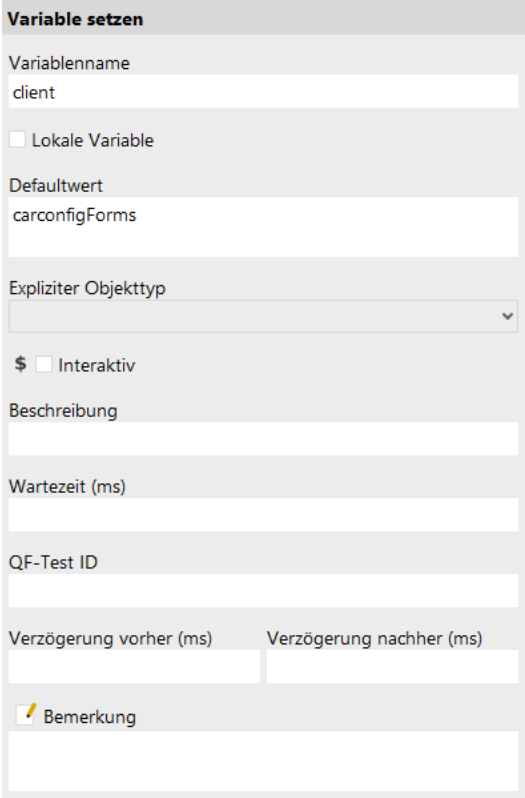

Abbildung 24.10: Details des Variable setzen Knoten

Wenn eine Variable mit dem Rückgabewert einer Prozedur gesetzt werden soll, geben Sie den Variablennamen im Attribut "Variable für Rückgabewert" des Prozeduraufrufs an. In der Prozedur selbst müssen Sie als letzten auszuführenden Knoten einen Return Knoten einfügen, der den betreffenden Wert zurückgibt.

Die Prozedur in der folgenden Abbildung liest den Rabattwert aus dem SUT und gibt den Wert an den aufrufenden Testfall zurück. Dort heißt die empfangende Variable `Rabatt` und ist als lokale Variable deklariert. Dieses Beispiel ist nicht in der Übungstestsuite enthalten.

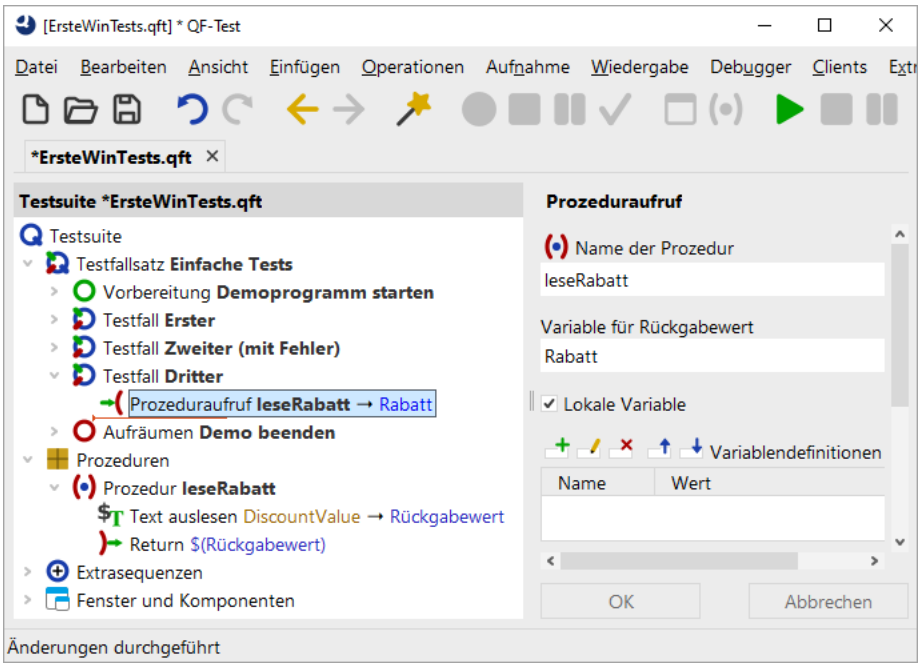

Abbildung 24.11: Prozedur mit Rückgabewert

Der Text auslesen Knoten in der obigen Abbildung ist einer der QF-Test Knoten, die direkt den Wert einer Variablen setzen. Dabei wird der Variablenname in dem entsprechend benannten Attribut eingetragen. Sie haben wiederum die Wahl, ob es eine lokale oder globale Variable werden soll.

Es gibt eine Reihe von Knoten, die eine Variablendefinitionen-Tabelle besitzen. Dort können Sie lokale Variablen setzen. Falls sich der betreffende Knoten in einer Prozedur befindet, wird die Variable als lokale Variable an die Prozedur gebunden, ansonsten als lokale Variable an den Testfall. Variablen, die mittels dieser Tabelle an den Testsuite Knoten gebunden sind, können von allen Knoten der Testsuite referenziert werden.

Alle Knoten, an die Variablen gebunden werden können, werden im Debugger-Modus im Variablen Definitionen Fenster rechts unten angezeigt, wenn sie gerade ausgeführt werden.

Variablen können auch über die Kommandozeile spezifiziert werden. Hierzu verwenden Sie den Kommandozeilenparameter `-variable`. Beispiel: `qftest -batch -variable "browser"="ie" test.qft`. Weitere Informationen hierzu finden Sie im Handbuch, Kapitel 'Kommandozeilenargumente'.

24.5 Ebenen für Variablendefinitionen

Hinweis

Dieser Abschnitt gibt Antworten auf die Frage, auf welcher Ebene eine Variable definiert werden sollte. Wenn Sie diese Frage momentan nicht interessiert, können Sie direkt zum nächsten Kapitel springen.

Variablen können auf unterschiedlichen Ebenen gebunden werden:

- Im Testsuite Knoten,
- in Testfällen und Prozeduren als Standard- oder als lokale Variablen,
- als Parameter in einem Prozeduraufruf,
- als globale Variable und
- als Kommandozeilenparameter.

Die Ebene, auf welcher eine Variable am sinnvollsten definiert wird, hängt vom Verwendungszweck der Variablen ab:

Prozedurparameter
Übergeben Sie einen Wert als Parameter an eine Prozedur, wenn die gleiche Prozedur mehr als einmal und mit unterschiedlichen Werten ausgeführt werden soll. Prozedurparameter werden in der Variable Definitionen Tabelle eines Prozeduraufruf Knoten angegeben.

Lokale Variablen in einer Prozedur
Lokale Variablen werden innerhalb der Prozedur definiert und sind nur dort gültig. Wenn die Prozedur beendet wird, werden sie gelöscht. Verwenden Sie eine lokale Variable, wenn diese nicht außerhalb der Prozedur benötigt wird. Sie sind das Mittel der Wahl für Zwischenergebnisse.

Lokale Variablen in einem Testfall
In einem Testfall können lokale Variable entweder während der Durchführung des Testfall angelegt werden oder über die entsprechende Tabelle in den Details des Testfall Knotens. Wenn Sie in einem Testfall mehrfach den gleichen Wert verwenden, ist es sinnvoll, diesen einmalig einer Variablen zuzuweisen und dann die Variable zu verwenden. Dies erhöht die Wartbarkeit. Auch für Zwischenergebnisse sollte man lokale Variablen verwenden.

Globale Variablen
Wenn globale Variablen einmal erstellt wurden, existieren sie, bis sie entweder explizit gelöscht werden oder bis QF-Test beendet wird. Auch Stopps und die erneute Ausführung von Tests ”überleben” sie. Verwenden Sie sie für

Werte, die in mehreren Testfällen genutzt werden. Ein Beispiel ist die Variable `client`, die im Vorbereitung Knoten beim Start der Applikation angelegt wird. Um sie wieder loszuwerden, wählen Sie den Menüpunkt Wiedergabe→Globale Variablen löschen. Auch beim Beenden von QF-Test werden sie gelöscht.

Kommandozeilenparameter

Variablen, die über Kommandozeile gesetzt werden, sind im Batch-Modus sinnvoll, wenn Sie mehrere Batch-Läufe mit unterschiedlichen Werten durchführen wollen. Kommandozeilenparameter gelten während der gesamten Laufzeit des Batch-Laufs. Ein typisches Beispiel ist die variablengesteuerte Ausführung auf verschiedenen Browsern. Variablen können über den Kommandozeilen-Parameter `-variable` spezifiziert (vgl. Kapitel 'Kommandozeilenargumente' im Handbuch).

Testsuite-Variablen

Testsuite-Variablen können von allen Testfällen verwendet werden. Ihr Verwendungszweck entspricht dem von globalen Variablen, nur dass sie im Batch-Modus durch Variablen in der Kommandozeile überschrieben werden können.

Standardwerte (Sekundärstapel)

Sie können Standardwerte für die Variablen von Prozeduren, Testfällen und Testfallsätzen definieren. Diese kommen zum Zug, wenn keine Variable mit dem gleichen Namen auf einer höheren Ebene definiert wurde.

Kapitel 25

Die Standardbibliothek (Win)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Die Standardbibliothek'
https://www.qftest.com/de/yt/tutorial-7.html

QF-Test stellt eine gewisse Anzahl an Knotentypen bereit, die für die Testerstellung genutzt werden können. Wenn Sie Funktionalität benötigen, die darüber hinausgeht, können Sie diese mittels Skript-Knoten implementieren. Um Ihnen die Arbeit zu erleichtern, wurden viele Funktionen, die häufig benötigt werden, bereits in Prozeduren implementiert und werden in einer Standard-Prozeduren-Bibliothek mit QF-Test ausgeliefert.

Wenn Sie also eine Aufgabenstellung haben, die nicht über die bereitgestellten Knoten gelöst werden kann, sollten Sie zunächst einmal in der Standardbibliothek forschen, ob Sie dort eine passende oder ähnliche Lösung finden. Wenn Sie eine ähnliche Lösung finden, kopieren Sie einfach die vorhandene Standardprozedur und passen sie Sie gemäß Ihrer Bedürfnisse an. Informationen zum Arbeiten mit Skripten erhalten Sie im Handbuch, Kapitel 12 "Skripting".

Die Bibliothek ist in der Datei `qfs.qft` enthalten und ist Teil der QF-Test Distribution. Da sie mit jeder QF-Test Version weiterentwickelt wird, ist es nicht ratsam, Änderungen in der ausgelieferten Standardbibliothek vorzunehmen, sondern bei Bedarf eine Prozedur in eine eigene Testsuite zu kopieren und dort anzupassen.

Um die Bibliothek `qfs.qft` verwenden zu können, muss sie im "Testsuite" Wurzelknoten Ihrer Suite in den "inkludierten Dateien" eingebunden werden. Bei neuen Testsuiten ist dies automatisch der Fall.

Aktion

- Wählen Sie den 'Testsuite'-Wurzelknoten Ihrer Testsuite aus.
- Überprüfen Sie in den Details des 'Testsuite'-Wurzelknoten, dass `qfs.qft` im Attribut "Inkludierte Dateien" aufgeführt ist.
- Fügen Sie `qfs.qft` zu dieser Liste dazu, falls es noch nicht enthalten ist.

Hinweis Eine Pfadangabe ist nicht notwendig, da das `include` Verzeichnis von QF-Test automatisch im Bibliothekspfad (siehe auch Referenzteil des Handbuchs) enthalten ist.

Im Folgenden beschreiben wir eine Auswahl der am häufigsten benötigten Standardprozeduren. Eine vollständige HTML-Dokumentation der Standardbibliothek finden Sie unter dem Menüpunkt Hilfe→Standardbibliothek qfs.qft...

25.1 Erforschen der Standardbibliothek

Zusätzlich zum Einfügen von Prozeduraufrufen aus der Standardbibliothek ist es hilfreich, einen Blick darauf zu werfen, wie Funktionen implementiert und organisiert sind.

Aktion

- Öffnen Sie die Bibliothek selbst, also die Suite `qfs.qft`, die sich im Verzeichnis `qftest-9.0.0/include` Ihrer QF-Test Installation befindet.

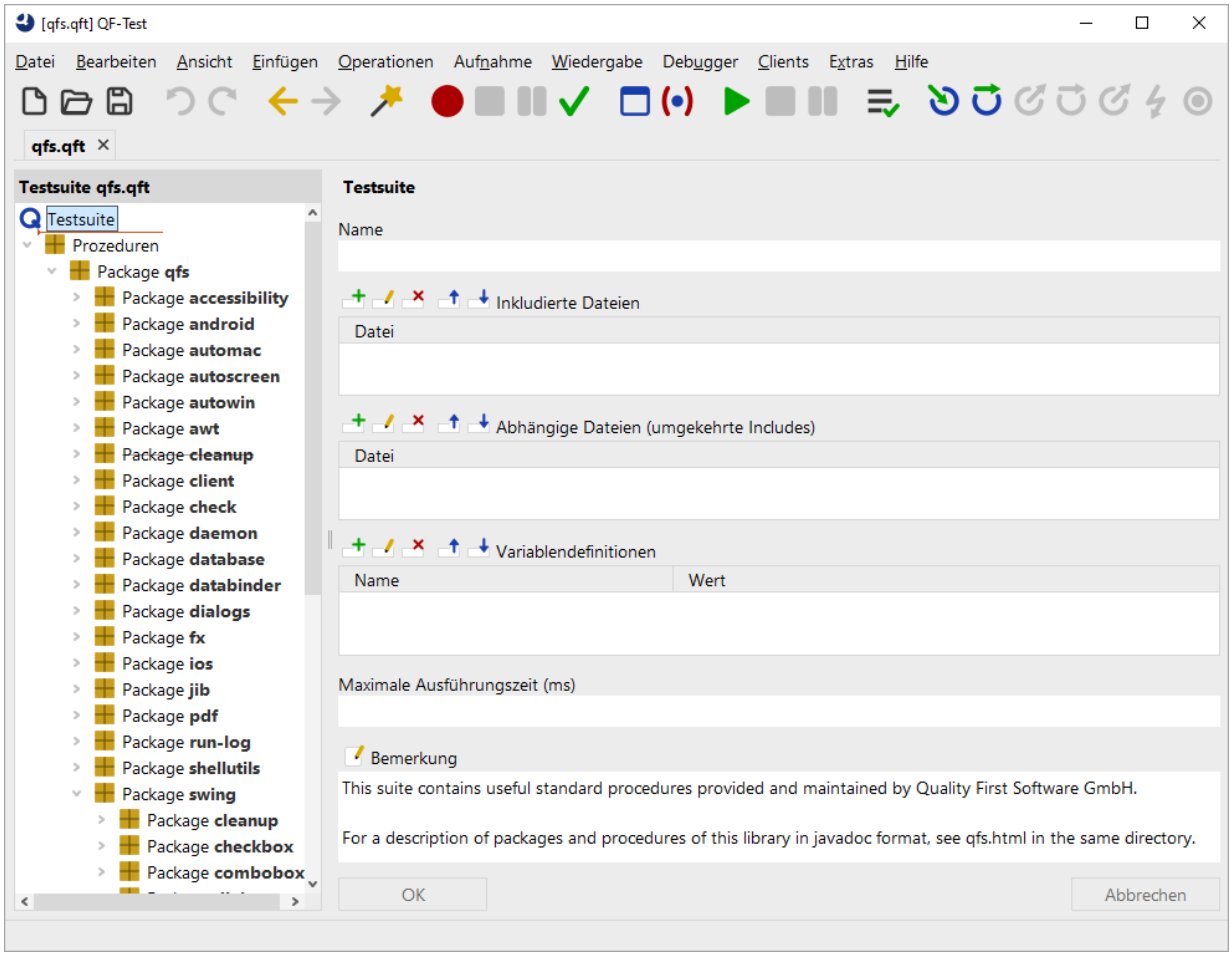

Abbildung 25.1: Die Standardbibliothek

Sie sehen ein Haupt-Package `qfs`, das die spezifischen Packages umschließt. Das `qfs` Package hilft dabei, die Prozeduren leicht als solche der Standardbibliothek zu identifizieren.

In fast allen Prozeduren unserer Bibliothek werden Sie die Verwendung der Variable `$(client)` bemerken. Dies ist ein Standardmechanismus, um Testsuiten unabhängig von einem spezifischen SUT zu gestalten. Für die Benutzung der Standardbibliothek wird vorausgesetzt, dass ein gültiger Wert für `$(client)` gesetzt wird, bevor eine ihrer Prozeduren verwendet werden kann.

25.2 Ausgewählte Packages und Prozeduren

Wir werfen nun einen genaueren Blick auf ein paar ausgewählte Packages und Prozeduren der Standardbibliothek.

25.2.1 Das Run-log Package

Das Package `qfs.run-log` enthält Prozeduren, um Meldungen in das Protokoll zu schreiben.

Hier sehen Sie die Liste von verfügbaren Prozeduren innerhalb des Packages:

- **logError** Schreibt eine Fehlermeldung ins Protokoll.
- **logWarning** Schreibt eine Warnung ins Protokoll.
- **logMessage** Schreibt eine Meldung ins Protokoll.

25.2.2 Das Run-log.Screenshots Package

Das `qfs.run-log.screenshots` Package enthält Prozeduren, die Bildschirmabbilder ins Protokoll schreiben und einige Hilfsprozeduren.

Hier sehen Sie die Liste von verfügbaren Prozeduren innerhalb des Packages:

- **getMonitorCount** Liefert die Anzahl der an den Computer angeschlossenen Monitore.
- **logScreenshot** Schreibt ein Bildschirmabbild des aktuellen Monitors ins Protokoll.
- **logImageOfComponent** Schreibt ein Bildschirmabbild einer Komponente ins Protokoll.

- **logScreenshotOfMonitor** Schreibt ein Bildschirmabbild eines angegebenen Monitors ins Protokoll.

25.2.3 Das Shellutils Package

Das `qfs.shellutils` Package beinhaltet Prozeduren für die wichtigsten Shell-Kommandos.

Hier sehen Sie die Liste von verfügbaren Prozeduren innerhalb des Packages:

- **copy** Kopiert eine angegebene Datei oder ein Verzeichnis an eine neue Stelle.
- **deleteFile** Löscht eine angegebene Datei.
- **exists** Prüft, ob eine angegebene Datei oder ein Verzeichnis existiert.
- **getBasename** Gibt den Dateinamen einer Datei zurück.
- **getParentDirectory** Gibt die Verzeichnisstruktur einer Datei zurück.
- **mkdir** Erzeugt ein Verzeichnis. Noch nicht existierende Verzeichnisse werden angelegt.
- **move** Verschiebt eine angegebene Datei oder ein Verzeichnis.
- **touch** Erzeugt eine Datei.
- **removeDirectory** Löscht ein angegebenes Verzeichnis.

25.2.4 Das Utils Package

Das Package `qfs.utils` enthält nützliche Prozeduren für häufig auftretende Anforderungen der Testentwicklung.

Hier sehen Sie einige Prozeduren des Packages:

- **getDate** Gibt einen String zurück, der ein Datum enthält. Standardmäßig wird das aktuelle Datum zurückgegeben. (Andere Daten sind konfigurierbar.)
- **getTime** Gibt einen String zurück, der eine Zeit enthält. Standardmäßig wird die aktuelle Zeit zurückgegeben. (Andere Zeiten sind konfigurierbar.)
- **logMemory** Schreibt den aktuellen Speicherverbrauch ins Protokoll.
- **printVariable** Gibt den Inhalt einer spezifizierten Variable auf der Konsole aus.

- **printMessage** Gibt den Inhalt einer angegebenen Nachricht auf der Konsole aus.
- **writeMessageIntoFile** Schreibt einen angegebenen String in eine angegebene Datei.

25.2.5 Das Database Package

Das Package `qfs.database` enthält nützliche Prozeduren, um mit Datenbanken zu interagieren.

Bitte beachten Sie, dass die jar-Datei mit dem Datenbanktreiber vor dem Start von QF-Test ins `qftest` Pluginverzeichnis kopiert werden muss.

Für weitere Informationen über den Aufbau einer Datenbankverbindung kontaktieren Sie bitte einen Entwickler oder werfen Sie einen Blick auf www.connectionstrings.com.

Die wichtigsten Prozeduren sind:

- **executeSelectStatement** Führt einen angegebenen SQL-Select-Befehl aus. Das Ergebnis wird zum einen in die globale Variable ”resultRows” des Jython Variablenstacks geschrieben und ist somit in jedem Jython Skript verfügbar. Zum anderen wird das Ergebnis auch in eine Gruppenvariable mit dem Standardnamen ”resultGroup” geschrieben und ist somit direkt von QF-Test Knoten aus ansprechbar.
- **executeStatement** Führt einen angegebenen SQL Befehl aus. Hier kann jedes beliebige SQL Kommando ausgeführt werden.

25.2.6 Das Check Package

Das `qfs.check` Package enthält Prozeduren, die Checks ausführen.

Die wichtigsten Prozeduren sind:

- **checkEnabledStatus** Überprüft, ob eine Komponente en- bzw. disabled ist. Im Fehlerfall wird von der Prozedur ein entsprechender Fehler ins Protokoll geloggt.
- **checkSelectedStatus** Überprüft, ob eine Komponente selektiert bzw. nicht selektiert ist. Im Fehlerfall wird von der Prozedur ein entsprechender Fehler ins Protokoll geloggt.
- **checkText** Überprüft den Text einer Komponente. Im Fehlerfall wird von der Prozedur ein entsprechender Fehler ins Protokoll geloggt.

25.2.7 Das Databinder Package

Das Package `qfs.databinder` enthält Prozeduren zur Ausführung innerhalb eines Datentreiber Knotens, um Daten für datengetriebenes Testen zu binden.

Die wichtigsten Prozeduren sind:

- **bindList** Bindet eine Liste von Werten an eine Variable. Die Werte sind durch Leerzeichen oder das als Parameter übergebene Trennzeichen getrennt.
- **bindSets** Bindet Sätze von Werten an einen Satz von Variablen. Die Sätze von Werten sind durch Zeilenumbrüche getrennt, die Werte innerhalb eines Satzes durch Leerzeichen oder das als Parameter übergebene Trennzeichen.

Kapitel 26

Ablaufsteuerung (Win)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Ablaufsteuerung'
https://www.qftest.com/de/yt/tutorial-8.html

Die zwei wichtigsten Kontrollstrukturen von QF-Test sind Schleifen und die bedingte Ausführung von Knoten. Schleifen können über zwei verschieden Knoten implementiert werden: While und Schleife Knoten. If, Elseif und Else Knoten stehen für die bedingte Ausführung von Knoten zur Verfügung.

26.1 If - else

If Knoten kennen Sie bereits aus der Vorbereitung Sequenz im Kapitel Starten der Anwendung[214]. Sehen wir uns diesen nun etwas genauer an.

Abbildung 26.1: Setup Sequenz mit If/Elseif Knoten

Über einen If Knoten können Sie steuern, ob bestimmte Knoten ausgeführt werden oder nicht. In unserem Fall geht es um den Start des SUT. Zunächst müssen wir herausfinden, ob die Applikation bereits läuft. Dies geschieht über den Warten auf Client

Knoten, der als Ergebnis entweder `true` (wahr) oder `false` (falsch) in die Variable `isSUTRunning` schreibt.

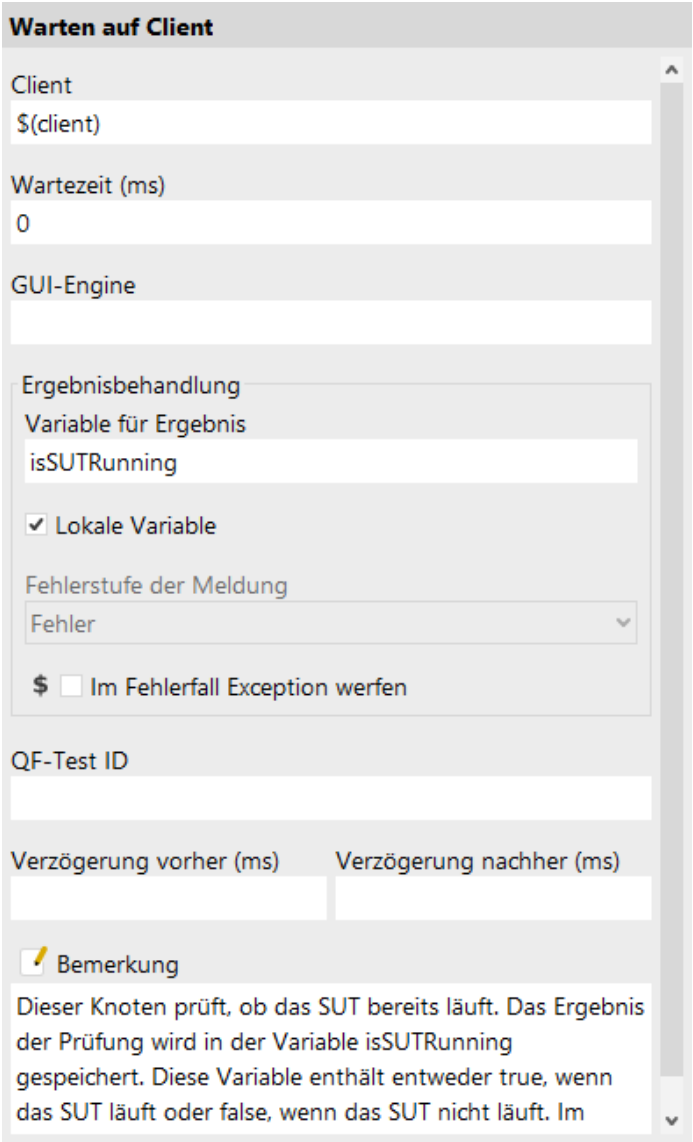

Abbildung 26.2: Warten auf Client setzt die Variable "isSUTRunning" mit dem Ergebnis

Der If Knoten wertet die Ergebnisvariable `isSUTRunning` im 'Bedingung' Attribut aus. Da auf den Wert der Variablen zugegriffen werden soll, wird die Syntax `$()` verwendet (vgl. Hinweis zu Variablensyntax in Kapitel Abschnitt 24.1 (266)).

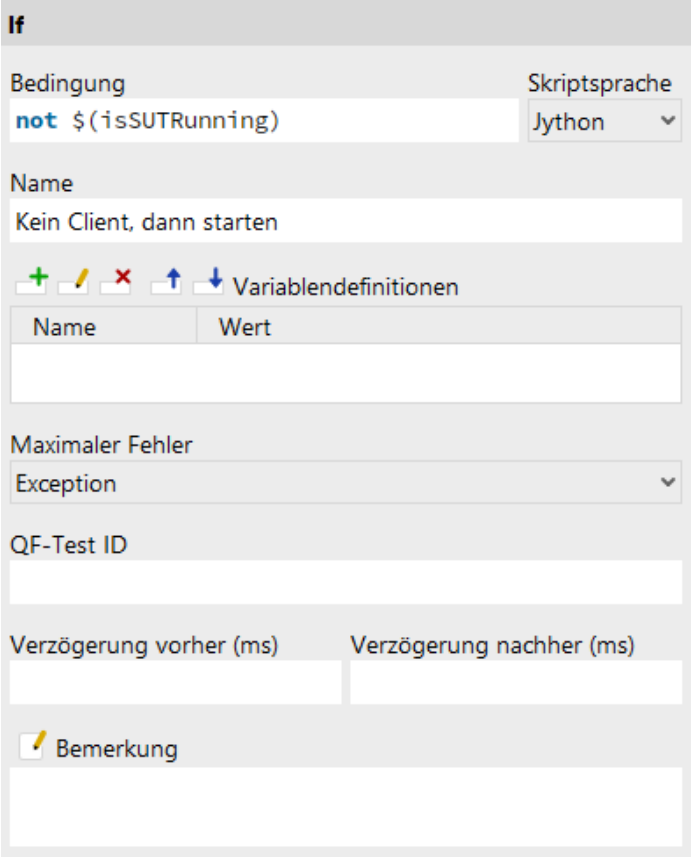

Abbildung 26.3: Der If Knoten wertet die Variable aus

Je nachdem, ob die Applikation bereits läuft, führt QF-Test die im If Knoten enthaltenen Knoten aus oder nicht.

Aktion

- **Beenden Sie das CarConfig Demo**, falls es läuft.
- **Führen Sie den Vorbereitung Knoten mit Einzelschritten aus.**
- **Führen Sie den Vorbereitung Knoten nochmals mit Einzelschritten aus** während das CarConfig Demo läuft.

In der Variablendefinition-Tabelle können Sie den Wert der Variablen `isSUTRunning` prüfen. Beim ersten Mal ist der Wert `false` und damit die Bedingung `not $(isSUTRunning)` wahr, d.h. die Knoten für den SUT-Start werden ausgeführt. Beim zweiten Mal ist der Wert `true` und damit die Bedingung `false`. Die Knoten im If Knoten werden übersprungen.

Hinweis

Im ersten Knoten der Vorbereitung werden If Knoten verwendet, um abhängig vom Be-

triebssystem den zu verwendenden Win-Browser in einer globalen Variable zu hinterlegen. Der besseren Lesbarkeit halber werden hier nur If Knoten verwendet. Es wäre ebenso möglich mit Elseif und Else Knoten zu arbeiten. In einem Elseif Knoten wird eine weitere Bedingung formuliert, die dann geprüft wird, wenn die Bedingung im If Knoten nicht zutrifft. Die Kindknoten des Else Knoten werden nur dann ausgeführt, wenn weder die Bedingung des If noch die der Elseif Knoten zutreffen.

Bei der Prüfung des Betriebssystems wird direkt auf eine QF-Test Variable zugegriffen. QF-Test speichert die Betriebssysteminformation in einer Gruppenvariable ab, wobei die Gruppe `qftest` und die Variablen `linux`, `macos` oder `windows` heißen. Die Syntax für den Zugriff auf Gruppenvariablen ist `${group:varname}`, z.B. `${qftest:windows}`.

26.2 Schleifen

QF-Test stellt zwei Knotentypen für die Implementierung von Schleifen zur Verfügung:

- Schleife Knoten führen ihre Kindknoten so oft aus, wie angegeben ist. Man kann die Schleife jedoch über einen Break Knoten jederzeit verlassen.
- While Knoten führen ihre Kindknoten so oft aus, bis die angegebene Bedingung nicht mehr gegeben ist. Derartige Schleifen können ebenfalls über einen Break Knoten jederzeit verlassen werden.

Hinweis

Schleife Knoten enden auf jeden Fall nach der angegeben Anzahl von Wiederholungen. Bei While Knoten muss man jedoch selbst dafür sorgen, dass die Ausführung irgendwann endet, indem die Bedingung falsch wird. Ansonsten kommt es zur Endlosschleife. Im interaktiven Modus können Sie in so einem Fall einfach die Pausetaste ⏸ drücken. Im Batch-Modus, d.h. wenn Sie QF-Test mit dem Kommandozeilenparameter `-batch` starten um die angegebene Testsuite ohne die QF-Test Benutzeroberfläche auszuführen, müssen Sie dann jedoch den QF-Test Prozess "abschießen".

In der folgenden Übung wollen wir einen Testfall implementieren, der prüft, ob eine bestimmte Zeile in der Tabelle des CarConfig Demos angezeigt wird.

Die im Testfall durchgeführten Aktionen sind:

- Anzahl Tabellenzeilen bestimmen.
- Über alle Zeilen iterieren und prüfen, ob die Zeile passt.
- Wenn die Zeile gefunden wurde, die Schleife abbrechen.
- Falls die Zeile nicht gefunden wurde, einen Fehler ins Protokoll schreiben.

Bitte beginnen Sie mit der Aufnahme eines Checks auf die zu suchende Zeile:

Aktion

- **Aktivieren Sie den Check-Aufnahmemodus** über "Checks aufnehmen" ✓ .
- **Führen Sie einen Rechtsklick auf eine Tabellenzeile** im CarConfig Demo aus und wählen Sie den Menüpunkt Zeile aus dem Popup-Menü.
- **Beenden Sie die Aufnahme** über "Aufnahme beenden" ■ .
- **Ändern Sie den Namen der aufgenommenen Sequenz** z.B. in `Zeile prüfen`.
- **Wandeln Sie die Sequenz in einen Testfall um:** Rechtsklick auf den Sequenz Knoten und Auswahl des Untermenüpunkts Knoten konvertieren in→Testfall im Popup-Menü.

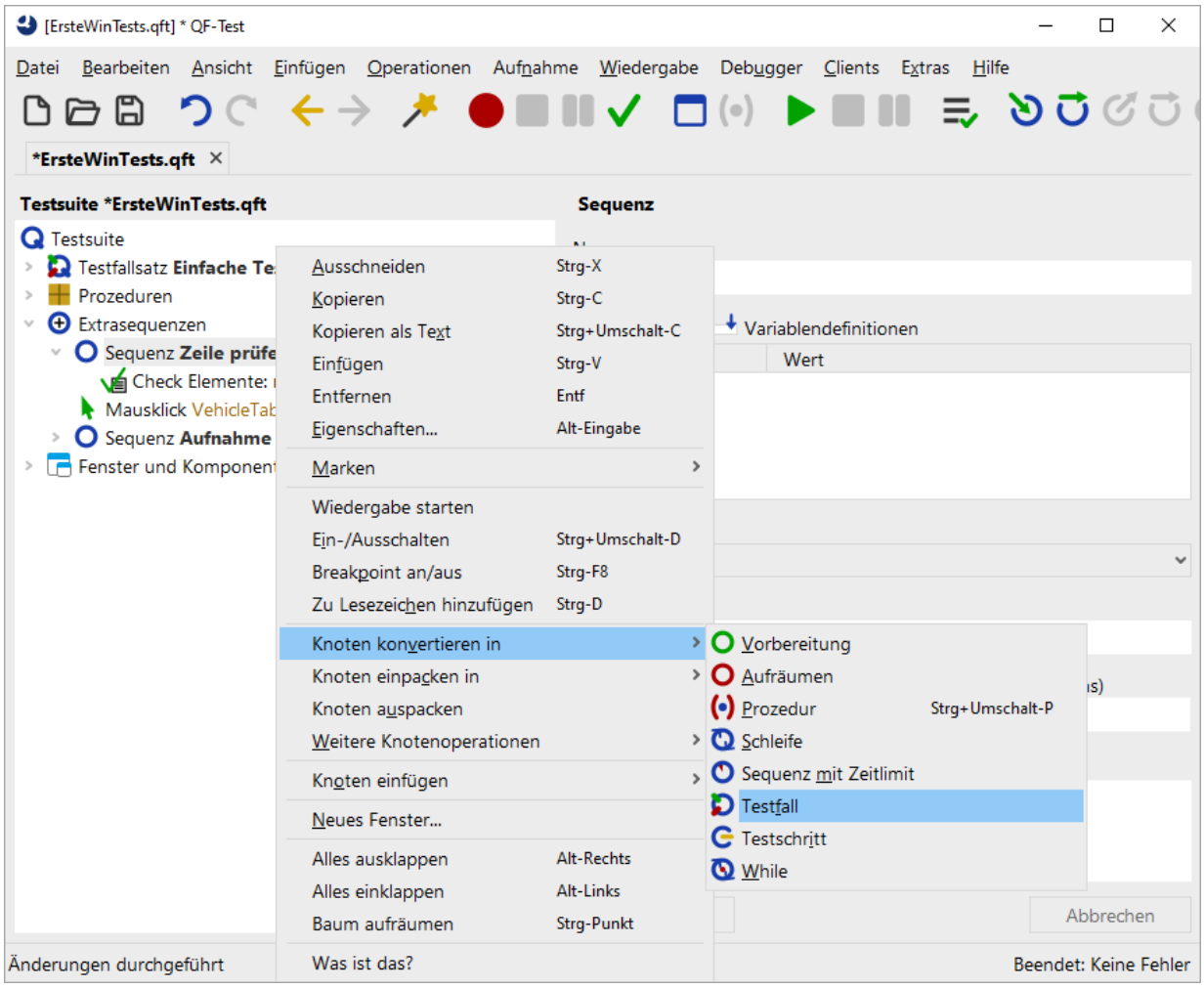

Abbildung 26.4: Knoten konvertieren

In QF-Test können Sie sehr effizient Knoten hinzufügen, indem Sie einen Knoten in einen anderen einpacken:

Aktion

- Öffnen Sie den Testfall Knoten und **packen Sie den aufgenommenen Check Knoten in eine Schleife** indem Sie rechts auf den Knoten klicken und in dem sich öffnenden Popupmenü den Punkt Knoten einpacken in→Schleife auswählen.

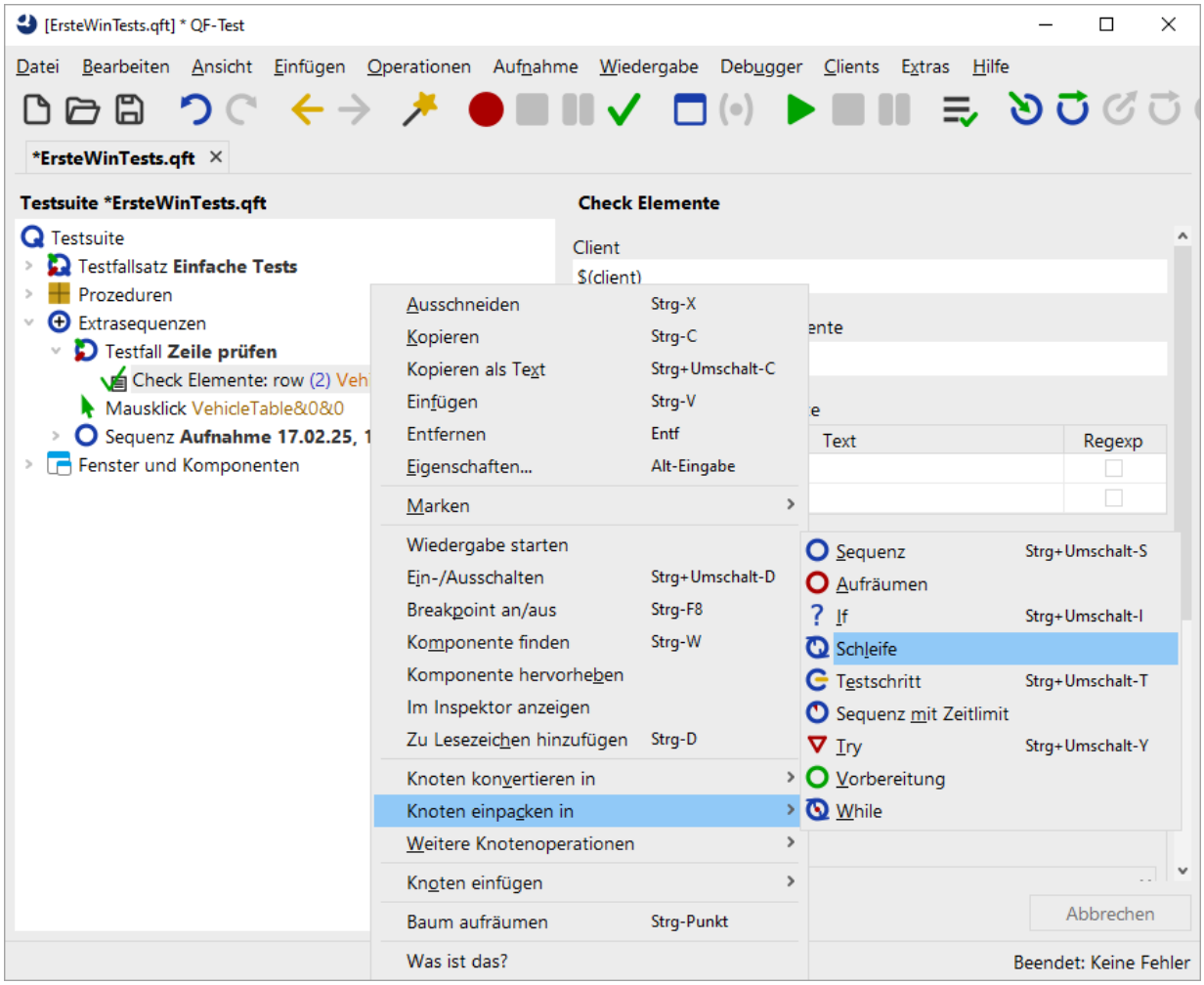

Abbildung 26.5: Knoten einpacken

QF-Test ermittelt dynamisch, in welche Knoten ein Knoten eingepackt werden kann und bietet nur diese zur Auswahl an. Entsprechend kann es passieren, dass Sie ”Schleife” im Untermenü nicht finden. Sie sollten dann prüfen, ob Sie den Rechtsklick auf den richtigen Knoten ausgeführt haben. Dasselbe gilt für die Operationen ”Knoten konvertieren in” und ”Knoten einfügen”.

Als nächstes setzen wir den Wert für das Attribut 'Anzahl Wiederholungen' des Schleife Knoten. Dazu müssen wir bestimmen, wie viele Zeilen die Tabelle hat. Es gibt keinen Knoten, der diese Operation direkt ausführen kann. Allerdings gibt es eine derartige Prozedur in der im letzten Kapitel besprochenen Standardbibliothek. Diese befindet sich im Package `qfs.win.table` und heißt `getRowCount`.

Aktion

- **Selektieren Sie den Testfall Knoten und drücken Strg-A**.
- **Klicken Sie die Schaltfläche "Prozedur auswählen"** links neben der Überschrift 'Name der Prozedur'.
- **Wählen Sie den Reiter 'qfs.qft'** im 'Prozedur auswählen' Dialog.
- **Navigieren Sie zu 'getRowCount' im Package 'qfs.win.table'**
- **Drücken Sie 'OK'** um die Prozedur auszuwählen.
- **Drücken Sie 'OK'** um den 'Prozeduraufruf' Dialog zu schließen.

Das Hinzufügen einer Prozedur über Strg-A wurde in Manuelle Erstellung von Prozeduren(244) beschrieben. Dort finden Sie auch Screenshots zur Aktion.

Aktion

- **Fügen Sie eine Variable** mit dem Namen `Zeilen` **im Attribut 'Variable für Rückgabewert' ein**.
- **Ändern Sie den Standardwert für `id` in** der Variablendefinitionen-Tabelle auf die QF-Test Komponenten-ID der Tabelle **`VehicleTable`**.
- Klicken Sie **OK**.
- **Wählen Sie den 'Schleife'-Knoten**.
- Im Attribut 'Anzahl Wiederholungen' des Schleife Knotens **tragen Sie eine Referenz auf die Variable `$(Zeilen)` ein**.
- **Tragen Sie den Namen der Zählervariable**, z.B. `i`, im entsprechenden Attribut des Schleife Knotens **ein**.
- Klicken Sie **OK**.

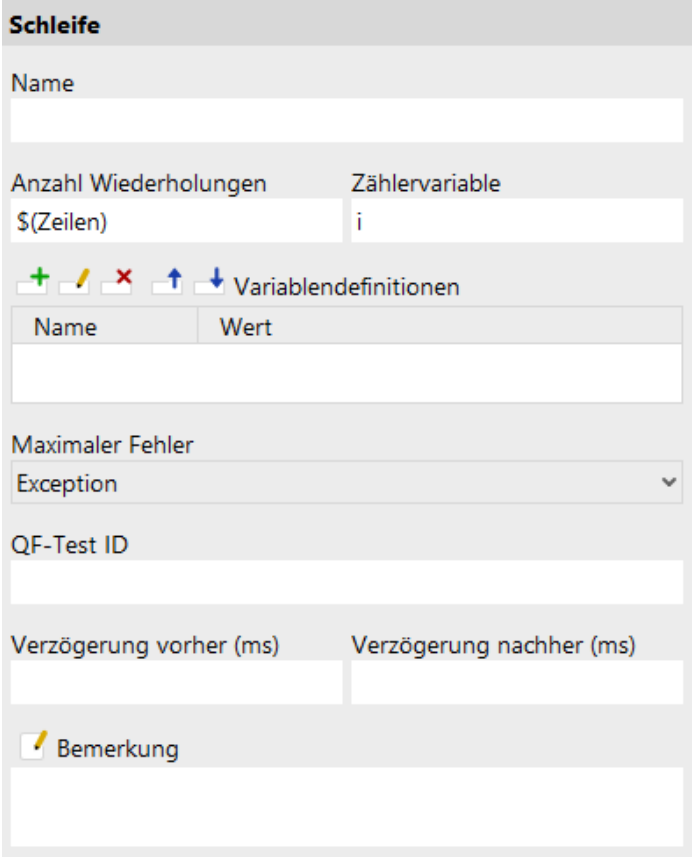

Abbildung 26.6: Details eines Schleife Knotens

In den Details des Check Knotens tragen wir nun in der QF-Test ID der Komponente statt des aufgenommenen Zeilenindex eine Referenz auf die Zählervariable ein und setzen eine Ergebnisvariable. Außerdem fügen wir einen If Knoten unter dem Check Knoten hinzu, der das Ergebnis auswertet und die Schleife über einen Break Knoten verlässt, wenn die entsprechende Zeile gefunden wurde.

Aktion

- **Öffnen Sie den Schleife Knoten**.
- **Selektieren Sie den Check Knoten**.
- **Ändern Sie den aufgenommenen Zeilenindex** der QF-Test ID der Komponente in Zählervariable `$(i)`. Die QF-Test ID der Komponente sollte nun `VehicleTable@Modell&$(i)` lauten.
- **Tragen Sie den Variablennamen `ZeileGefunden` in das Attribut 'Variable für Ergebnis' ein** und klicken **OK**.

- Führen Sie einen **Rechtsklick auf den Check Knoten** aus und wählen Sie aus dem Popup-Menü den Unterpunkt Knoten einfügen→Ablaufsteuerung→Break aus.
- **Drücken Sie 'OK'** im 'Break' Dialog.
- **Packen Sie den Break Knoten in einen If Knoten** mittels des Tastaturkürzels Strg-Umschalt-I (Sie können natürlich auch über das Menü gehen).
- In den Details des 'If'-Knotens **tragen Sie `$(ZeileGefunden)` im Attribut 'Bedingung' ein** und klicken **OK**.

Die Variable `ZeileGefunden` wird vom Check Knoten entweder auf den Wert 'true' oder auf den Wert 'false' gesetzt, so dass wir im Attribut 'Bedingung' des If Knoten nur die Referenz auf die Variable `$(ZeileGefunden)` einzutragen brauchen.

In den nächsten Schritten wollen wir einen Else Knoten als letzten Knoten im Schleife Knoten einfügen. Er wird nur ausgeführt, wenn die Schleife so oft wie angegeben ausgeführt wurde, was in unserem Fall bedeutet, dass die Variable `ZeileGefunden` nie wahr wurde, weil die Zeile nicht gefunden wurde.

Aktion

- **Schließen Sie den If Knoten**, falls dies nicht bereits der Fall ist. Dies ist wichtig, da sonst der Else Knoten zum If Knoten und nicht zum Schleife Knoten gehören würde.
- Führen Sie einen Rechtsklick auf den If Knoten aus und wählen Sie auf dem Popup-Menü den Unterpunkt **Knoten einfügen→Ablaufsteuerung→Else**.
- **Klicken Sie im 'Else' Dialog 'OK'**.
- **Öffnen Sie den Else Knoten**.
- **Fügen Sie** aus der Standardbibliothek **die Prozedur `logError`** aus dem Package `qfs.run-log` wie oben beschrieben **ein**.
- In der 'Variablendefinitionen' Tabelle **tragen Sie `Zeile nicht gefunden`** als Wert der Zeile `message` **ein.**
- **Tragen Sie `true`** als Wert der Zeile `withScreenshots` **ein.**
- Drücken Sie **OK**.

Wenn Sie die Tests im Batch-Modus ausführen, sind Screenshots eine gute Unterstützung bei der Fehleranalyse. Da aber eine große Zahl Screenshots sehr große Protokolldateien erzeugen würden, ist der Standardwert für `withScreenshots false`.

Nun bleibt nur noch, den Testfall mit Vorbereitung und Aufräumen Knoten zu vervollständigen und ihn in den oberen Teil der Testsuite zu verschieben.

Aktion

- **Kopieren Sie die Vorbereitung und Aufräumen Knoten** aus 'Testset: Einfache Tests' in den neuen Testfall als ersten und letzten Knoten.
- **Verschieben Sie den Testfall** aus dem Bereich Extrasequenzen in den oberen Bereich der Testsuite hinter den Knoten 'Testset: Einfache Tests'.

Damit würde der neue Testfall wie folgt aussehen:

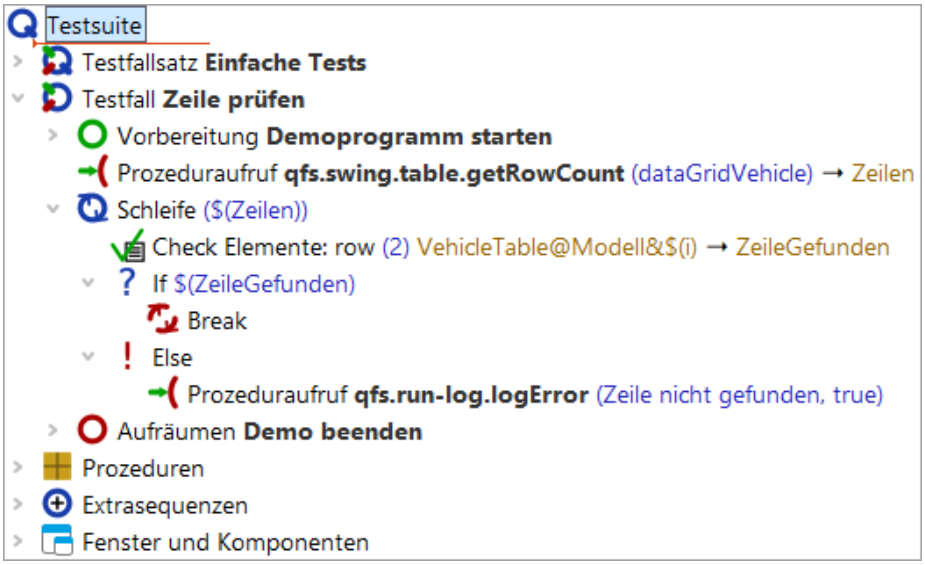

Abbildung 26.7: Der neue Testfall

Aktion

- **Führen Sie den neuen Testfall aus.**

Er sollte ohne Fehler laufen.

Aktion

- **Ändern Sie** nun in den Details des Check Elemente Knotens **den Namen des Fahrzeugs zum Beispiel auf `Falscher Wert`**.

Check Elemente

Client
$(client)

QF-Test ID der Komponente
VehicleTable@Modell&$(i)

Elemente

	Text	Regexp
0	Falscher Wert	☐
1	15.000,00 €	☐

Name des Check-Typs
row

Wartezeit (ms)

Ergebnisbehandlung
Variable für Ergebnis
ZeileGefunden

☐ Lokale Variable

Fehlerstufe der Meldung
Fehler

$ ☐ Im Fehlerfall Exception werfen

Name

QF-Test ID

Verzögerung vorher (ms) Verzögerung nachher (ms)

Bemerkung

Abbildung 26.8: Details eines Check Elemente Knoten

Aktion

- **Führen Sie den Testfall nochmals aus.**

Nun sollte der Test den Else Knoten der Schleife ausführen und eine Fehlermeldung anzeigen.

Kapitel 27

Nun ist es Zeit, Ihre eigene Anwendung zu starten (Win)

Video

Dieses Kapitel ist auch als Video verfügbar unter

'Nun ist es Zeit, Ihre eigene Anwendung zu starten'
https://www.qftest.com/de/yt/tutorial-9.html

Nachdem wir so viel Zeit mit all den Beispielanwendungen verbracht haben, sind Sie nun wirklich bereit, Ihre eigene Applikation zu starten (falls Sie dies nicht schon zwischendurch getan haben).

Der Schnellstart-Assistent, welcher über das Menü Extras→Schnellstart-Assistent... erreichbar ist, hilft Ihnen bei dieser Aufgabe. Folgen Sie einfach den Schritten innerhalb des Assistenten, um eine passende Startsequenz zu erzeugen. Bitte schauen Sie auch ins Handbuch Kapitel 3 "Schnellstart".

Es ist an der Zeit, das Gelernte in die Tat umzusetzen - kurze Sequenzen von Events und Checks aufzunehmen, Prozeduren zu erzeugen etc., um eine eigene Testbibliothek aufzubauen.

Damit endet der Basisteil in diesem Tutorial.

Teil IV

Mobile-Anwendungen testen mit QF-Test

Aktuell gibt es noch kein spezielles Tutorialkapitel zum Testen von Android und iOS-Anwendungen. Es gibt aber ausführliche Handbuchkapitel zum Testen von Android- und iOS-Anwendungen, welche ausführlich beschreiben, wie Sie die jeweilige Testumgebung aufsetzen und mit dem Automatisieren Ihrer Tests beginnen können. Bitte schauen Sie sich dieses unbedingt an!

Die Arbeitstechniken, die in den anderen Kapiteln dieses Tutorials beschrieben werden, sind grundsätzlich auch für Android- und iOS-Testen gültig. So können Sie gerne einen Blick z.B. in das Kapitel Java-GUIs testen mit QF-Test(2) werfen, um Schritt für Schritt zu lernen, wie sich die QF-Test Features am besten zur Testautomatisierung einsetzen lassen.

Im Teil V(305) werden weiterführende Funktionalitäten von QF-Test erklärt, die für Tests aller unterstützter GUI-Technologien genutzt werden können.

Teil V

Weiterführende Features von QF-Test

Dieser Abschnitt des Tutorials beschreibt Features von QF-Test, welche für fortgeschrittene Benutzer interessant sind.

Die folgenden Kapitel verwenden die Java-Variante der CarConfigurator Demoanwendung, die Sie bereits aus dem Basisteil des Tutorials kennen. Der CarConfigurator ist in Java/Swing implementiert, die Konzepte allerdings sind für jede unterstützte Technologie anwendbar.

Für jedes Kapitel gibt es auch spezielle Testsuiten, damit Sie die Themen jedes Kapitels einzeln nachverfolgen können. Diese Dateien finden Sie unter `qftest-9.0.0/doc/tutorial/advanced-demos/de`.

Kapitel 28

Datengetriebenes Testen: Einen Test case mit unterschiedlichen Testdatensätzen starten

Dieses Kapitel erklärt, wie man datengetriebenes Testen mit QF-Test verwirklichen kann.

Sie finden unter `qftest-9.0.0/doc/tutorial/advanced-demos/de/datadrivenTesting.qft` die hier gezeigten Testfälle.

In der zweiten mitgelieferten Testsuite `qftest-9.0.0/doc/tutorial/datadriver.qft` finden Sie weitere Beispiele, wie das Auslesen einer Excel-Datei oder einer Schleife um Testfälle.

Bitte achten Sie darauf, dass Sie alle Testsuiten vorher in einen projektspezifischen Ordner kopieren und diese dort modifizieren.

28.1 Situation

Die Anwender des CarConfigurators können unterschiedliche Rabattstufen gewähren. Der Testdesigner hat drei Rabattstufen definiert, die getestet werden sollen.

Diese Rabattstufen sind 0%, 10% und 15%.

Der Ablauf, um einen Rabatt zu gewähren, ist für jede dieser drei Rabattstufen derselbe. Daher können wir denselben Testfall benutzen, um diese zu testen. Der einzige Unterschied ist der eingegebene Rabattsatz und der zu prüfende Preis. Wir sollten denselben Testfall für jede Rabattstufe benutzen, um Seiteneffekte zu vermeiden, die bei unterschiedlichen Implementierungen der Testfälle auftreten können. Darüber hinaus

sparen wir uns auch den Implementierungsaufwand zum Erstellen mehrerer Testfälle. Der logische Testfall, d.h. die Schritte des Testfalles, sehen folgendermaßen aus:

1. Starten des SUTs.
2. Ein Modell auswählen.
3. Den Rabatt eingeben.
4. Prüfen, ob der Rabatt für die Preiskalkulation herangezogen wurde.
5. Das SUT stoppen.

Die folgenden Abschnitte zeigen nun die Implementierung dieses Szenarios.

28.2 Die traditionelle Methode für datengetriebenes Testen

In QF-Test steht ein Testfall für einen Testablauf zusammen mit einem speziellen Testdatensatz. Wenn man zwei Testdatensätze testen möchte, so muss man zwei Testfälle erstellen. Diese Testfälle können auch in einem Testfallsatz zusammengefasst werden.

Bei der konventionellen Lösung implementiert man also einen Testfall pro Rabattstufe. Das sieht dann so aus:

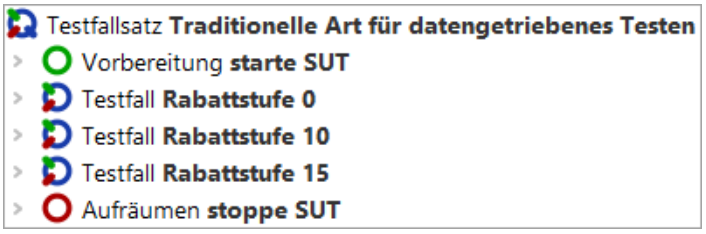

Abbildung 28.1: Konventionelle Methode für datengetriebenes Testen

Wir sehen für jede der drei Rabattstufen genau einen Testfall Knoten. Diese drei Knoten sind in einem Testfallsatz zusammengefasst. Der Testfallsatz Knoten beinhaltet auch die Vorbereitung und die Aufräumen Knoten, welche das SUT vor jedem Testfall starten und nach jedem Testfall stoppen. Dies geschieht, um sicherzustellen, dass die Vorbedingungen jedes einzelnen Testfalles dieselben sind. Wenn Sie die SUT nicht nach jedem Testfall neu starten wollen, dann können Sie die drei Testfälle in einem weiteren Testfallsatz zusammenfassen, wie Sie hier sehen können:

Abbildung 28.2: Konventionelle Methode mit einem verschachtelten Testfallsatz

Das Kapitel Abhängigkeiten: Automatisches Sicherstellen der korrekten Vorbedingungen jedes Testfalles[316] zeigt Ihnen eine elegantere und effizientere Art, Vor- und Nachbedingungen von Testfällen zu organisieren.

Wie Sie sich sicher vorstellen können, wird es mit dieser Methode ziemlich mühsam sein, die Testdaten zu verwalten. Insbesondere, wenn Sie eine neue Rabattstufe anlegen oder eine Stufe wegfällt. Ein weiterer Nachteil dieses Ansatzes ist, dass die Testdaten in QF-Test gehalten werden.

Das nächste Kapitel Datentreiberkonzept[308] zeigt Ihnen nun, wie Testsuiten organisiert werden können, in denen nur ein Testfall implementiert wird und die Testdaten unabhängig vom Testfall abgelegt werden.

28.3 Datentreiberkonzept

Wenn man nun einen Testfall starten möchte, der mehrere Testdatensätze verwendet, muss man zuerst die Testdatensätze in einer Datenquelle definieren. Diese Datenquelle muss innerhalb eines Datentreiber Knotens definiert werden. QF-Test bietet Standardverknüpfungen für Datenbanktabellen, CSV-Dateien, Excel-Dateien und für innerhalb von QF-Test definierte Datentabellen an. Die QF-Test Datentabelle speichert die Testdaten in der Testsuite selbst. Wir werden diese Datenquelle für unser folgendes Beispiel nutzen. Jede andere Datenquellenart, z.B. XML Dateien, kann mit einem selbstimplementierten Skript angebunden werden.

Fügen Sie zuerst einen neuen Testfallsatz in die Testsuite ein. Sie können den Namen frei wählen.

Ein Datentreiber Knoten kann in einen Testfallsatz mittels Rechtsklick und Auswahl von Knoten einfügen→Datentreiber→Datentreiber eingefügt werden. Sie müssen nur noch einen Namen für diesen Knoten definieren. Die eigentliche Testdatenquelle kann als Kindknoten des Datentreiber Knotens eingefügt werden. In unserem Fall werden wir eine 'Datentabelle' mittels Rechtsklick und Auswahl von Knoten einfügen→Datentreiber→Datentabelle einfügen. Sie sollten nun diesen

Dialog erhalten:

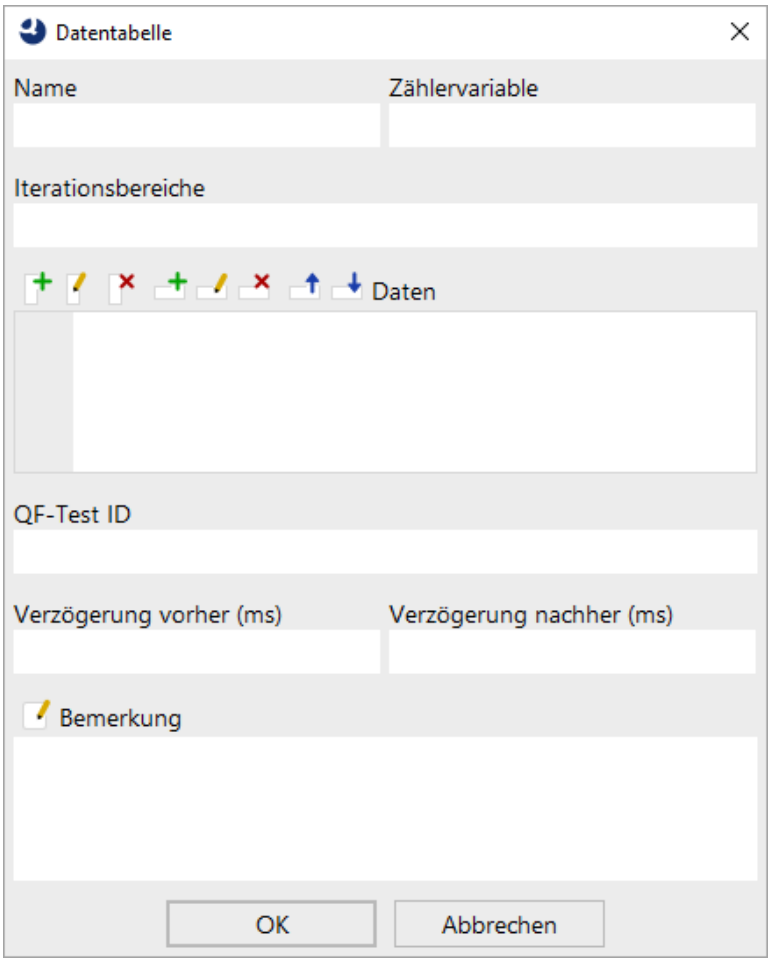

Abbildung 28.3: Dialog für eine Datentabelle

Zuerst müssen wir einen Namen für diese Datenquelle spezifizieren. Wir sollten auch eine 'Zählervariable' definieren. Die Zählervariable beinhaltet den Index des aktuell verwendeten Datensatzes während der Testausführung.

Der nächste Schritt ist die Definition der Testdaten. Dafür klicken Sie auf den Knopf 'Spalte einfügen'. Das ist der erste Knopf im 'Daten'-Bereich. Dann müssen Sie einen Namen für diese Spalten definieren. Setzen Sie den Namen auf 'rabatt'. Danach drücken Sie auf 'OK' und die Spalte sollte eingefügt worden sein. Diese Spalte wird später der Variablenname in den Tests sein.

Nun klicken Sie auf den 'Zeile einfügen' Knopf, um eine neue Zeile einzufügen. Jede Zeile wird für einen eigenen Testdatensatz stehen, d.h., dass wir jetzt drei Zeilen einfügen müssen. Die erste Zeile soll '0', die zweite '10' und die dritte '15' beinhalten.

Die Tabelle sollte nun so aussehen:

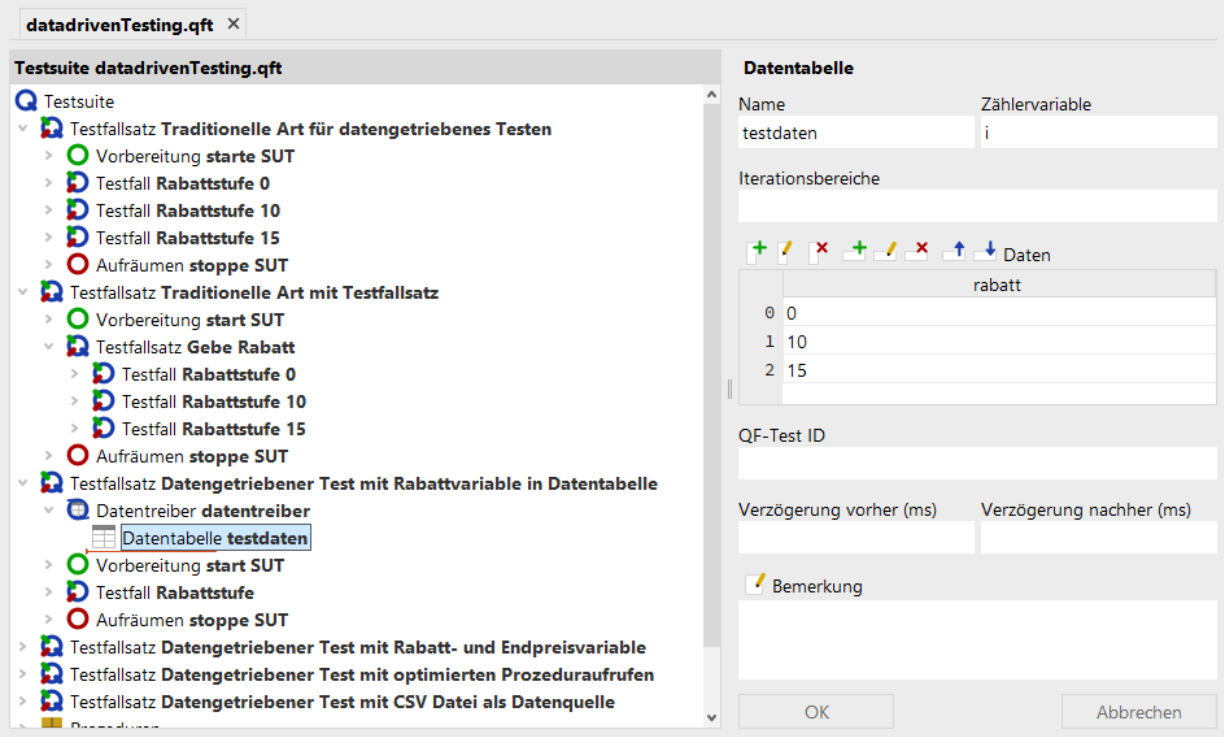

Abbildung 28.4: Die gefüllte Datentabelle

Nun kommen wir zur eigentlichen Implementierung des Testfalles. Dafür müssen wir nur einen Testfall zum Testfallsatz hinzufügen.

Hinweis Wenn sie einen Testfall zu einem Testfallsatz hinzufügen wollen, dann müssen Sie auf den geschlossenen Datentreiber klicken, um den Testfall einzufügen.

Der Testfall wird die Prozeduraufrufe für die benötigten Testschritte beinhalten. Sie können auch die Vorbereitung und die Aufräumen Knoten aus dem vorigen Testfallsatz in den neuen Testfallsatz kopieren. Der gesamte Testfallsatz sollte nun so ausschauen:

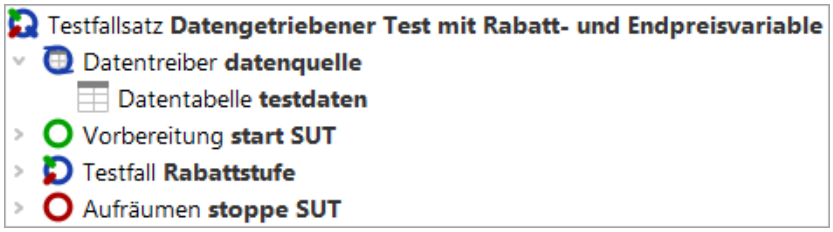

Abbildung 28.5: Testfallsatz mit Datentreiber

Jetzt müssen wir noch die Variable 'rabatt', welche in der Datentabelle definiert wurde, verwenden. Wir sollten diese Variable als Parameter für den 'setzeRabatt' Prozeduraufruf hinzufügen. Wenn wir das gemacht haben, sollte unser Test so aussehen:

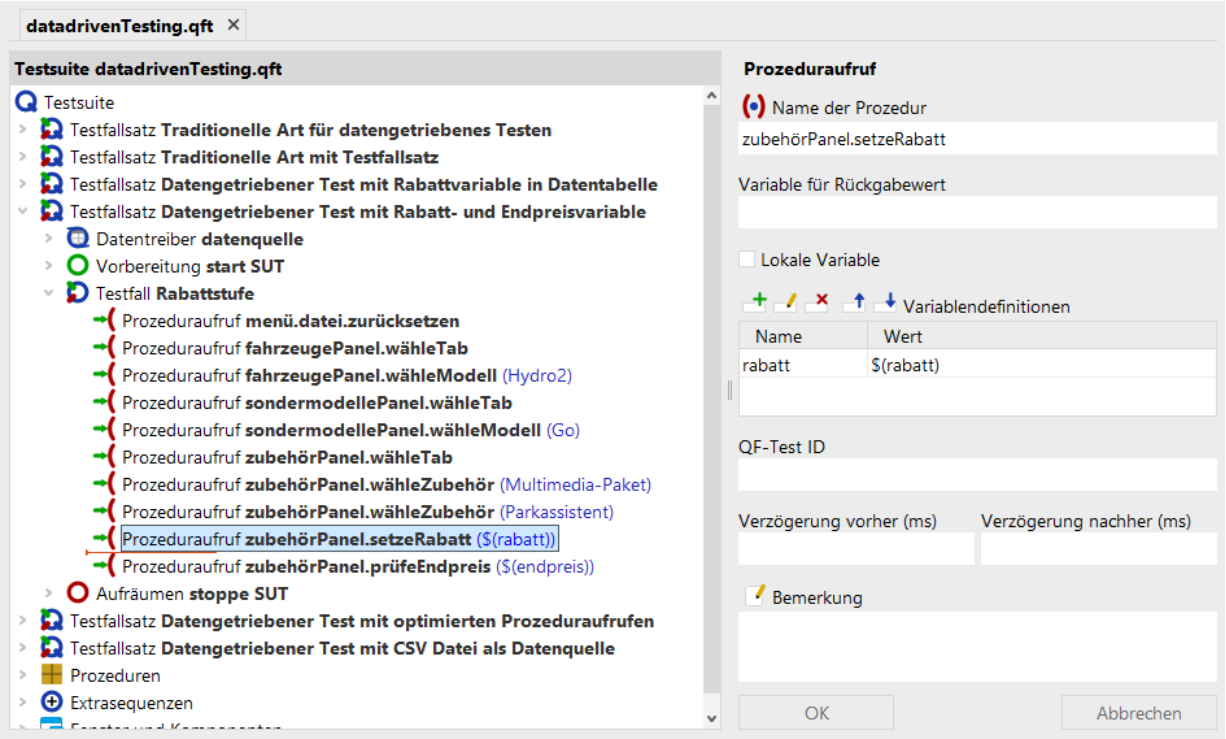

Abbildung 28.6: Der `$(rabatt)` Parameter

Nun sind wir bereit den Testfallsatz zu starten.

Nach der Ausführung der Tests sollten wir mindestens zwei Fehler bekommen. Diese Fehler kommen daher, dass das 'Endpreis' Textfeld natürlich unterschiedliche Werte enthält wir aber immer denselben Wert prüfen. In unserem Fall sollte also der erwartete Wert für das 'Endpreis' Textfeld als zweite Spalte in die 'Datentabelle' eingefügt werden.

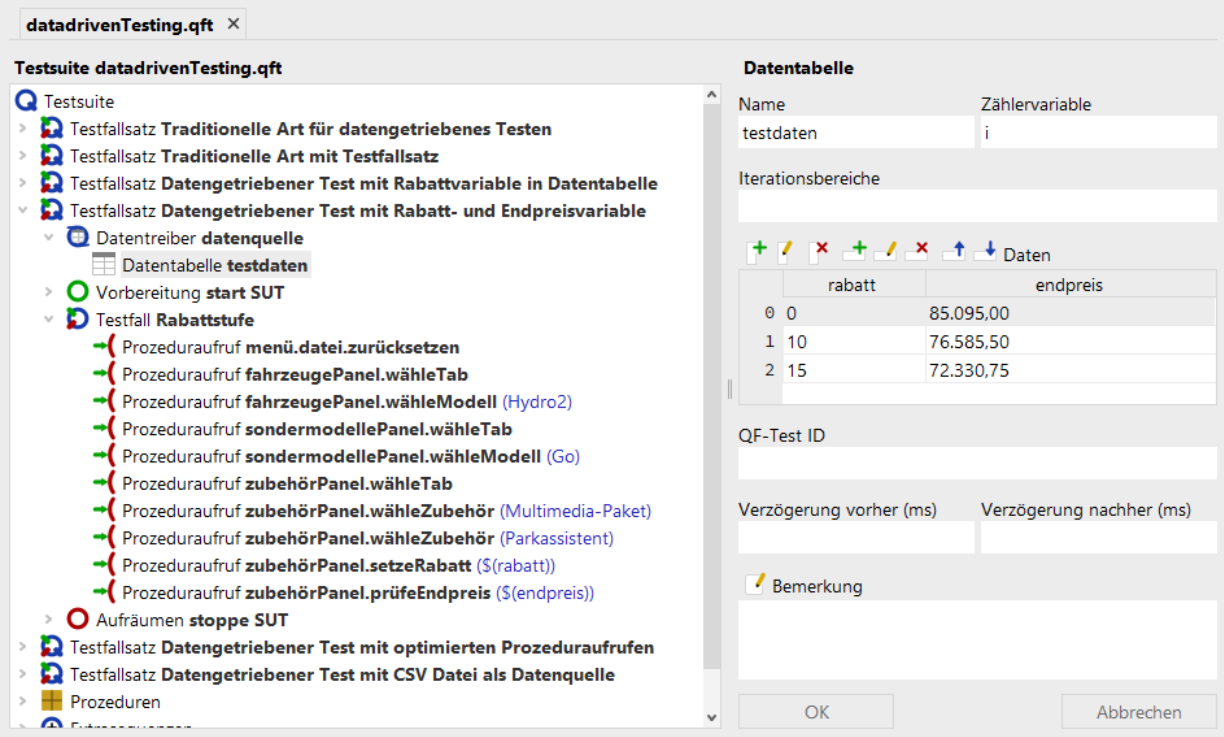

Abbildung 28.7: Vollständige Datentabelle

Ein weiterer Nachteil ist, dass wir im HTML Report und im Protokoll immer denselben Testfallnamen sehen. Um dies zu vermeiden, sollten wir die Eigenschaft 'Name für Protokoll und Report' des Testfall Knotens editieren. In dieser Eigenschaft sollten wir zumindest eine datensatz-spezifische Variable verwenden, z.B. 'rabatt' in unserem Fall. Lassen Sie uns dieses Attribut also auf 'Rabattstufe: $(rabatt)' setzen.

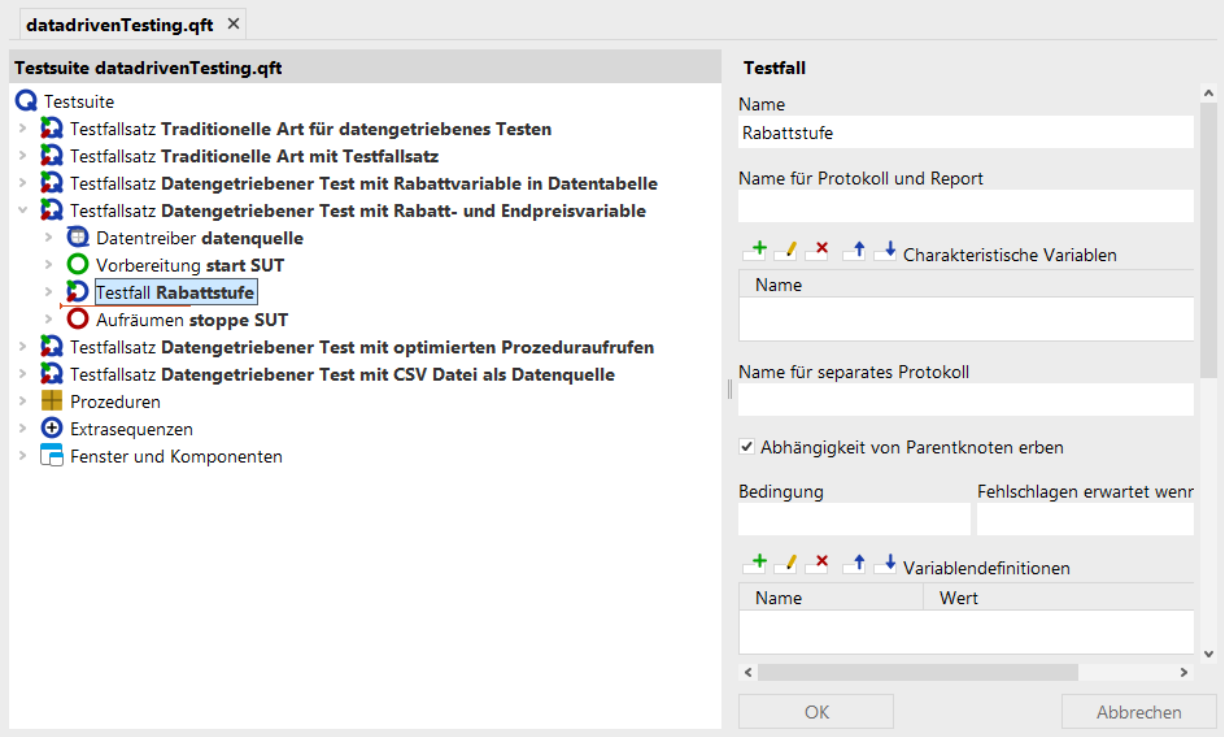

Abbildung 28.8: Name für Protokoll und Report Eigenschaft

Wenn wir den Test nun ausführen, sollten wir keine Fehler mehr erhalten und im Protokoll sowie im HTML Report sollte jeder Testfall einen eigenen Namen haben. Das erzeugte Protokoll sollte folgendermaßen aussehen:

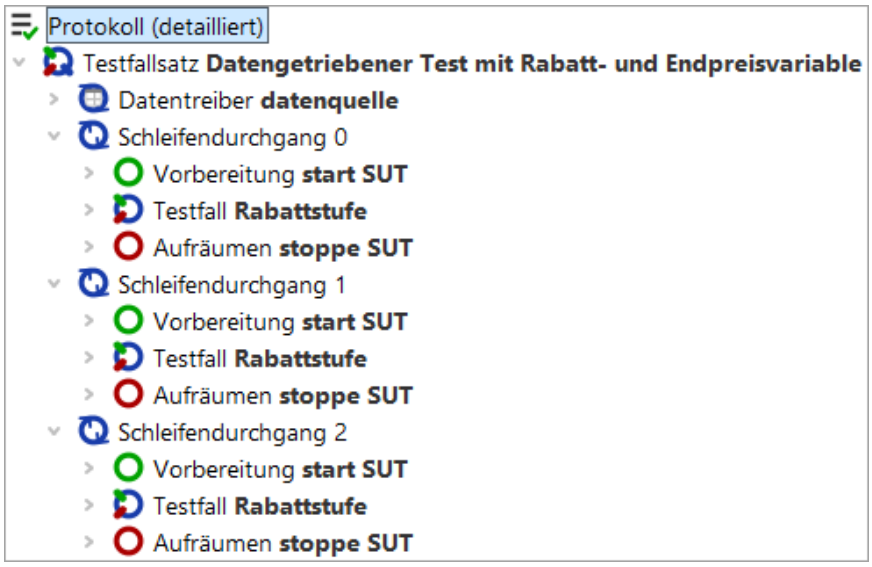

Abbildung 28.9: Protokoll mit unterschiedlichen Namen für Testfälle

Wenn Sie nur einen Testfall mit einem Datensatz ausführen möchten, ohne den gesamten Testfallsatz auszuführen, dann sollten Sie Defaultwerte als globale Variablen des Testsuite Knoten definieren.

Hinweis Wenn der Name der Variable im Datentreiber der selbe ist, wie der des Prozedurparameters, dann können Sie die Variablendefinition beim Prozeduraufruf weglassen. Das kann gemacht werden, weil die Variablen des Datentreiber Knotens auch auf den Variablenstack von QF-Test gelegt werden und so jeder Schritt innerhalb des Testfall Knotens auf diese Variablen zugreifen kann. Sie können eine solche Implementierung im Testfallsatz 'Datengetriebener Test mit optimierten Prozeduraufrufen' in der Demotestsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/datadrivenTesting.qft` sehen.

In der Demotestsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/datadrivenTesting.qft` finden Sie auch einen Testfallsatz, der eine CSV-Datei als Datenquelle nutzt.

28.4 Zusammenfassung

Das Datentreiber Konzept von QF-Test ermöglicht es dem Benutzer, logische Testfälle zu erstellen und die Testdaten von der eigentlichen Testimplementierung separiert zu halten.

Es ist auch möglich, verschachtelte Datentreiber Knoten in Testfälle zu verwenden. Dies

kann realisiert werden, indem man einen Testschritt Knoten zu einem Testfall hinzufügt. Der Testschritt Knoten kann dann den Datentreiber beinhalten.

Eine detaillierte Beschreibung von datengetriebenem Testen finden Sie im Handbuch im Kapitel Datengetriebenes Testen.

In der zweiten mitgelieferten Testsuite `qftest-9.0.0/doc/tutorial/datadriver.qft` finden Sie weitere Beispiele, wie das Auslesen einer Excel-Datei oder einer Schleife um Testfälle.

Kapitel 29

Abhängigkeiten: Automatisches Sicherstellen der korrekten Vorbedingungen jedes Testfalles

Video

Video:

Abhängigkeiten
https://www.qftest.com/de/yt/abhaengigkeiten-basics-45.html

Dieses Kapitel erklärt das Abhängigkeiten Konzept von QF-Test. Dieses Konzept ist für die Erstellung robuster Testfälle sowie für das Recovery Management sehr wichtig. Abhängigkeiten wurden eingeführt, um sicherzustellen, dass jeder Testfall mit erfüllten Vorbedingungen laufen kann.

Die Beispiele aus diesem Kapitel finden Sie in der Testsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/dependencies.qft`. Des weiteren gibt es noch eine zweite Testsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/dependencies_work.qft`, die Sie für die Erstellung der Beispiele verwenden können. Bitte achten Sie darauf, dass Sie alle Testsuiten vorher in einen projektspezifischen Ordner kopieren und diese dort modifizieren.

29.1 Einführung

Bitte kopieren Sie die Testsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/dependencies_work.qft` in ein projektspezifisches Verzeichnis und öffnen diese. Werfen Sie einen Blick auf den ersten Testfallsatz 'Rabattstufen Tests'. Dieser enthält drei Testfall Knoten und eine Vorbereitung, sowie einmal Aufräumen, um das SUT vor jedem Testfall zu starten bzw. zu

stoppen. Das ist ein typisches Beispiel, wie Testsuiten in Projekten aussehen können.

Abbildung 29.1: Erster Testfallsatz von dependencies_work.qft

Angenommen, wir wollen nur einen speziellen Testfall starten, weil genau dieser einen Defekt verifiziert oder dieser beim letzten Lauf fehlerhaft war. Dafür müssten wir entweder den gesamten Testfallsatz ausführen oder dafür sorgen, dass alle Vorbedingungen erfüllt sind, d.h. die einzelnen Vorbereitung Knoten müssten manuell ausgeführt werden.

Diese Situation passiert sehr häufig, ist allerdings nicht einfach aufzulösen mit den jetzt bekannten Mitteln. Für solche Fälle liefert QF-Test das Abhängigkeiten Konzept. Es erleichtert die Verwaltung von Vorbedingungen und erlaubt es einen Testfall einzeln zu starten. In diesem Fall wird QF-Test die Kontrolle über die Sicherstellung der Vorbedingungen übernehmen, also z.B. das SUT starten oder ein Fahrzeugmodell selektieren.

Eine Abhängigkeit kann einen Vorbereitung, einen Aufräumen, einen Fehlerbehandlung und einen Catch Knoten beinhalten. Die Vorbereitung einer Abhängigkeit wird vor jedem Testfall ausgeführt, damit immer sichergestellt ist, dass die Vorbedingungen des jeweiligen Testfalles erfüllt sind. Das Sicherstellen der Vorbedingungen ist ein sehr wichtiger Aspekt für eine robuste und stabile Testausführung. Sie können sich eine Situation vorstellen, in der ein Testfall das SUT beendet und daher der darauffolgende Testfall das SUT wieder starten muss. Genau für diese Situationen liefert das Abhängigkeit Konzept eine stabile und attraktive Lösung.

Der zweite Aspekt des Abhängigkeiten Konzeptes ist die Optimierung der Testausführung. Mit den bisherigen Mitteln mussten wir das SUT vor jedem Testfall starten und nach jedem Testfall stoppen. Dies ist für kleinere Applikationen wie den CarConfigurator auch kein Problem, aber stellen Sie sich das für eine große Applikation, wie eine Eclipse/RCP Anwendung oder ein ERP System, vor. Hier könnte dieses Vorgehen ziemlich ineffizient werden. Genau deshalb wird bei Abhängigkeiten die Aufräumen nur bei Bedarf ausgeführt.

Ein weiterer Vorteil von Abhängigkeiten sind globale Fehlerbehandlung und Catch Knoten für die Implementierung von Recovery Management Schritten. Dieses Feature ist besonders dann wichtig, wenn Sie viele Testfälle hintereinander ausführen und ein fehlerhafter die Ausführung der darauffolgenden behindern kann, z.B. durch das Erscheinen eines modalen Fehlerdialoges wie 'OutOfMemoryException'.

Zusammengefasst sind Abhängigkeiten:

1. eine Stelle, um Vorbedingungen eines Testfall zu definieren.
2. sehr nützlich, um Testfälle unabhängiger voneinander zu gestalten.
3. ein besserer Ansatz, um Vorbereitung und Aufräumen Schritte zu implementieren.
4. eine Stelle, um Recoveryschritte im Fehlerfall bzw. beim Auftreten von Exceptions zu definieren.
5. eine Optimierungsmöglichkeit für die Testausführung.
6. wiederverwendbar, da sie im Prozeduren Bereich abgelegt werden können.

Die folgenden Abschnitte zeigen, wie man Abhängigkeiten anlegt und benutzt.

29.2 Sicherstellen von Vorbedingungen

Bitte kopieren Sie die Testsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/dependencies_work.qft` in ein projektspezifisches Verzeichnis und öffnen diese, falls Sie das nicht ohnehin schon getan haben sollten.

Diese Datei enthält einen Testfallsatz 'Rabatt-Tests' mit drei Testfälle und der herkömmlichen Implementierung von Vorbereitung und Aufräumen Knoten. Wir werden nun in diesen Testfallsatz eine Abhängigkeit einbauen.

Abbildung 29.2: Erster Testfallsatz von dependencies_work.qft

Zuerst müssen wir einen Abhängigkeit Knoten einfügen. Dies macht man mittels Rechtsklick auf den Testfallsatz und Auswahl von Knoten einfügen→Abhängigkeiten→Abhängigkeit. Geben Sie der Abhängigkeit einen Namen, z.B. "SUT gestartet".

Der nächste Schritt ist das Verschieben der Vorbereitung und Aufräumen Sequenzen in diese Abhängigkeit. Hierfür müssen Sie den Abhängigkeit Knoten öffnen und die entsprechenden Knoten hineinschieben. Das können Sie entweder via Drag and Drop oder Rechtsklick Ausschneiden und Einfügen oder mittels Ctrl-X und Ctrl-V.

Die Testsuite sollte nun so aussehen:

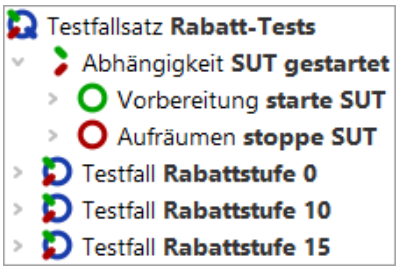

Abbildung 29.3: Beispiel Testsuite mit der ersten Abhängigkeit

Jetzt wollen wir die Abhängigkeit testen. Stoppen Sie bitte vorher alle laufenden SUTs. Dann selektieren Sie einen Testfall, z.B. 'Rabattstufe 10' und starten diesen.

Sie sollten nun sehen, dass der Testfall ausgeführt wurde und das SUT am Ende des Testlaufes nicht gestoppt wurde. Bitte öffnen Sie das Protokoll um nachzusehen, was genau passiert ist.

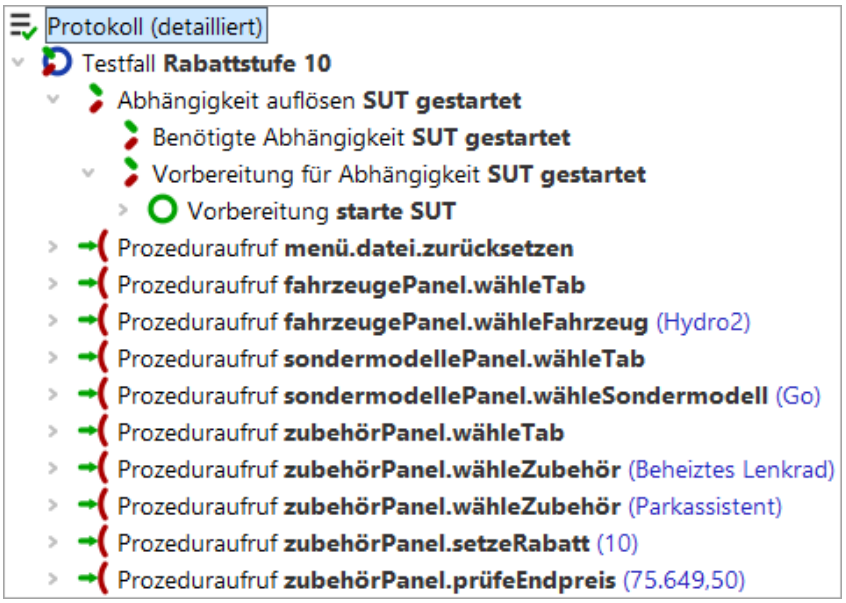

Abbildung 29.4: Das Protokoll der Ausführung

Wenn Sie im Protokoll den Testfall öffnen, dann sehen Sie einen 'Abhängigkeiten

auflösen'-Knoten. Wenn Sie diesen öffnen, werden Sie zwei weitere Knoten sehen. Der letzte der beiden Knoten zeigt Ihnen, dass die Vorbereitung Sequenz ausgeführt wurde. Der erste Knoten wird im nächsten Beispiel erklärt.

Bis jetzt haben wir gesehen, dass die Vorbereitung Sequenz automatisch vor dem Testfall ausgeführt wird. Jedoch wurde Aufräumen noch nicht ausgeführt. Wenn Sie jetzt einen weiteren Testfall starten, z.B. 'Rabattstufe 15', wird dieser auf der bereits gestarteten Anwendung ausgeführt.

Die Vorbereitung einer Abhängigkeit wird auf jeden Fall vor jedem Testfall ausgeführt. Dies geschieht um die Vorbedingungen jedes einzelnen Testfall's sicherzustellen. Das Aufräumen einer Abhängigkeit wird nur bei Bedarf ausgeführt, d.h. nur dann, wenn die Schritte der Vorbereitung nicht mehr benötigt werden. In unserem Fall wurde das Aufräumen nicht ausgeführt, weil beide Testfälle dieselbe Abhängigkeit haben. Die Testausführung liefert allerdings keine Fehler, weil die Prozedur `startStop.starteApplikation` bereits prüft, ob das SUT überhaupt gestartet werden soll.

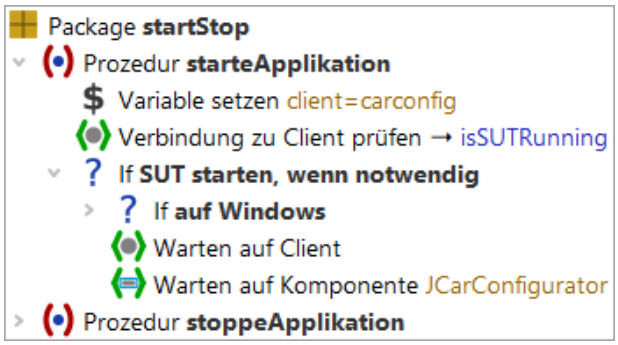

Abbildung 29.5: Prozedur startStop.starteApplikation

Der nächste Schritt ist, den gesamten Testfallsatz mittels Klick auf 'Wiedergabe starten' auszuführen.

Alle drei Testfälle sollten erfolgreich durchgelaufen sein und das SUT sollte auch nicht zwischen den Ausführungen gestoppt worden sein. Wir haben also auch die Testausführung optimiert. Die Aufräumen wurde nicht ausgeführt, da alle drei Testfälle auf dieselbe Abhängigkeit verweisen. Somit sieht unsere Testumgebung auch mehr nach einer realen Umgebung aus, da die wenigsten Benutzer das SUT nach jeder Aktion neu starten werden.

Das nächste Ziel ist, unsere Abhängigkeit für andere Testfallsätze unseres Projektes verfügbar zu machen. Hierfür müssen wir die Abhängigkeit in den Prozeduren Bereich verschieben. Danach müssen wir auf den Testfallsatz klicken und eine Bezug auf Abhängigkeit einfügen. Dies macht man mittels Rechtsklick und Auswahl von Knoten einfügen→Abhängigkeiten→Bezug auf Abhängigkeit. Der erscheinende Dialog sieht dann dem Prozeduraufruf Dialog ziemlich ähnlich. Wählen Sie die gerade

verschobene Abhängigkeit aus. Die Testsuite sollte nun wie folgt aussehen:

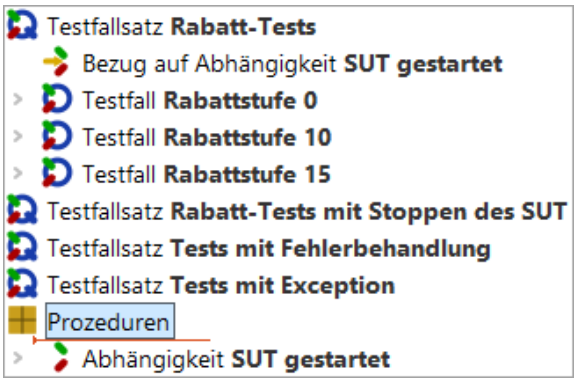

Abbildung 29.6: Die Testsuite mit Bezug auf Abhängigkeit

Wir empfehlen alle Abhängigkeiten in ein separates Package 'abhängigkeiten' zu schieben.

Wenn Sie nun den Testfallsatz ausführen, wird bei der ersten Ausführung nach Verschieben der Abhängigkeit das SUT gestoppt und das SUT wiederum gestartet. Das passiert, weil die Abhängigkeit in den Prozeduren Bereich verschoben wurde und es deshalb eine andere Abhängigkeit ist als vorher.

Werfen Sie nun einen Blick auf den zweiten Testfallsatz der Demotestsuite 'Rabatt-Tests mit Stoppen des SUT'. Der zweite Testfall 'Rabattstufe 10' stoppt das SUT, jedoch benötigt der dritte Testfall 'Rabattstufe 15' auch ein laufendes SUT. Wie wir in diesem Abschnitt gelernt haben, wird das Abhängigkeiten Konzept dafür Sorge tragen, dass das SUT vor dem dritten Testfall ausgeführt wird. Dieses Beispiel sollte noch einmal die Vorteile von Abhängigkeiten verdeutlichen.

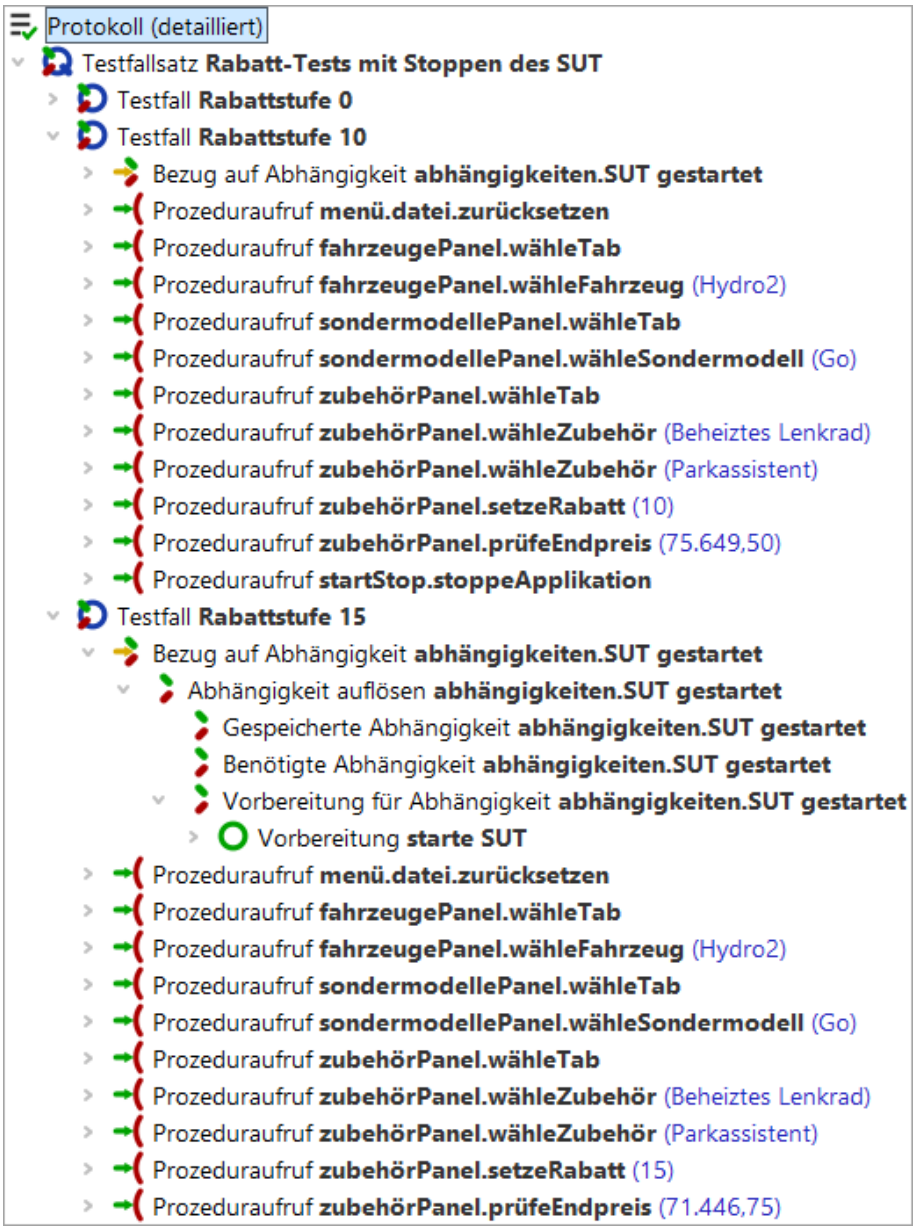

Abbildung 29.7: Sicherstellen der Vorbedingungen für Testfall 'Rabattstufe 15'

29.3 Verschachtelte Abhängigkeiten

Verwalten von Vorbedingungen kann ein durchaus komplexeres Thema werden, als dass wir nur Sicherstellen, ob das SUT gestartet wurde oder nicht. In vielen Projekten gibt es verschiedene Gruppen von Testfällen mit unterschiedlichen Vorbedingungen.

Nehmen wir an, dass wir ein großes ERP System mit mehreren Perspektiven, wie 'Anbieter' und 'Artikel' testen möchten. Jeder Testfall für die 'Anbieter' Perspektive bezieht sich darauf, dass die 'Anbieter' Perspektive auch geöffnet ist. Genauso verhält es sich bei allen Tests für die 'Artikel' Perspektive. Das Öffnen der jeweiligen Perspektive hängt wiederum vom eingeloggten Benutzer ab, und das Einloggen basiert auf einem gestarteten SUT. Sie sehen also, es gibt so etwas wie einen Baum von Vorbedingungen.

QF-Test ermöglicht es dem Benutzer, solch verschachtelte Abhängigkeit Knoten zu erstellen. Hierfür muss man Bezug auf Abhängigkeit Knoten zu einer Abhängigkeit hinzufügen. Wir werden nun ein kleines Beispiel mit zwei verschachtelten Abhängigkeiten für den CarConfigurator bauen.

Im CarConfigurator können Sie den 'Fahrzeuge' Dialog mittels der Menüaktion 'Einstellungen' -> 'Fahrzeuge' öffnen. Wir wollten jetzt Tests für diesen Dialog erstellen. Später werden wir auch Tests für den 'Zubehör' Dialog erstellen, welcher auch über das Menü mittels 'Einstellungen' -> 'Zubehör' geöffnet werden kann.

Zuerst definieren wir die Tests, welche wir erstellen wollen.

Testfall 1: Anlegen des Modells 'test1' mit Preis '100'.

- Starten des SUT, falls notwendig.
- Fahrzeugdialog öffnen.
- Name 'test1' und Preis '100' setzen.
- 'Neu' klicken.
- Den Dialog mittels 'OK' schließen.
- Den Fahrzeugdialog nochmal öffnen.
- Das angelegte Modell 'test1' selektieren.
- Den Dialog mit 'Abbrechen' schließen.
- Das SUT stoppen, falls notwendig.

Testfall 2: Anlegen des Modells 'test2' mit Preis '99999'.

- Start des SUT, falls notwendig.
- Fahrzeugdialog öffnen.
- Name 'test2' und Preis '99999' setzen.
- 'Neu' klicken.

- Den Dialog mittels 'OK' schließen.
- Den Fahrzeugdialog nochmal öffnen.
- Das angelegte Modell 'test2' selektieren.
- Den Dialog mit 'Abbrechen' schließen.
- Das SUT stoppen, falls notwendig.

Testfall 3: Anlegen des Zubehöres 'testzubehör' mit Preis '12'.

- Start des SUT, falls notwendig.
- Zubehördialog öffnen.
- Name 'testzubehör' und Preis '12' setzen.
- 'Neu' klicken.
- Den Dialog mittels 'OK' schließen.
- Den Zubehördialog nochmal öffnen.
- Das angelegte Zubehörteil 'testzubehör' selektieren.
- Den Dialog mit 'Abbrechen' schließen.
- Das SUT stoppen, falls notwendig.

Wenn wir uns nun die Testschritte der oben definierten Testfälle genauer anschauen, sehen wir, dass jeder Testfall ein laufendes SUT benötigt. Daher sollten wir eine Abhängigkeit 'SUT gestartet' implementieren. Das Stoppen des SUT ist ein optionaler Schritt, der innerhalb des Aufräumen Knotens dieser Abhängigkeit implementiert werden kann. Diese Abhängigkeit haben wir schon im vorigen Beispiel erstellt und können diese also wiederverwenden.

Der nächste Punkt ist, dass sowohl Testfall 1 wie auch Testfall 2 einen geöffneten Fahrzeugdialog benötigen. Da wir weitere Tests in diesem Bereich planen, sollten wir eine Abhängigkeit 'Fahrzeugdialog geöffnet' erstellen. Diese sollte in der Vorbereitung das Öffnen des Dialoges und in der Aufräumen das Schließen mittels 'Abbrechen' beinhalten. Wir können diesen Dialog nur dann öffnen, wenn das SUT bereits läuft, deshalb ist diese Abhängigkeit auch von der Abhängigkeit 'SUT gestartet' abhängig. Die Implementierung der 'Fahrzeugdialog geöffnet' Abhängigkeit sieht wie folgt aus:

Abbildung 29.8: 'Fahrzeugdialog geöffnet' Abhängigkeit

Hinweis

In der Vorbereitung müssen wir prüfen, ob der Dialog bereits geöffnet wurde. Es könnte nämlich sein, dass ein voriger Testfall den Dialog bereits geöffnet und nicht mehr geschlossen hat. Das Attribut 'Wartezeit' des Warten auf Komponente Knotens steht hier auf '0', weil wir erwarten, dass der Dialog offen sein soll, wenn nicht, dann muss dieser ohnehin geöffnet werden.

Wir sollten auch eine Abhängigkeit 'Zubehördialog geöffnet' erstellen, welche ähnlich zur 'Fahrzeugdialog geöffnet' Abhängigkeit, den Zubehördialog öffnet.

Nach Erstellen der Abhängigkeiten müssen wir nun die entsprechenden Testschritte aufzeichnen und die Testfälle erstellen. Die Testschritte wurden schon erstellt und können als Prozeduren in der entsprechenden Dialog Packages Struktur gefunden werden.

Die Testfälle sollten in einem Testfallsatz namens 'Verschachtelte Abhängigkeiten' zusammengefasst erstellt werden. Dieser Testfallsatz sollte zwei weitere Testfallsatz beinhalten. Das erste ist 'Tests für den Fahrzeugdialog', das zweite 'Tests für den Zubehördialog'. Der Testfallsatz 'Tests für den Fahrzeugdialog' beinhaltet die Implementierungen der Testfälle 1 und 2, sowie eine Bezug auf Abhängigkeit auf die 'Fahrzeugdialog geöffnet' Abhängigkeit. Der zweite Testfallsatz 'Tests für den Zubehördialog' beinhaltet die Implementierung des Testfalles 3 und eine Bezug auf Abhängigkeit auf die Abhängigkeit 'Zubehördialog geöffnet'.

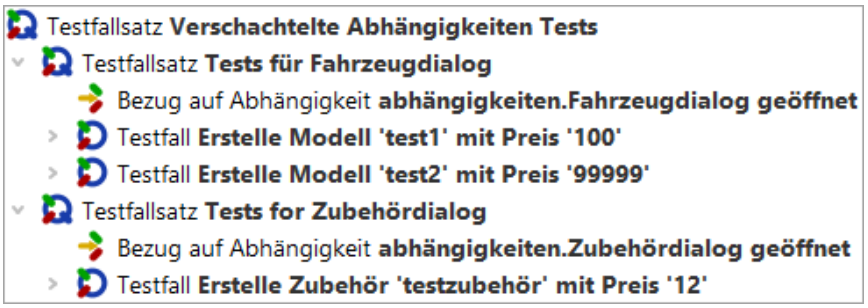

Abbildung 29.9: Implementierung der Testfälle

Wenn Sie nun den obersten Testfallsatz starten, werden Sie sehen, dass QF-Test das SUT zuerst stoppt, das kommt von der Abhängigkeit des vorherigen Beispieles. Danach wird das SUT gestartet und es werden die Schritte von Testfall 1 und Testfall 2 ausgeführt und schlussendlich die Schritte von Testfall 3. Wenn Sie im Protokoll einen genaueren Blick auf den Anfang von Testfall 3 werfen, dann werden Sie sehen, dass auch die Aufräumen der 'Fahrzeugdialog geöffnet' Abhängigkeit ausgeführt wurde. Dies passierte, weil die Abhängigkeit 'Fahrzeugdialog geöffnet' nicht mehr benötigt wurde. Testfall 3 hat allerdings die 'Zubehördialog geöffnet' Abhängigkeit benötigt und hat deshalb deren Vorbereitung durchlaufen. Da diese beiden Abhängigkeiten auf der 'SUT gestartet' Abhängigkeit aufbauen, wurde deren Aufräumen Sequenz nicht ausgeführt.

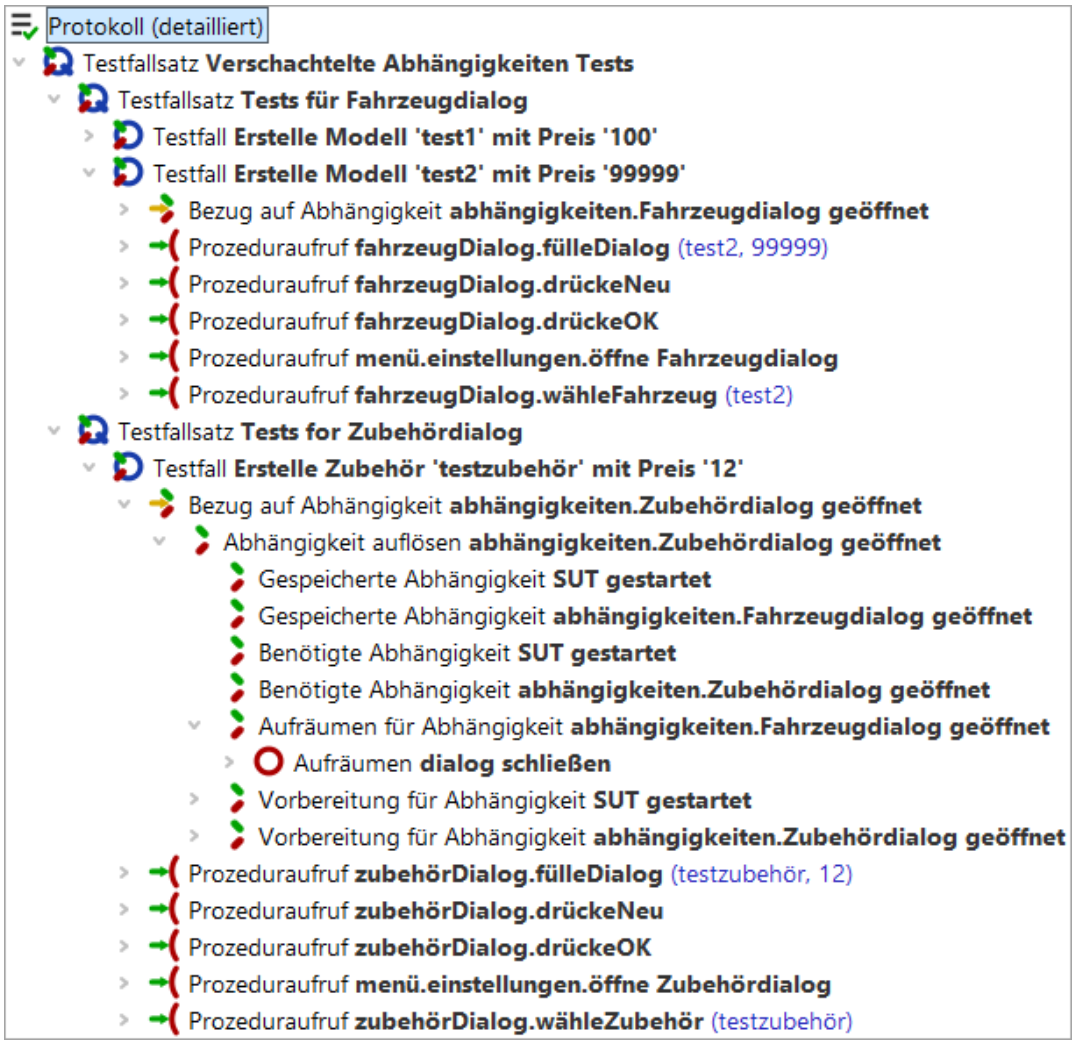

Abbildung 29.10: Protokoll von verschachtelten Abhängigkeiten

Das Verschachteln von Abhängigkeiten und die Möglichkeit, eine Aufräumen Sequenz nur bei Bedarf aufzurufen, ermöglichen Ihnen relativ viele vor- und nachbereitende Schritte in eine Abhängigkeit zu packen. Ein anderer Anwendungsfall für den CarConfigurator könnte eine Abhängigkeit 'Fahrzeug angelegt' sein, welche sicherstellt, dass das verwendete Fahrzeug vorher angelegt wird.

29.4 Fehler- und Exceptionbehandlung

29.4.1 Fehlerbehandlung

Bitte kopieren Sie die Testsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/dependencies_work.qft` in ein projektspezifisches Verzeichnis und öffnen diese, falls Sie das nicht ohnehin schon getan haben sollten. Dort finden Sie einen Testfallsatz 'Tests mit Fehlerbehandlung'. Der zweite Testfall ist fehlerhaft.

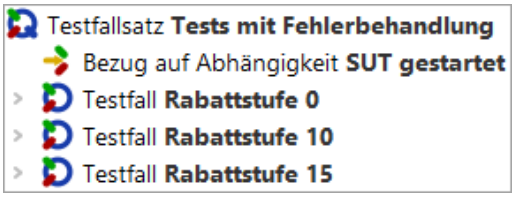

Abbildung 29.11: Testsuite für Fehlerbehandlung

Angenommen wir wollen bestimmte Aktionen auslösen, die nach einem fehlerhaften Testfall ausgeführt werden sollen. In unserem Fall könnten wir im Fehlerfall einfach das SUT stoppen. Dies könnte notwendig sein, um zu garantieren, dass die folgenden Testfälle auf einer sauberen Umgebung aufsetzen können. Wir wissen bis jetzt, dass die Vorbereitung Sequenz vor jedem Testfall ausgeführt wird und die Aufräumen nur bei Bedarf ausgeführt wird. Aber wie können wir jetzt diese spezielle Fehlerbehandlung implementieren?

Die Lösung ist der so genannte Fehlerbehandlung Knoten für eine Abhängigkeit. Wenn Sie auf die geschlossene Aufräumen Sequenz klicken, können Sie mit Rechtsklick Knoten einfügen→Abhängigkeiten→Fehlerbehandlung.

Im Fehlerbehandlung Knoten können Sie die Schritte für das Stoppen des SUT aufrufen. Die Abhängigkeit `SUT gestartet` sollte nun wie folgt aussehen:

Abbildung 29.12: Abhängigkeit mit Fehlerbehandlung

Führen Sie nun den gesamten Testfallsatz 'Tests mit Fehlerbehandlung' aus und öffnen

Sie das Protokoll nachdem die Ausführung abgeschlossen ist.

Im Protokoll können Sie sehen, dass der Fehlerbehandlung Knoten nach dem zweiten Testfall ausgeführt wurde.

Abbildung 29.13: Protokoll einer Abhängigkeit mit Fehlerbehandlung

29.4.2 Exception Behandlung

Im vorigen Abschnitt haben wir gelernt, dass man mit Fehlerbehandlung Knoten Schritte definieren kann, welche bei fehlerhaften Testfälle ausgeführt werden. Neben Fehlern können allerdings auch Exceptions beim Testlauf auftreten. Eine Exception ist ein unerwartetes Verhalten während der Testausführung, z.B. ein Dialog erscheint und blockiert die Ausführung oder eine Komponente konnte nicht gefunden werden. Wie soll man mit solchen Exceptions umgehen?

In der Demotestsuite `dependencies_work.qft` finden Sie ein Beispiel Testfallsatz namens 'Tests mit Exception'.

Natürlich können Sie entsprechende Testschritte mit einem Try-Catch umrunden und eine dedizierte Exceptionbehandlung in jedem Testfall implementieren. Im Beispiel Testfallsatz wurde dies so implementiert. Dieser Ansatz kann jedoch zu viel Redundanz führen und die Testfälle werden noch unleserlicher.

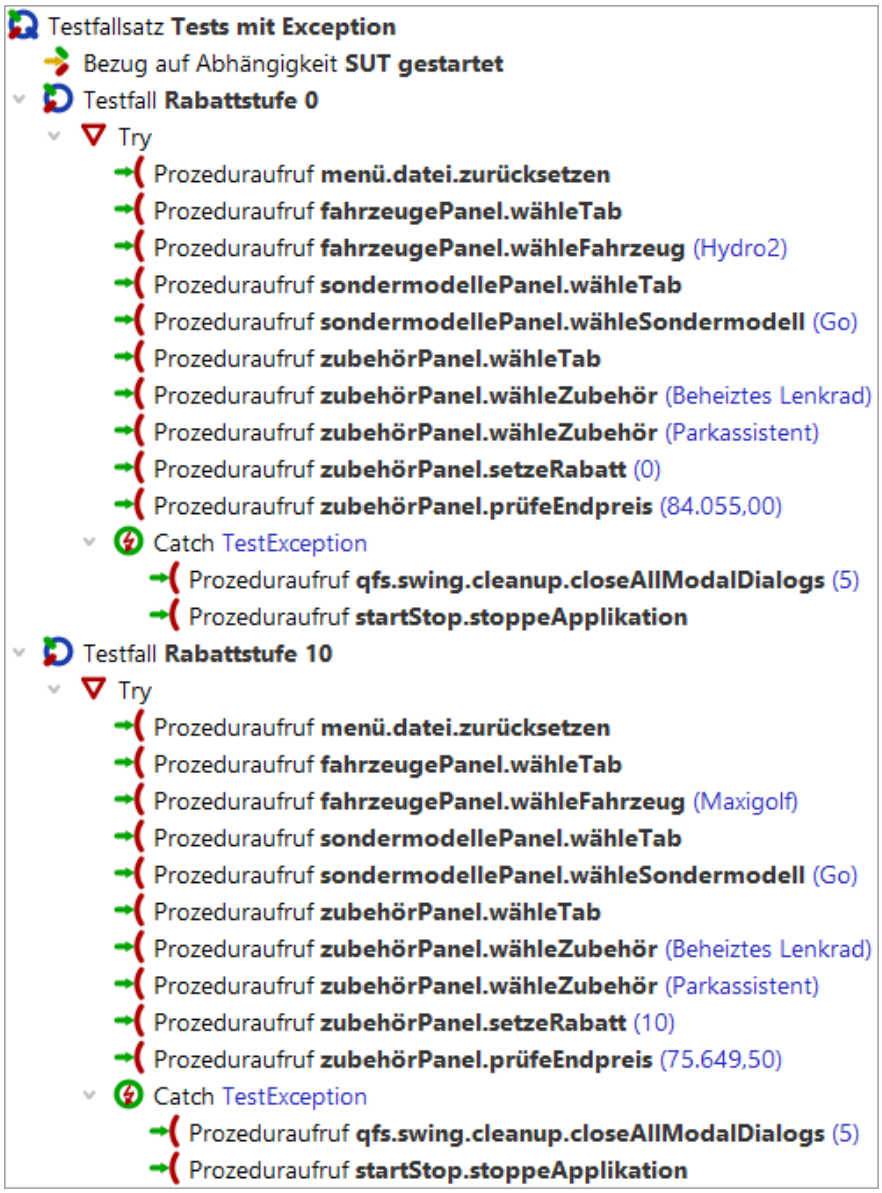

Abbildung 29.14: Try-Catch Knoten in Testfälle

Unser Ziel ist es nun, die Redundanz in den Testfällen zu reduzieren und die einheitliche Exceptionbehandlung an eine zentrale Stelle zu verschieben. Diese zentrale Stelle wird unsere Abhängigkeit sein.

Der erste Schritt hierfür ist das Einfügen des Catch Knotens in eine Abhängigkeit. Hierzu müssen Sie auf dem geschlossenen Fehlerbehandlung Knoten einen Rechtsklick ausführen und Knoten einfügen→Ablaufsteuerung→Catch einfügen. Danach können wir die Schritte aus einem der Catch Knoten in den neuen Knoten verschieben und in den Testfälle die Prozeduraufrufe aus dem Try Block herausziehen und den Try Block dann

löschen.

Die Testsuite sieht nun so aus:

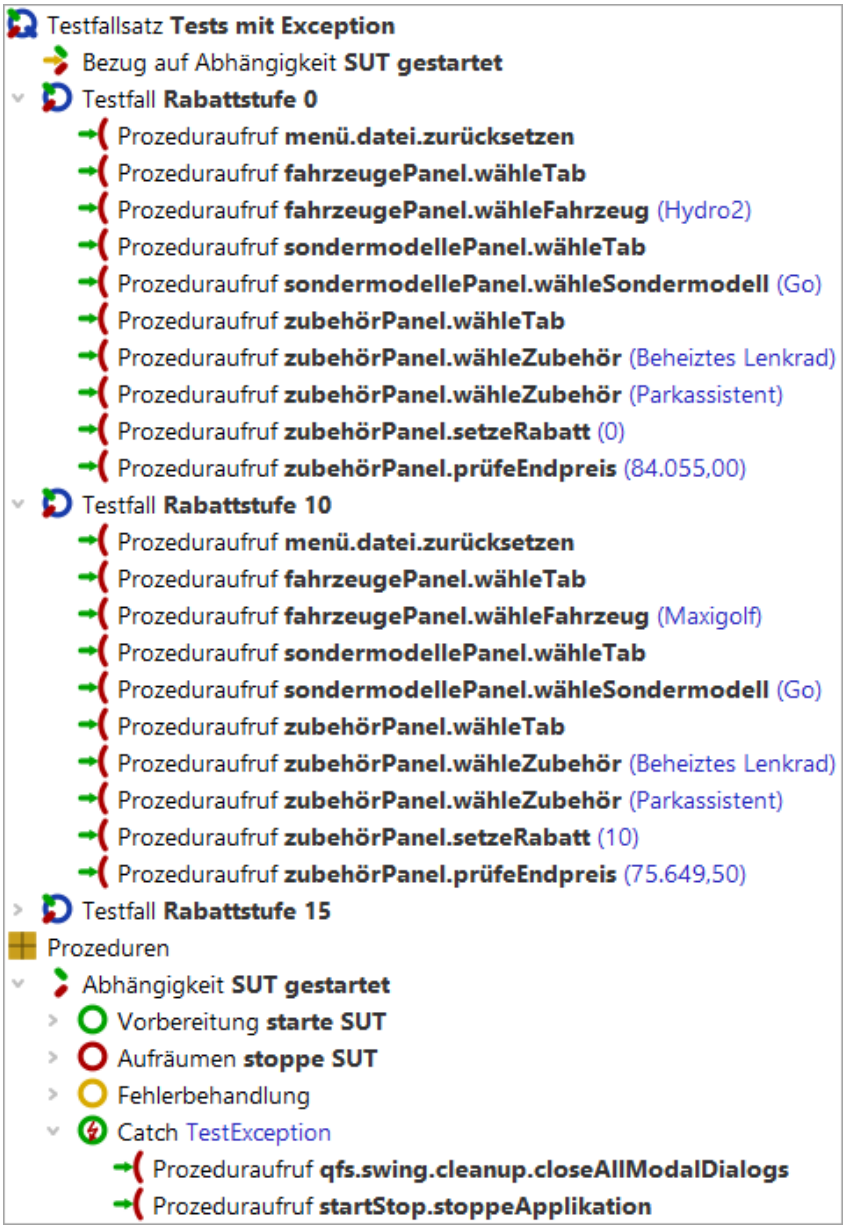

Abbildung 29.15: Testsuite mit Catch

Nun können Sie den Testfallsatz 'Tests mit Exception' starten. Der zweite Testfall wirft eine IndexNotFoundException, weil das ausgewählte Modell nicht existiert. Diese Exception sollte nun vom globalen Catch Knoten der Abhängigkeit behandelt werden.

Hinweis Wenn Sie den Debugger aktiviert haben, wird QF-Test den Testlauf an der Stelle un-

terbrechen, wo die Exception auftritt. In unserem Fall können Sie dann die Exception mit dem Knopf 'Exception erneut werfen' weiterwerfen oder den Debugger mittels dem Menüeintrag Debugger→Debugger aktivieren deaktivieren.

Öffnen Sie nach der Ausführung das Protokoll um nachzusehen, was passiert ist.

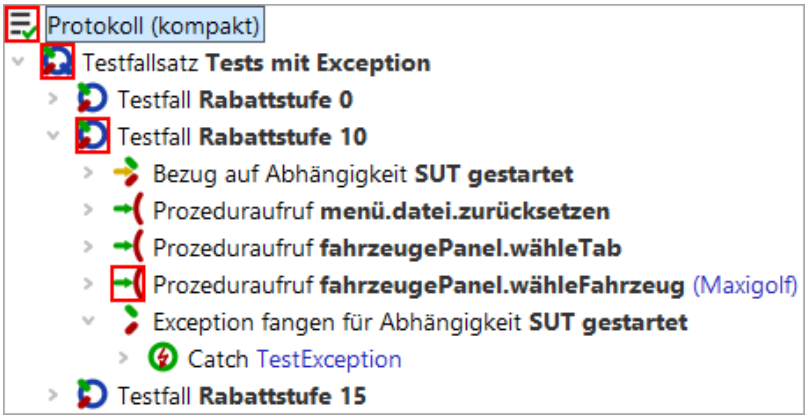

Abbildung 29.16: Protokoll der Ausführung Abhängigkeit mit Catch

In einem normalen Projekt sollten Sie mindestens einen solchen global Catch Knoten für 'TestException' erstellen. Dieser Knoten sollte dann auch entweder die Prozedur `qfs.swing.cleanup.closeAllModalDialogs` oder die Prozedur `qfs.swt.cleanup.closeAllModalDialogsAndModalShells` aufrufen. Diese Prozeduren schließen jeglichen modalen Dialog, d.h. jedes Fenster, das die Ausführung der Tests blockieren könnte.

29.4.3 Zusammenfassung

Sie haben nun gesehen, dass man ein robustes Recovery Management für Testfälle mittels Fehlerbehandlung und noch mehr mittels Catch Knoten für eine Abhängigkeit implementieren kann.

In den meisten Projekten ist ein globaler Catch Knoten sehr wichtig, besonders im Falle von ComponentNotFoundExceptions und ModalDialogExceptions.

29.5 Mehr zu Abhängigkeiten

Im oberen Bereich haben wir gesehen, dass man verschiedene Abhängigkeiten verschachteln kann und dass die Aufräumen Sequenz einer Abhängigkeit nur dann ausgeführt wird, wenn die entsprechende Abhängigkeit nicht mehr benötigt wird. Wir können

eine Abhängigkeit auch so konfigurieren, dass die Aufräumen jedes mal ausgeführt wird. Das kann mittels Setzens der Option 'Aufräumen erzwingen' der Abhängigkeit bewerkstelligt werden.

Es gibt noch viel mehr Interessantes über Abhängigkeiten zu entdecken, z.B. kann man die Ausführung der Aufräumen auch mittels Variablen steuern. Diese Variable heißt charakteristische Variable. Diese und mehr Details finden Sie im Handbuch im Kapitel Abhängigkeiten. Dieser Ansatz könnte verwendet werden, um eine Abhängigkeit 'Login' zu erstellen, deren Aufräumen, d.h. das Ausloggen, nur dann ausgeführt wird, wenn sich der Inhalt der Variable Benutzer ändert.

Eine detaillierte Beschreibung von Abhängigkeiten finden Sie im Handbuch im Kapitel Abhängigkeiten.

Kapitel 30

Automatische Erstellung von Basisprozeduren

Dieses Kapitel beschreibt, wie man mit QF-Test Prozeduren für die GUI Elemente automatisch erzeugen kann. Der Vorteil dieser Technik ist, dass man nicht mehr jeden Schritt des Tests einzeln aufzeichnen muss. Darüber hinaus wird auch eine standardisierte Package und Prozeduren Struktur bereitgestellt.

Sie finden die fertiggestellten Beispiele in der Datei `qftest-9.0.0/doc/tutorial/advanced-demos/de/automated_procedures.qft` . Es gibt auch noch eine zweite Testsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/automated_procedures_work.qft` für Ihre eigenen Implementierungen. Bitte achten Sie darauf, dass Sie alle Testsuiten vorher in einen projektspezifischen Ordner kopieren und diese dort modifizieren.

30.1 Einführung

Wenn wir für alle Features des CarConfigurator Tests erstellen wollen, so müssen wir auch Aktionen für jedes involvierte GUI Element aufzeichnen. Der CarConfigurator ist eine kleine Applikation mit vielleicht fünf Dialogen und ca. dreißig GUI Elementen. Das Erstellen der wichtigsten Testfälle für diese Applikation wird ein bis zwei Tage dauern. Aber stellen Sie sich ein großes Projekt vor, wie ein ERP System mit über fünfzig Dialogen und hunderten von GUI Elementen. Hier wird das Erstellen der Testfälle erheblich länger dauern und ebenso könnte die Wartung der Tests schwieriger werden.

Als ersten organisatorische Schritt empfehlen wir, jeden Testschritt als Prozedur zu erstellen und diese dann von den entsprechenden Testfälle aufzurufen. Wenn Sie Ihre Tests in unterschiedlichen Testsuiten organisieren, dann könnten Sie diese in zwei Schichten aufteilen. Die erste Schicht enthält nur GUI-Komponenten bezogene Proze-

duren und die zweite Schicht beinhaltet nur Testfälle, welche die Prozeduren der ersten Schicht aufrufen.

Der Ansatz jeden Testschritt als Prozedur zu implementieren bringt uns in die Lage, unsere Arbeit in zwei Bereich aufzuteilen:

1. Erstellung und Wartung von Prozeduren, welche die Testschritte repräsentieren
2. Erstellung und Wartung von Testfällen

QF-Test liefert nun ein Feature, das diese Basisprozeduren für GUI Elemente automatisch erstellt. Wenn Sie dieses Feature benutzen, reduziert sich der Erstellungsaufwand für Testsuiten und Testfälle drastisch und es unterstützt Sie in der Erstellung wartbarer Testsuiten.

Sie finden in der Demotestsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/automated_procedures.qft` einige Testfälle, die mit diesem Feature erstellt worden sind.

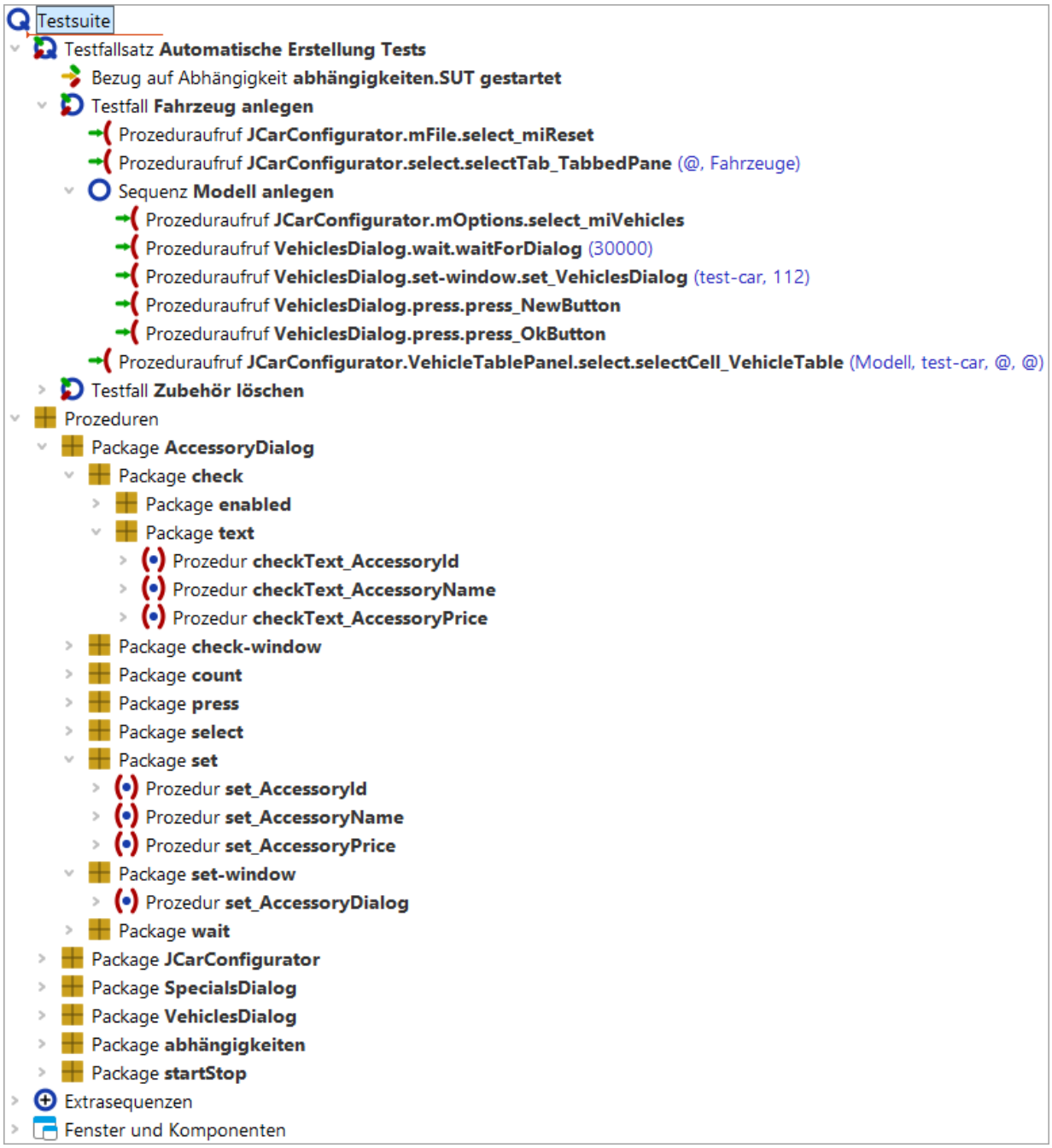

Abbildung 30.1: Bildschirmabbild der Testsuite

Die folgenden Abschnitte beschreiben nun, wie man diese Prozeduren erstellt und die Testfälle organisiert.

30.2 Automatische Erstellung von Prozeduren

Bitte kopieren Sie die Demotestsuite `qftest-9.0.0/doc/tutorial/advanced-demos/de/automated_procedures_work.qft` in ein projektspezifisches Verzeichnis und öffnen diese dort.

Diese Datei enthält einen Testfallsatz, der sich auf die Abhängigkeit 'Start SUT' bezieht.

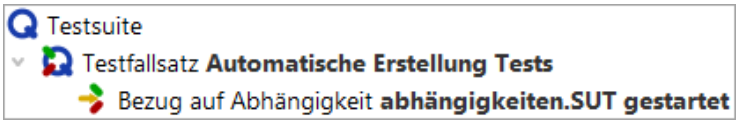

Abbildung 30.2: Die Testsuite automated_procedures_work.qft

Zuerst müssen wir das SUT starten. Hierfür selektieren Sie die Abhängigkeit und drücken auf 'Wiedergabe starten'.

Sobald das SUT läuft, können wir die Testschritte aufzeichnen. Normalerweise würden wir jetzt auf 'Aufnahme starten' klicken, die entsprechenden Schritte aufzeichnen und dann die Aufnahme mit 'Aufnahme beenden' stoppen. Danach würden wir die Aufnahme reorganisieren, d.h. Prozeduren erstellen und diese parametrisieren. Genau diese Schritte können jetzt automatisiert erfolgen.

Wir werden jetzt zuerst die Basisprozeduren für das Hauptfenster erstellen. Bevor wir loslegen, müssen wir allerdings die Konfiguration von QF-Test anpassen.

Öffnen Sie die Optionen mittels Bearbeiten→Optionen. Dann wechseln Sie nach 'Aufnahme' -> 'Prozeduren'. Dort ändern Sie den Wert von 'Konfigurationsdatei für die Prozedurenaufnahme' auf den Pfad unserer Demokonfigurationsdatei nämlich `qftest-9.0.0/demo/procbuilder/carconfig-procbuilderdef.qft`. Danach klicken Sie auf 'OK'. Details über diese Datei erfahren Sie im nächsten Abschnitt.

Nachdem Sie die Konfiguration geändert haben, fahren Sie mit folgenden Schritten fort:

- Drücken Sie den 'Prozeduren erstellen' Knopf
- Klicken Sie mit der rechten Maustaste auf das SUT
- Wählen Sie 'Ganzes Fenster' aus
- Drücken Sie den 'Prozeduren erstellen' Knopf nochmals.

Jetzt erstellt QF-Test die Basisprozeduren für das Hauptfenster. Sie sollten nun unter Prozeduren ein Package namens `procbuilder` sehen. Dieses Package enthält weitere Packages und Prozeduren, die Aktionen für die einzelnen GUI Elemente und den gesamten Dialog beinhalten.

Hinweis Die aktuelle Konfiguration erstellt das Package `JCarConfigurator` als Container für alle Prozeduren innerhalb des Hauptfensters.

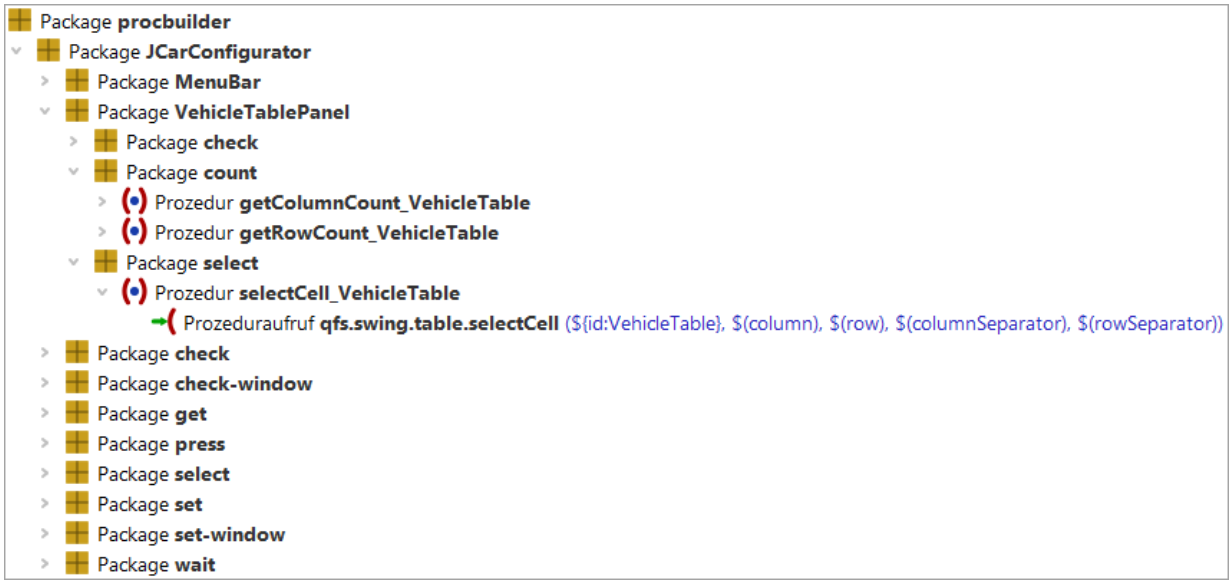

Abbildung 30.3: Die aufgezeichneten Prozeduren

Beachten Sie, dass die involvierten Komponenten auch unter Fenster und Komponenten aufgezeichnet worden sind.

Der nächste Schritt ist das Prüfen, ob die erstellten Prozeduren für uns nützlich oder einige überflüssig sind.

Werfen wir nun einen genaueren Blick auf die erstellten Packages:

Package	Inhalt
`JCarConfigurator`	Dieses Package beinhaltet alle Prozeduren für Aktionen auf Komponenten des JCarConfigurator Fensters. Dieses Package wurde aufgrund der aktuellen Konfiguration erstellt, welches die Hierarchie der Komponenten beachtet.

Tabelle 30.1

Das Package `JCarConfigurator` enthält folgende Packages:

Package	Inhalt
`MenuBar`	Dieses Package enthält alle Prozeduren für Aktionen auf Menüeinträge des SUT. In unserem Fall finden wir hier nur Prozeduren für das Klicken auf die einzelnen Menüeinträge.
`VehicleTablePanel`	Dieses Package enthält alle Prozeduren für Aktionen auf Komponenten auf den 'VehicleTablePanel' Panel des SUT. In unserem Fall finden wir hier nur Prozeduren für das Objekt 'VehicleTable', weil es die einzige Komponente auf diesem Panel ist.
`check`	Dieses Package enthält Prozeduren um Komponenten zu prüfen.
`check-window`	Dieses Package enthält Prozeduren um die alle Komponenten eines Fenster zu prüfen. Dies ist eine Containerprozedur.
`get`	Dieses Package beinhaltet Prozeduren um Werte von Elementen auszulesen und zurückzuliefern, z.B. das Auslesen eines Textes.
`select`	Dieses Package beinhaltet Prozeduren um Elemente auszuwählen. In unserem Fall gibt es eine Prozedur um einen Tab des TabbedPanes auszuwählen.
`set`	Dieses Package beinhaltet Prozeduren um Komponenten zu setzen. In unserem Fall sind das Setzmethoden für die Textfelder.
`set-window`	Dieses Package beinhaltet Prozeduren um alle Komponenten des Fensters mittels einen Prozeduraufrufes zu setzen. In unserem Fall ruft die erstellte Prozedur alle 'set' Prozeduren des JCarConfigurator Fensters auf. Diese Containerprozedur ist auch eine typische Workflowprozedur.
`wait`	Dieses Package beinhaltet alle Prozeduren, um auf diverse Komponenten zu warten.

Tabelle 30.2

In unserem Fall sind alle Prozeduren nützlich. Wir können also das gesamte `JCarConfigurator` Package aus dem `procbuilder` Package direkt unter Prozeduren verschieben. Bitte vergessen Sie nicht beim 'Referenzen aktualisieren' Dialog auf 'Ja' zu klicken.

Jetzt können die Prozeduren von Testfällen genutzt werden.

Schlussendlich sieht unsere Testsuite so aus:

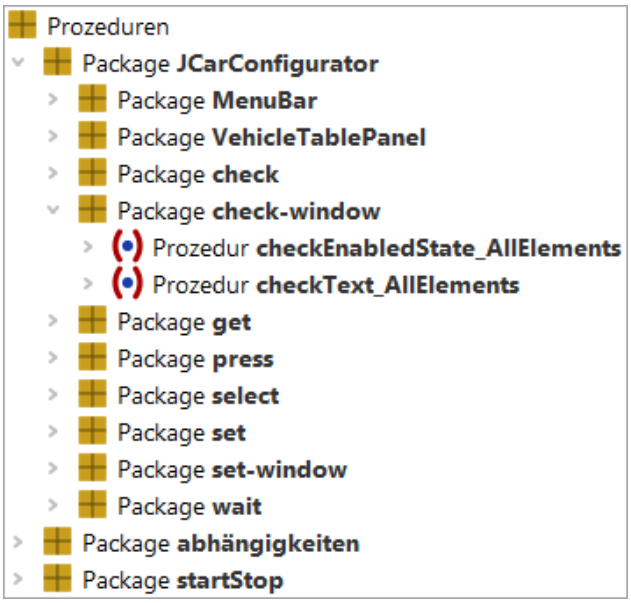

Abbildung 30.4: Die Testsuite mit den Prozeduren

Wiederholen Sie nun diese Aufnahme für das 'Sondermodelle' und das 'Zubehör' Panel. Sie müssen dann nur noch die neuen Prozeduren und Packages in das 'JCarConfigurator' Package verschieben. In unserem Fall sind das nur die Packages 'SpecialsPanel' und 'AccessoryTablePanel'.

Die vollständige Testsuite sieht nun so aus:

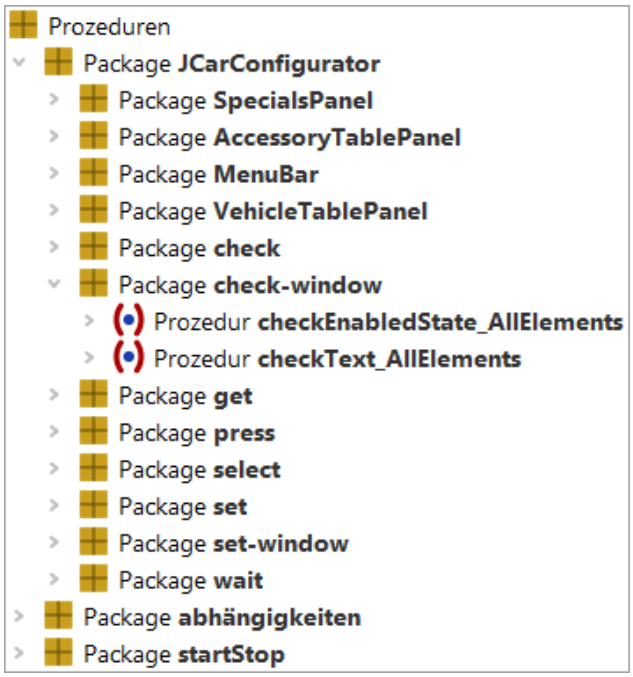

Abbildung 30.5: Die Prozeduren für alle Panels

Nun können wir wirklich die Testfälle mit den automatisch generierten Prozeduren erstellen. Sie können natürlich vorher noch Prozeduren für alle Dialoge, z.B. den 'Fahrzeug' Dialog erzeugen. Diesen Dialog erreichen Sie im SUT mittels 'Optionen' -> 'Fahrzeuge'. Ebenso gilt dies für den 'Sondermodelle' und 'Zubehör' Dialog.

Sie müssen nicht immer das gesamte Fenster aufzeichnen. Sie können auch durch Auswahl von 'Nur Komponente' Prozeduren für eine bestimmte Komponente generieren oder mittels 'Komponente mit Kindern' Prozeduren für ein gesamtes Panel erzeugen.

30.3 Konfiguration der automatischen Erstellung

30.3.1 Einführung

Im vorigen Beispiel haben wir die Datei `qftest-9.0.0/demo/procbuilder/carconfig-procbuilderdef.qft` als Konfigurationsdatei für die automatische Generierung benutzt. In diesem Abschnitt wollen wir uns die Konfigurationsmöglichkeiten von QF-Test genauer ansehen. Öffnen Sie hierzu diese Datei.

Abbildung 30.6: Die aktuelle Konfiguration

Das Package `procbuilder` ist das Wurzelpackage für alle erstellten Packages. Wenn Sie einen anderen Namen für dieses Package verwenden wollen, können Sie dieses einfach umbenennen.

Wenn Sie dieses Package öffnen, dann sehen Sie die 'Klassen' Ebene. Diese Ebene beschreibt die Klassen der GUI-Komponente, welche für die Erstellung berücksichtigt werden sollen. Die nächste Ebene beinhaltet dann die Informationen über die zu erstellende Packagestruktur und deren Prozeduren. Sie können eine detaillierte Beschreibung im Handbuch im Kapitel Die Procedure Builder Definitionsdatei finden.

30.3.2 Erstes Beispiel

In unserem ersten Beispiel wollen wir eine neue Konfigurationsdatei erstellen, welche wir selbst Schritt für Schritt aufbauen.

Bitte führen Sie folgende Schritte durch:

- Öffnen Sie eine neue Testsuite und speichern diese. Geben Sie der Testsuite einen Namen, wie 'mySettings.qft'.
- Erstellen Sie ein neues Package mit dem Namen 'myProcedures'.

Bis jetzt sieht die neue Testsuite folgendermaßen aus:

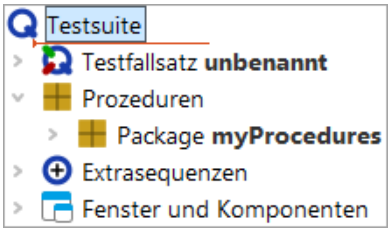

Abbildung 30.7: Die eigene Konfigurationsdatei

Jetzt sind wir bereit die Prozedurenvorlagen für spezielle Klassen zu erstellen. Wir sollten zuerst Prozeduren für Textfelder des Hauptfensters erstellen. In unserem Projekt könnte es interessant sein, den Inhalt der Textfelder zu prüfen, daher brauchen wir Prozeduren hierfür.

Um diese Prozeduren zu erstellen, müssen wir ein Package unter 'myProcedures' anlegen. Dieses Package sollte den Namen 'javax_swing_JTextField' haben. 'javax.swing.JTextField' ist die Klasse aller Textfelder, allerdings ist ein '.' nicht in Packagenamen erlaubt, deshalb ersetzen wir diesen mit '_'. Dieses Package wird QF-Test nun instruieren, Prozeduren für eine Komponente zu erstellen, sobald diese von der entsprechenden Klasse ist. Das ist sehr wichtig, damit wir unsere Prozedurvorlagen auf Klassenebene definieren können.

Nun erstellen wir eine Vorlage für die eigentliche Prüfprozedur der Komponenten. Die Prozedurvorlage sollte allerdings noch Teil eines weiteren Packages sein. Der Name des Packages sollte die Gruppe der Prozeduren bezeichnen, z.B. 'checkers'. Nach Erstellung des 'checkers' Package, sollten Sie eine Prozedur 'checkText' zu diesem Package hinzufügen. Die Prozedur sollte einen Knoten 'Check Text' beinhalten, welcher zum Prüfen von Texten geeignet ist. Bitte fügen Sie diesen 'Check Text'-Knoten mittels Rechtsklick und Auswahl von 'Knoten einfügen' -> 'Check-Knoten' -> 'Check Text' ein. Setzen Sie das Attribut 'client' auf `$(client)`, das Attribut QF-Test component ID auf 'dummy' und das Attribut 'text' auf `$(text)`.

Nach dem Bestätigen dieser Eingaben werden wir selbstverständlich eine Warnung erhalten, in der wir darauf hingewiesen werden, dass eine Komponente namens 'dummy' nicht existiert. Diese Warnung dürfen wir an dieser Stelle ignorieren.

Die Testsuite sieht nun wie folgt aus:

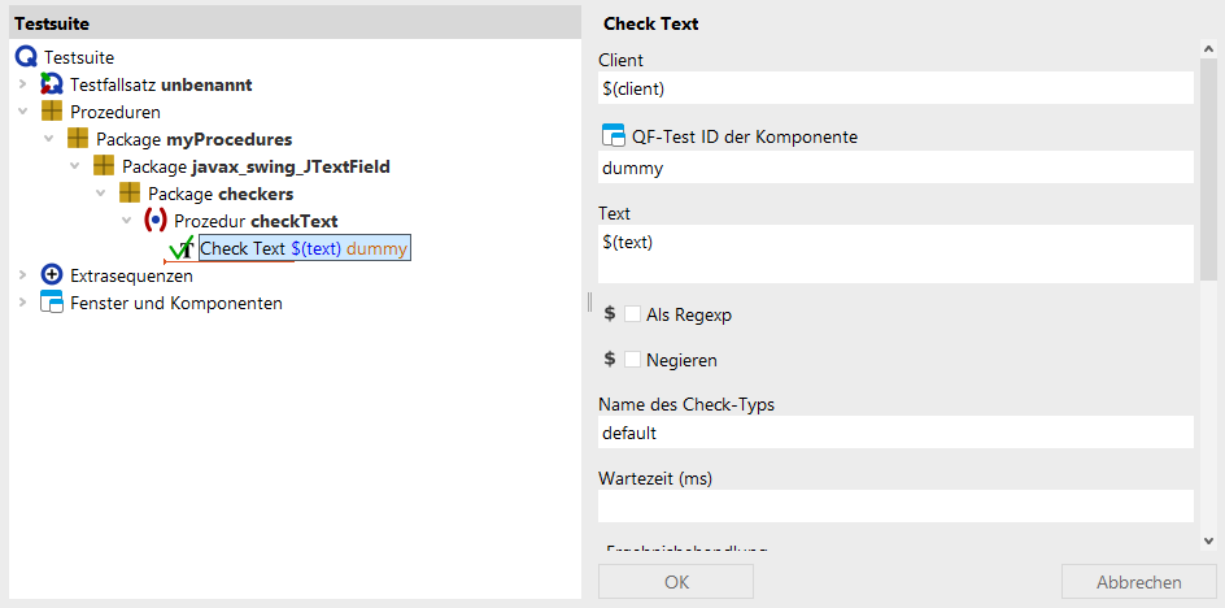

Abbildung 30.8: Die `checkText` Prozedur

Die Prozedur sollte einen Parameter 'text' mit einem leeren Standardwert beinhalten.

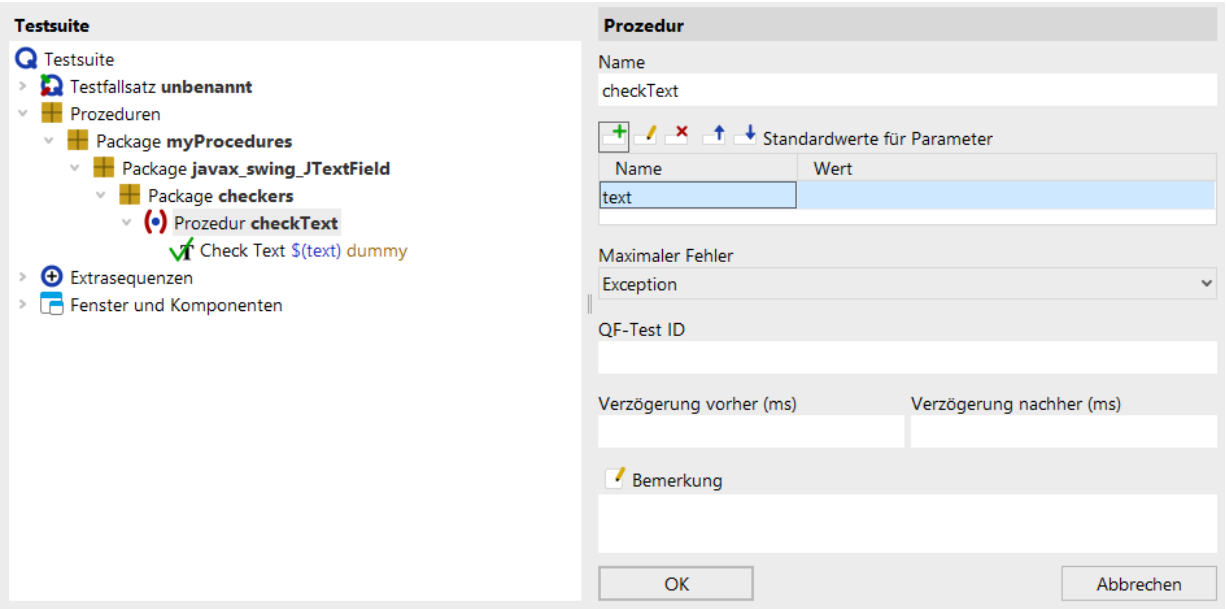

Abbildung 30.9: Die `checkText` Prozedur mit Parametern

Die erste Prozedurvorlage ist nun fast vollständig, jetzt müssen wir uns allerdings noch über einen Aspekt Gedanken machen. Jede Komponente hat ihre eigene und eindeuti-

ge QF-Test ID, also wäre es praktisch, wenn diese QF-Test ID bereits bei der Erstellung berücksichtigt wird, anstatt diese manuell nachzupflegen. Außerdem wollen wir, dass die Prozeduren komponentenbezogene Namen aufweisen statt einfach nur 'checkText'. Der Platzhalter `<COMPID>` steht in QF-Test für die QF-Test ID der aktuellen Komponente. Also müssen wir den Prozedurnamen in `checkText_<COMPID>` ändern. Wir sollten auch den Platzhalter `<COMPID>` direkt in das Attribut QF-Test component ID des 'Check Text'-Knotens einfügen.

Schlußendlich sieht unsere Prozedurvorlage wie folgt aus:

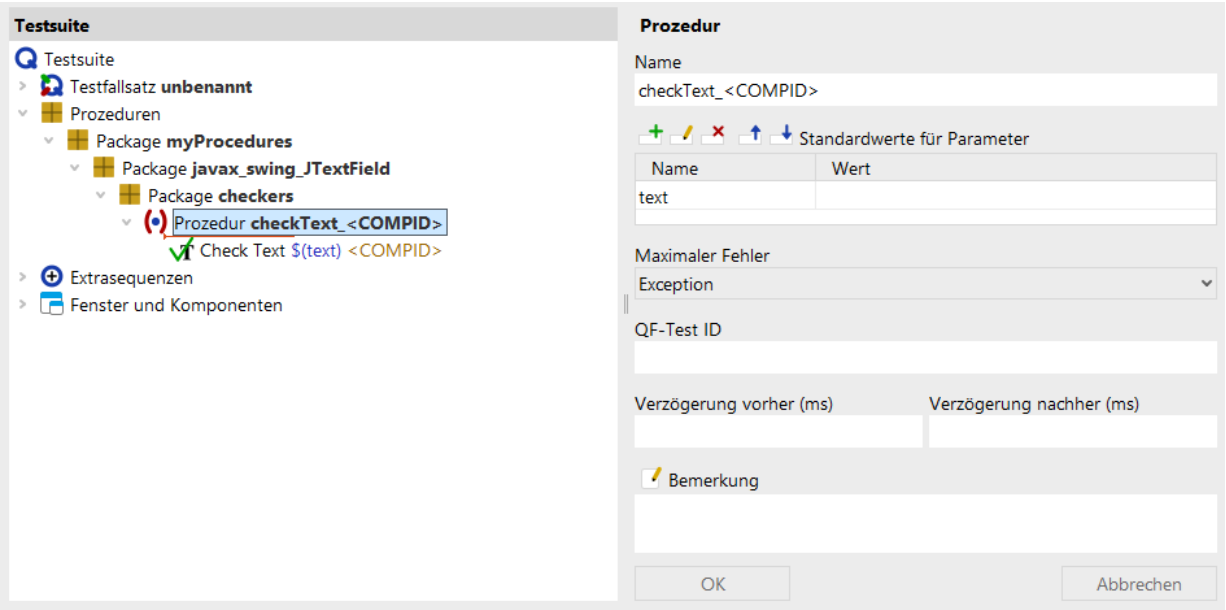

Abbildung 30.10: Der `<COMPID>` Platzhalter

Jetzt können wir unsere eigene Konfigurationsdatei verwenden. Hierfür müssen wir QF-Test noch mitteilen, dass diese Datei verwendet werden soll. Öffnen Sie die Optionen mittels Bearbeiten→Optionen und wechseln Sie in den 'Aufnahme' -> 'Prozeduren' Bereich. Dort setzen Sie den Pfad Ihrer eigenen Datei im Attribut 'Konfigurationsdatei für die Prozedurenaufnahme'. Danach bestätigen Sie die Änderung mit 'OK'.

Danach starten Sie den CarConfigurator. Wenn dieser vollständig gestartet wurde, fahren Sie mit folgenden Schritten fort:

- Drücken Sie den 'Prozeduren erstellen' Knopf
- Führen Sie Rechtsklick auf dem SUT aus
- Selektieren Sie 'Ganzes Fenster'

- Drücken Sie den 'Prozeduren erstellen' Knopf.

Gratulation! Sie haben eigene Testschritte mit QF-Test erstellt.

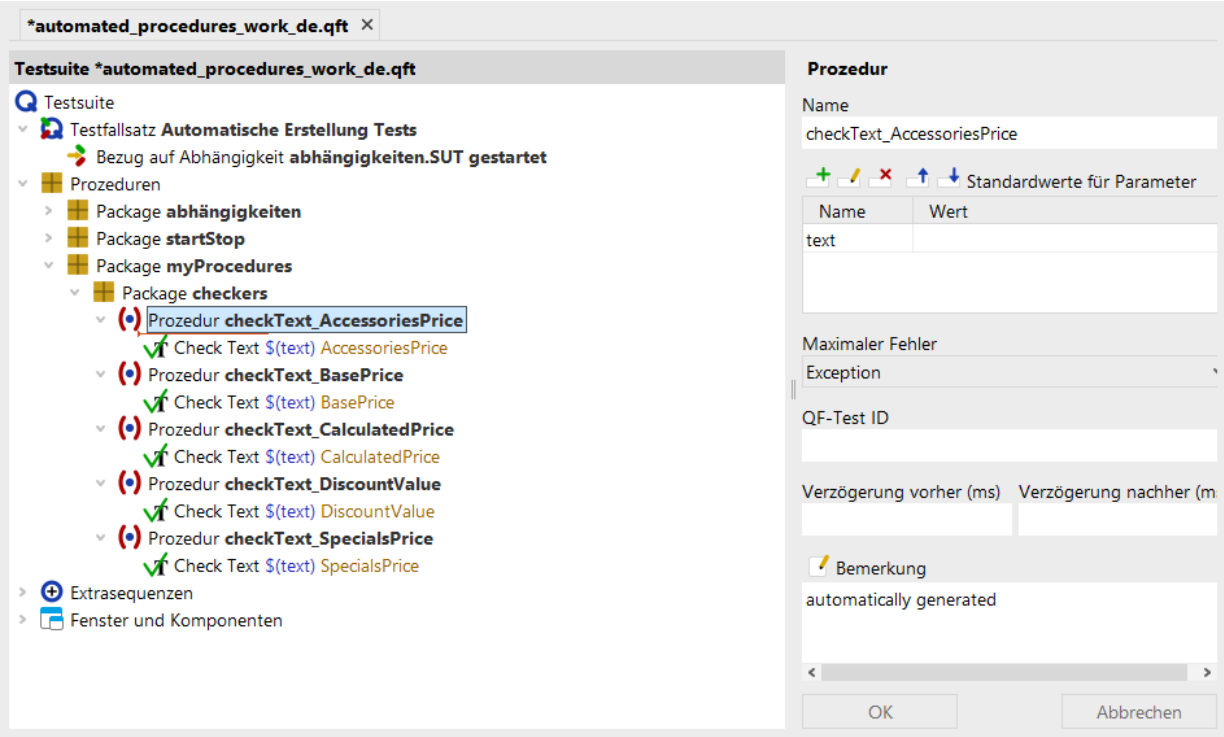

Abbildung 30.11: Die selbst erstellten Testschritte

30.3.3 Den aktuellen Text verwenden

Die 'checkText' Prozeduren haben den Parameter 'text' für den zu prüfenden Text. Bis jetzt müssen wir den zu prüfenden Text immer beim Aufruf der Procedure angeben. Stellen wir uns nun ein Szenario vor, in dem wir die Standardwerte der Textfelder nach dem Start prüfen wollen. In diesem Fall müssten wir jedem einzelnen Prozeduraufruf der vier 'checkText' Prozeduraufrufe den entsprechenden Parameterwert mitgeben. QF-Test bietet einen Platzhalter, um den aktuellen Text während der Prozedurerstellung einzubinden. Hierfür müssen wir in der Konfigurationsdatei den Standardwert des Parameters 'text' auf `<CURRENTVALUE>` setzen. Danach sollten Sie sich versichern, dass das Package `myProcedures` nicht mehr unter Prozeduren existiert, damit wir die Prozeduren neu aufzeichnen können. Falls dieses Package doch existieren sollte, wird ein neues Package `myProcedures1` erstellt um die Eindeutigkeit der erstellten Packages zu gewährleisten. Erstellen Sie nun die Prozeduren wie im vorigen Beispiel.

Die geänderte Konfigurationsdatei:

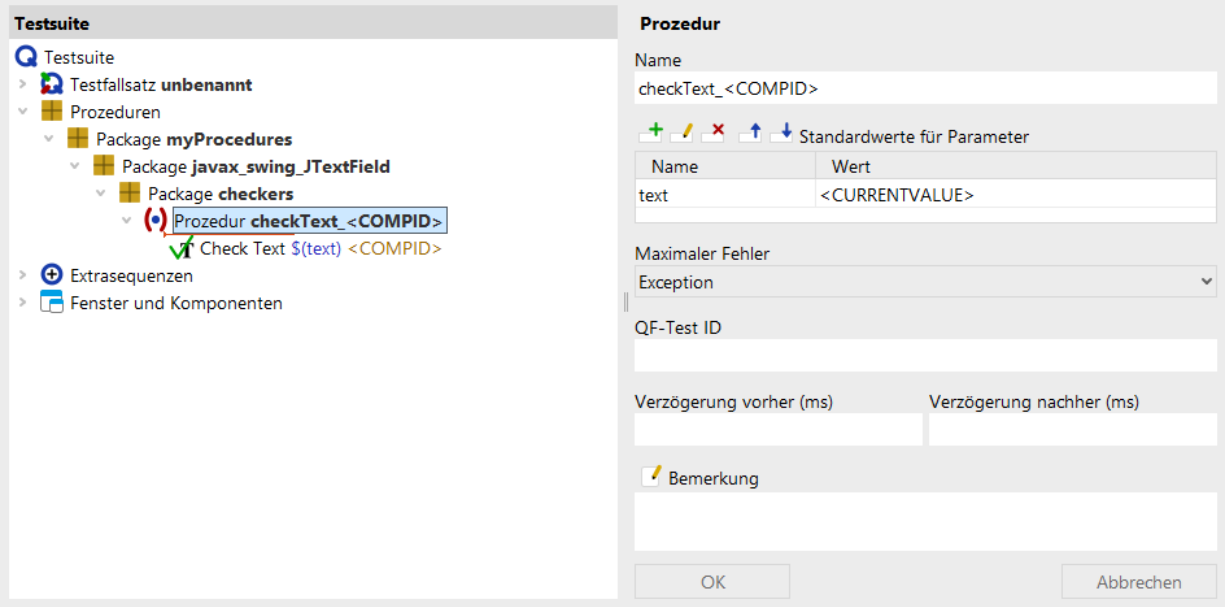

Abbildung 30.12: Die Konfigurationsdatei mit dem aktuellen Text

Die neu erstellten Prozeduren:

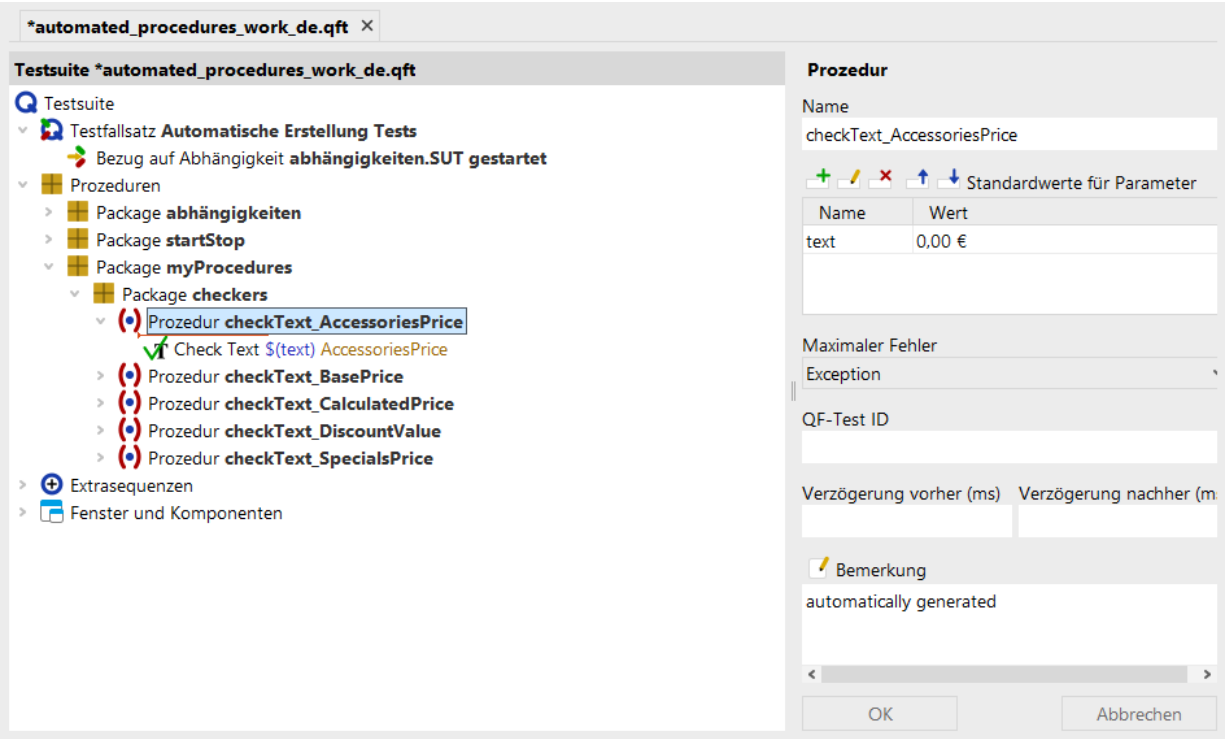

Abbildung 30.13: Die generierten Prozeduren mit dem aktuellen Text

30.3.4 Generieren von Container Prozeduren

Im vorigen Beispiel haben wir 'checkText' Prozeduren für die einzelnen Textfelder erstellt. Sie sollten nun selbstständig in der Lage sein 'set' Prozeduren, zum Setzen des Textes zu erstellen. Bis jetzt haben wir allerdings nur mit einzelnen Komponenten gearbeitet.

In einigen Testszenarien kann es durchaus von Interesse sein, dass man mit einem Prozeduraufruf alle Textfelder eines Dialoges oder eines speziellen Panels prüft. Ein weiteres Szenario wäre das Setzen aller sichtbaren Textfelder. Solche Prozeduren arbeiten mit Container-Komponenten, deshalb heißen diese Prozeduren Container Prozeduren

In unserem Fall wollen wir nun eine Prozedur erstellen, die alle vier 'checkText' Prozeduren des CarConfigurator aufruft. Aber wie erstellen wir diese?

Zuerst müssten wir ein weiteres Klassenpackage in 'mySettings.qft' einfügen. Der Name des Packages sollte 'javax_swing_JFrame' sein. Das Hauptfenster des CarConfigurator ist eine Instanz von `javax.swing.JFrame`, deshalb sollten wir dieses Klassenpackage erstellen. Darin sollte wir ein Typpackage namens 'checkers-window' erstellen. Das Typpackage sollte wiederum eine Prozedur `checkTextOfElements_<COMPID>` beinhalten, welche die einzelnen Checks aufruft. Wir benutzen hier den Platzhalter `<COMPID>`, damit wir erkennen können zu welchem Dialog die erzeugte Prozedur gehört.

Der nächste Schritt ist das Spezifizieren des Prozedureninhaltes. Schauen wir mal, wie man das macht.

Wir haben vier Textfelder, welche alle mittels der Prozedur `checkText_<COMPID>` geprüft werden können. QF-Test ermöglicht es uns nun, alle vier Prozeduraufrufe mit nur einem zu konfigurieren. Daher fügen Sie bitte einen Prozeduraufruf Knoten in diese Prozedur ein. Sie müssen dann die Prozedur `javax_swing_JTextField.checkers.checkText_<CCOMPID>` aufrufen.

Die Konfigurationsdatei nach dieser Änderung:

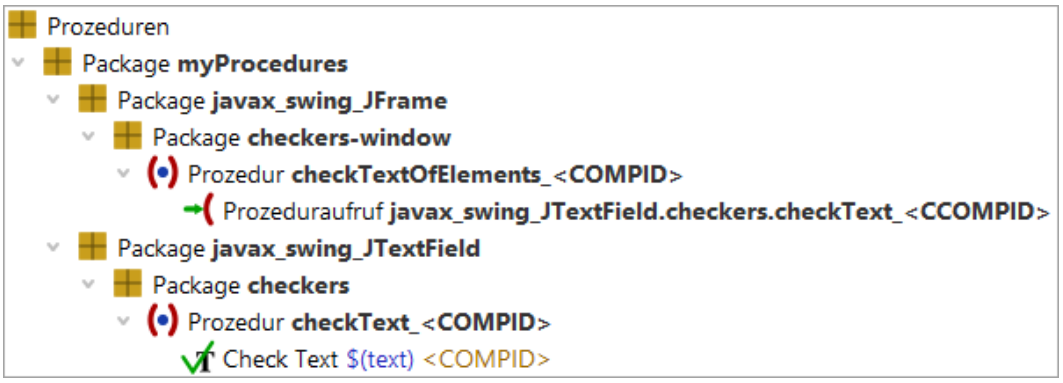

Abbildung 30.14: Die Vorlage für die Containerprozedur

Als letzten Schritt muss man QF-Test so konfigurieren, dass auch wirklich eine Containerprozedur und keine normale Komponentenprozedur erstellt wird. Hierfür muss man den Wert `@FORCHILDREN` im Attribut 'Bemerkung' der Prozedur `checkTextOfElements_<COMPID>` eintragen.

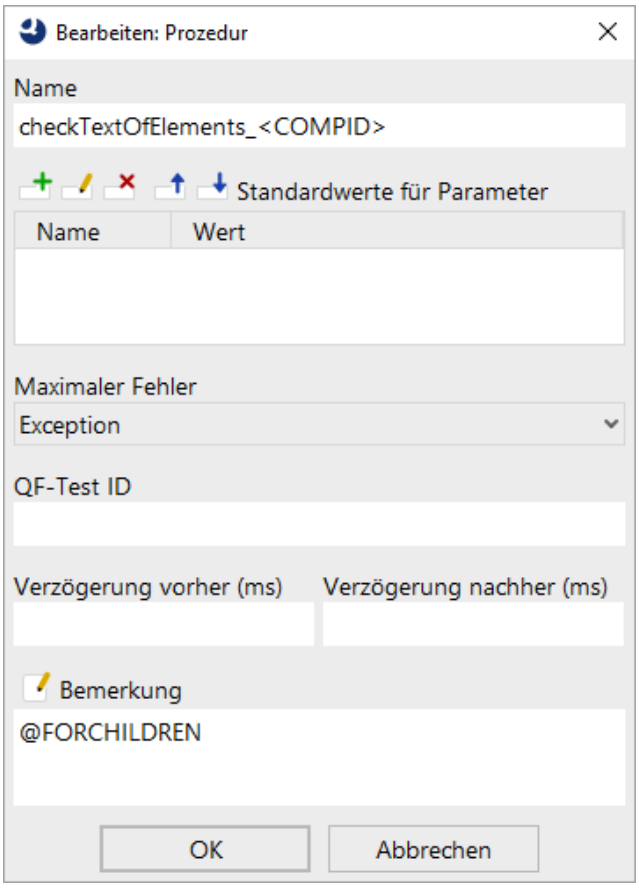

Abbildung 30.15: Die Verwendung von `@FORCHILDREN`

Jetzt können Sie die Prozeduren, wie im vorigen Beispiel aufzeichnen. Vergessen Sie bitte nicht das 'myProcedures' Package vorher aus Prozeduren zu löschen. Jetzt sollten unter Prozeduren folgende Prozeduren erstellt werden:

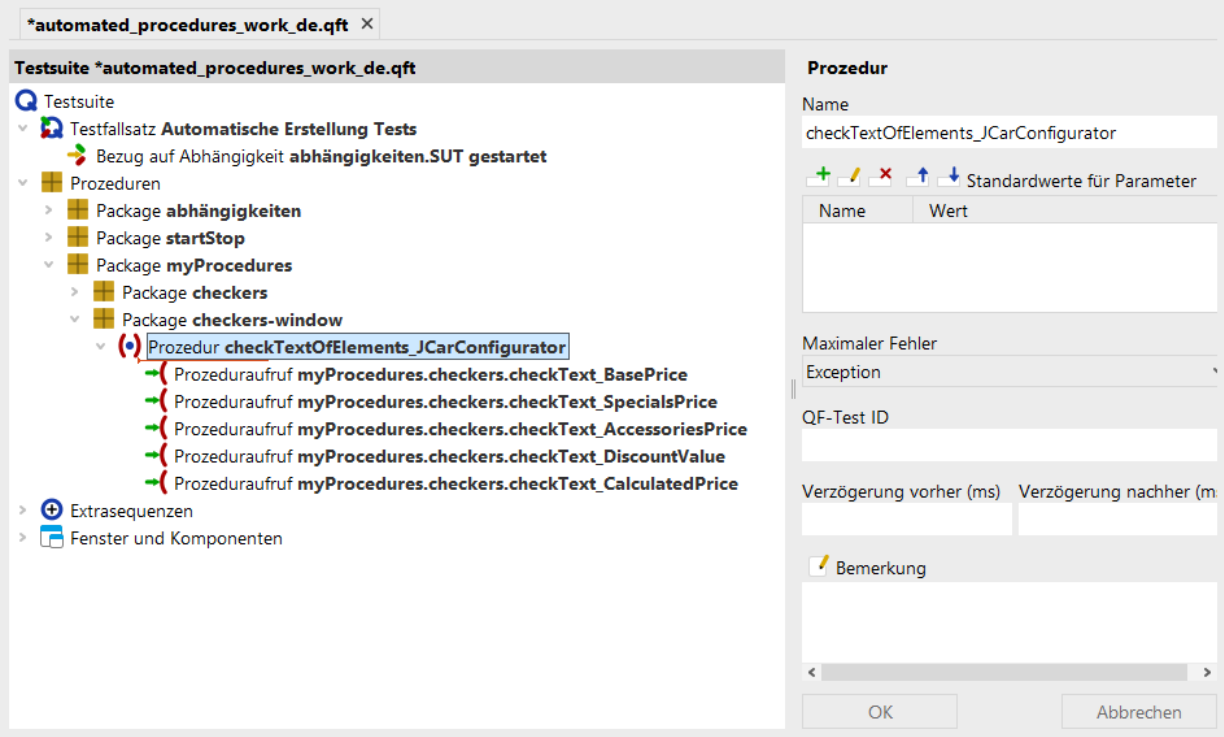

Abbildung 30.16: Die generierten Containerprozeduren

Hinweis QF-Test ersetzt den 'Klassen' Teil des Prozeduraufrufes durch den Namen des Konfigurationspackages. In unserem Fall ist dies 'myProcedures'.

30.3.5 Der aktuelle Wert der Kindkomponente

Wir können uns eine ähnliche Situation wie in Den aktuellen Text verwenden[346] beschrieben, auch für die Containerprozeduren vorstellen. Im obigen Beispiel haben wir `<CURRENTVALUE>` hierfür benutzt. Jetzt müssen wir den Parameter 'text' bei den einzelnen Prozeduraufrufen in der Containerprozedur 'checkTextElements' setzen. Dafür fügen wir diesen Parameter zum Prozeduraufruf in unserer Konfigurationsdatei 'mySettings.qft' hinzu. Der Wert sollte auf `<CCURRENTVALUE>` gesetzt werden.

Abbildung 30.17: Konfiguration mit <CCURRENTVALUE>

Wenn wir jetzt die Prozeduren nochmals erstellen, werden Sie sehen, dass die aktuellen Werte zu den Prozeduraufrufen hinzugefügt worden sind. Vergessen Sie bitte wieder nicht, vorher das 'myProcedures' Package aus Prozeduren zu löschen.

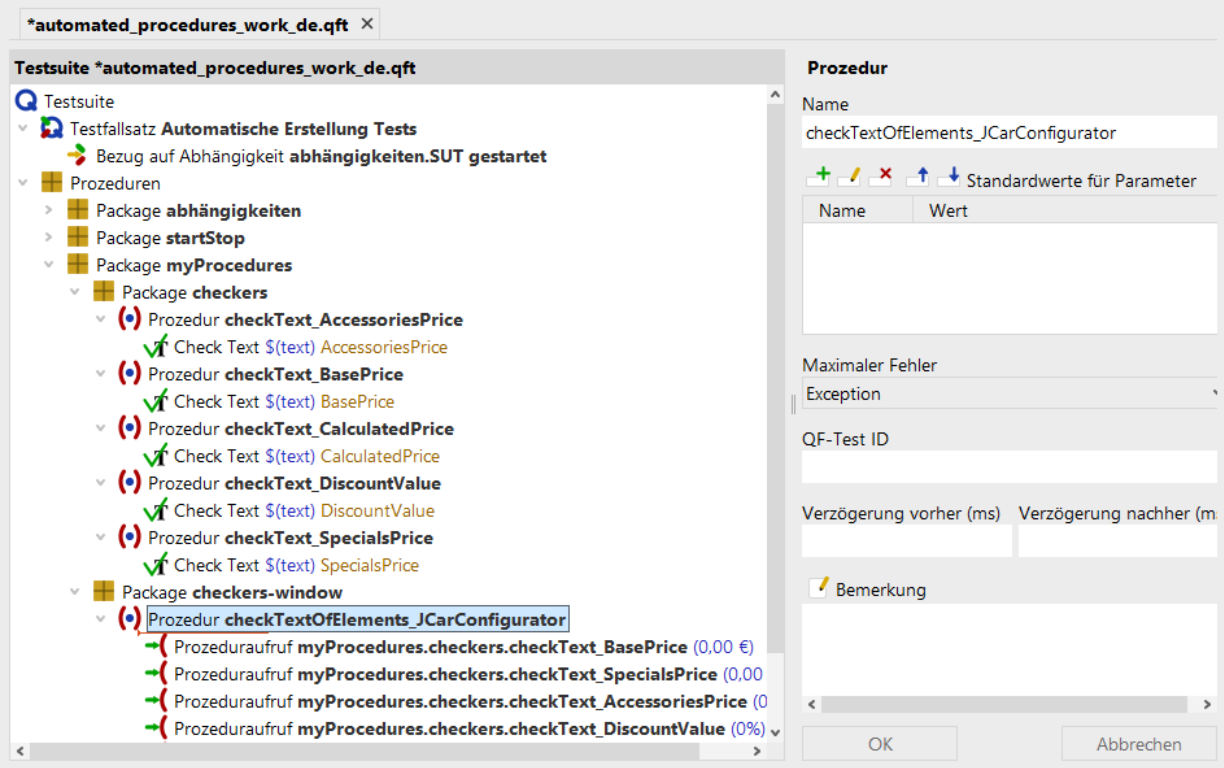

Abbildung 30.18: Testsuite mit <CCURRENTVALUE>

Wenn Sie sich die generierte Prozedur 'checkTextElements' genauer anschauen, so

werden Sie bei jedem Prozeduraufruf den gesetzten Text sehen. Vielleicht wäre es von Vorteil wenn dieser 'text' Parameter auch als Standardwert der Containerprozedur zur Verfügung stünde. Um dies zu erreichen, müssten Sie einen weiteren Parameter zur Prozedurvorlage hinzufügen. Der Name des Parameters ist `<CCOMPID>` und der Wert `<CCURRENTVALUE>`. Danach müssen Sie den Wert des 'text' Parameters beim Prozeduraufruf in `$(<CCOMPID>)` ändern. Die Konfiguration sollte also wie folgt aussehen:

Abbildung 30.19: Parameter für Containerprozeduren

Wenn Sie nun wiederum die Prozeduren erstellen, dann werden Sie sehen, dass die 'checkTextElements' Prozedur vier Parameter bekommen hat und die aktuellen Werte der Textfelder jeweils als Standardwerte eingetragen worden sind. Darüber hinaus hat jeder einzelne Prozeduraufruf für den Parameter 'text' eine Variable als Wert, die wie die QF-Test ID der Komponente heißt, welche auch gleichzeitig der Name des Parameters ist.

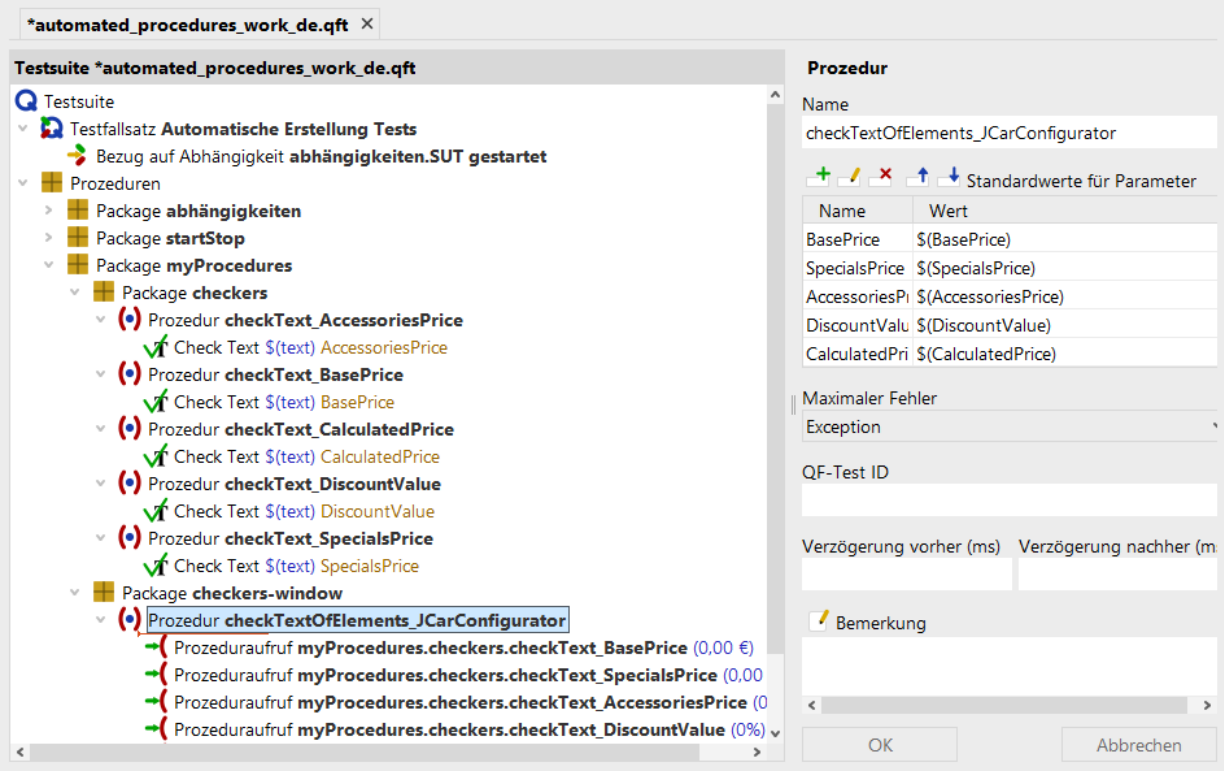

Abbildung 30.20: Parameter für die Containerprozedur in der Testsuite

30.3.6 Weitere Konfigurationsmöglichkeiten

Wie Sie in den vorigen Abschnitten gesehen haben gibt es eine Menge Konfigurationsmöglichkeiten für die automatische Prozedurenerstellung. Es gibt jedoch noch weitere Möglichkeiten. Um mehr darüber zu erfahren schauen Sie bitte ins Handbuch ins Kapitel Die Procedure Builder Definitionsdatei.