Chapter 15

Testing native Windows applications

5.0+

15.1 Getting started

This chapter covers automation and testing of Windows desktop applications, namely

- Classical Win32 applications,
- .NET applications based on Windows Presentation Foundation (WPF) or Windows Forms,
- Universal Windows Platform (UWP) applications using XAML controls.

All these kinds of applications support Microsoft UI Automation or Microsoft Active Accessibility (MSAA) for software test automation, joined together in the Windows Automation API, please see section 15.2(157) for background information.

As a precondition of testing, QF-Test first needs a connection with the respective process. An appropriate Setup sequence can be created by help of the Quickstart Wizard(28) from the Extras menu. Choose 'A native Windows application' as application type then.

If your application is already up and running, just specify a regular expression for the title of its (main) window, for example `.*- Editor` for the Windows Notepad application. Otherwise give the path to your Windows executable (.exe) so that QF-Test can start the program, please see Launching/Attaching to an application(157) for details.

Once the application is connected to QF-Test (as GUI engine(532) `win`), capture and replay can be performed as described in chapter 4(33). However, due to the nature of UI Automation, you should observe the recording rules listed in section 15.4(158).

A few example test-suites can be found in the QF-Test installation folder, namely

- `qftest-5.0.0-ea/demo/carconfigForms/winDemoForms_en.qft`

- qftest-5.0.0-ea/demo/carconfigWpf/winDemoWPF_en.qft
- qftest-5.0.0-ea/demo/windows/*.qft

Also have a look at the (Current) Limitations[162], most of which are expected to be fixed or improved in future releases of QF-Test.

15.2 Technical background

A common framework for all Windows-based applications is the Windows Automation API consisting of Microsoft's Active Accessibility and its successor, Microsoft UI Automation. Being it the heart of the win engine, QF-Test is now able to control virtually all kinds of Windows applications.

A Windows application has to expose so-called Providers in order to follow the rules of UI Automation. This is done automatically when a framework like WPF is used to develop a program. This is also done for Win32 applications via proxy providers. That means, how good an application can be controlled and tested depends on the quality of the respective providers, i.e. usually on the framework used in application development. For example, as UI Automation was introduced along with WPF, these kinds of applications are supposed to be testable quite well. And when you wonder what cannot be tested via UI Automation, try out a Java Swing application. However, there is already a pretty good tool available when you want to deal with Swing ...

A test application that wants to control a program via UI Automation can get hold of so called Automation Elements which represent the actual UI elements in the SUT (System Under Test). Though every automation element has a control type (like Button, MenuItem, etc.), its actual functionality - for example, setting a value in a text field - depends on Control Patterns implemented by the respective providers.

To deal with the UI Automation framework, QF-Test starts a special Java program which serves as UI Automation client application. That program can access all UI Automation elements in a given process and handle them according to the rules of QF-Test (e.g. create a snapshot of an element as Component[667]).

15.3 Launching/Attaching to an application

Testing a native Windows application does not require you to launch that application from QF-Test. You can also connect to a running process and that way even control parts of the operating system, for example the Windows Taskbar.

In order to connect to a process you can specify a regular expression for the window title or the respective process ID or the window's UI Automation class name. Strictly

speaking, that window must not be a `Window` but could also be a `Pane` or a `Menu` in terms of UI Automation control types. Whatever feature is used for attaching, QF-Test will eventually determine the respective process ID and treat exactly that process as the actual client application (SUT).

To connect just define the attribute Window title (RegExp)[555] in the Start Windows client[555] node and this can be

- a regular expression for the title
- -pid <process ID>
- -class <class name>

For example, by specifying `.*- Editor` you can attach to a running Windows Notepad application, while `-class Shell_TrayWnd` will address the Windows Taskbar.

In order to find out the titles, process IDs and class names of running programs, you can run the procedure `qfs.autowin.logUIAToplevels` in `qfs.qft`, cf. The standard library[115].

Besides attaching to a running process, it is also possible to launch a program from the Start Windows client node. To this end, specify the path to the respective executable in the Windows application[555] attribute.

In some cases, it can also be useful to define both the Windows application and the Window title (RegExp) attribute. QF-Test will then first try to attach. If that fails, the given program will be started and connected via its process ID. If that fails too - the process may launch a child process and terminate itself or may not display a (graphical) user interface - another attempt to attach is made.

When you terminate a `win` client in QF-Test (either via the Stop client[558] node or from the Clients menu), the respective UI Automation client process will be stopped along with its sub-processes. That is, your actual SUT will terminate as well, if you started it from QF-Test. On the other hand, the SUT will not be stopped when it was running before you attached to it.

When you close the SUT, the UI Automation client will terminate as well.

To attach to an elevated processes (presenting the UAC prompt), you have to launch QF-Test as adminstrator.

15.4 Recording

After connecting QF-Test with the SUT, you can record events (section 4.1[33]), checks (section 4.3[36]) and components (section 4.4[37]).

However, as the communication between the SUT and the QF-Test UI Automation client is handled by Windows (the UI Automation core), accessing elements is not quite as fast as you may know it from the QF-Test Java automation. Furthermore, in contrast to Java and Web testing (QF-Driver), events are processed asynchronously, i.e. you cannot expect that an application's dispatch thread is blocked while QF-Test is handling an event.

That makes recording more difficult, because a target element might be destroyed by the action to be recorded, for example when selecting an entry from a ComboBox or clicking on a button that closes its parent window.

So you'd best get into the habit of following a few recording rules:

- Activate the recording mode and move the mouse over the element for which you want to record an event.
- As QF-Test may take a little time to retrieve information about the element below the mouse cursor, a red pane is displayed until it is done; the little 'QF-Test Element Information' window will then show which automation element was found.
- Now perform the mouse click to be recorded.
- When a mouse click will close a dialog or window (might also be a popup displaying a list), make sure to perform the click slowly, i.e. do not release the mouse button immediately after pressing it so that QF-Test has the opportunity to gather information before the window will disappear (when the mouse release is done).
- When recording checks or components, the respective frame around the element is drawn almost immediately when the mouse is hovering over an automation element. Before recording a check, you should wait until the frame disappears.
- When the SUT shows two or more windows, make sure that they do not overlap before entering the check recording mode.

Sometimes check recording (and transforming the node afterwards) may work better than event recording, for example when a click on a button (like OK, Cancel) closes the respective dialog or when a mouse down event recreates elements (for example the accessory table in the CarConfiguratorNet WPF demo application). In check recording mode QF-Test covers the SUT with an (almost) invisible window to prevent mouse clicks from triggering an action in the client application.

15.5 Components

In QF-Test an automation element is recorded (or can be inserted manually, of course) as Window(657) or Component(667) and stored within the Windows and components(678) node.

The QF-Test (generic) Class name often corresponds to the type of the UI Automation element, for example `Button`. To be able to differentiate between the UI Automation type and the generic class name, QF-Test adds a prefix `Uia.` to the type. Similarly, the UI Automation framework specifier is used as prefix for the automation element's class. So you may for example see a `classname: WPF.DataGrid` in the Extra features of a Table component recorded in a WPF application.

QF-Test does not strictly follow the hierarchy of the UI Automation elements. That is often the case with dialogs (like Notepad's Font dialog) which are usually listed below the main application window in the UI Automation tree. From the Win32 perspective as well as what QF-Test users would expect, such dialogs are also top-levels and thus listed as a sibling of the main window below Windows and components. On the other hand, a context menu can be a top-level in the UI Automation tree, but may be a window's child in QF-Test.

15.6 Playback and Patterns

The QF-Test `win` engine usually records mouse clicks with the Replay as "hard" event attribute. That is the safe way, i.e. a "hard" mouse click is likely to actually trigger the desired action.

However, UI Automation supports various "soft" actions which do not rely on mouse events. For example, to trigger a button's action you can play back

```
+Select:  invoke [myButtonID]
```

The effect should be the same as with

```
+Mouse click [myButtonID]
```

but no mouse is involved when using the Selection node. Instead, the UI Automation core will trigger the execution of a provider's `Invoke()` method in the SUT.

The Selection node does support the following actions in its Detail attribute:

- `invoke`: Usually equivalent with a mouse click.
- `expand, collapse`: Should expand/collapse a ComboBox, MenuItem or TreeItem.
- `select`: Should select an item in a list (also use 0 as Detail). 1 and -1 are meant to extend or reduce the selection when multi-selection is allowed.
- `scroll:horiz%,vert%`: Values between 0 and 100 are possible, defining the position of the scroll location in percent; specify -1 when you do not want to change a position (horizontally or vertically).

The actions actually supported depend on an automation element's patterns. They are recorded among the Extra features of a component or can be determined in an SUT script (see below).

What exactly a pattern means can vary from application to application. If, for example, both `SelectionItem` and `Invoke` patterns are supported, `Invoke` might be preferable because

```
+Select [list@item]
```

may only highlight the element but not trigger the respective action (e.g. Notepad Fonts).

Even worse, the formal support of a pattern does not mean that applying it has any effect, for example scrolling an (invisible) entry in the list of Windows Calculator's modes into view (`ScrollItem` pattern). To get around the problem, you can simply play back `select` here, whether or not the entry is currently visible.

As already mentioned above, because "soft" playback may simply not work (due to the provider implementation) or even block when a modal dialog is to be displayed (due to COM method invocation), QF-Test records "hard" mouse clicks by default. If you deactivate the Replay as "hard" event attribute, `invoke` is played back instead (only if the respective pattern is supported by the element).

Regarding Key events, a text can only be set directly by a Text input node if the `Value` pattern is supported. Otherwise single key events have to be played back.

15.7 Scripting

Internally, the `win` engine represents automation elements by a class `WinControl`. To obtain an element in a Groovy SUT script(530) node, run

```
def ctrl = rc.getComponent("myComponentID")
println ctrl
```

Example 15.1: Retrieving a `WinControl` in a Groovy SUT script

with the respective QF-Test component ID. The most important WinControl methods are

- `getType()`, `getClassName()`, `getFramework()`, `getName()`, `getId()`, `getHwnd()`, `getLocation()`, `getSize()`, `getLocationOnScreen()`, `getPatterns()`, `hasPattern()` to retrieve UI Automation properties of the element
- `getChildren()`, `getParent()`, `getChildrenOfType()`, `getAncestorOfType()`, `getElementsByClassName()` to traverse the element hierarchy

- `getControl()` to retrieve the respective control from the mmarque ui-automation library. A snapshot of that library is part of the QF-Test releases.

15.8 Options

The behaviour of the `win` engine can be influenced via QF-Test options. For example, you may sometimes want to play back an event on an element that is actually not visible (it may not be scrolled into view). To perform an `invoke` event then, you may need to get rid of the visibility test which is usually part of the component recognition. That is, run

```
rc.setOption(Options.OPT_WIN_TEST_VISIBILITY, false)
```

in an SUT script node before playing back

```
+Select:  invoke [myComponentID]
```

and afterwards

```
rc.unsetOption(Options.OPT_WIN_TEST_VISIBILITY)
```

to reset the original setting.

Another means to adapt QF-Test's behaviour is to set parameters for the (native) WinDriver part serving as interface between Java and the Windows UI Automation. This is also done in an SUT script via `rc.engine.preferences()`. For example,

```
def prefs = rc.engine.preferences()
prefs.setPref("windriver.restrict.tops.to.class", "false")
```

Example 15.2: Setting a preference in a Groovy SUT script

will revoke the (default) limitation to toplevels of the given class when using `-class <class name>` to attach (toplevels of another class in the same process can then be accessed too).

15.9 (Current) Limitations

There are a number of limitations in the current implementation status of the Windows testing functionality. We will try to further improve things within the next versions, but possibly not all of the following points will be resolved soon.

To avoid trouble with geometry (bounds of an element, coordinates for a hard mouse events) the Windows scaling should be set to 100% when recording and replaying events. This is supposed to be fixed in a future release of QF-Test.

As the support for UI Automation depends on the framework used for application development, the recording in QF-Test may not always be consistent. For example, a Wait for component to appear node may or may not be recorded when opening a dialog.

Dealing with applications consisting of several processes requires several `win` clients and can be tricky.

Further limitations / not yet implemented features (Jan 2019) are among other things

- Component recording does work only for a single component (not for children or the full window). This is supposed to be fixed in a future release of QF-Test.
- Supported check types are incomplete. 'Text', 'Image', 'Geometry' and 'Visible' are implemented for all WinControls. Additional checks may be available for Tables, Lists and other types. This is supposed to be fixed in a future release of QF-Test.
- Elements in the title bar of a Windows app cannot be accessed (easily), because they live in a different process. This might be improved in a future release of QF-Test.
- MenuItems in .NET Forms application are not properly recorded, you only get a mouse click on a menu popup with the respective location. Replay should work though. This might be fixed in a future release of QF-Test. As a workaround you may create yourself a generic MenuItem as described in section 33.6(331).
- Redirection from a Button's Text element to the Button element is done when recording a mouse click, but may be missing elsewhere. This is supposed to be improved in a future release of QF-Test.
- Prefer Groovy to Jython in SUT script nodes. The latter does not yet support `WinControl.getControl()`. This is supposed to be fixed in a future release of QF-Test.
- Big hierarchies of automation elements may cause performance problems. This is supposed to be improved in a future release of QF-Test.

15.10 Links

The Windows Automation API is described here: https://docs.microsoft.com/en-US/windows/desktop/WinAuto/windows-automation-api-portal.

More about Mark Humphrey's ui-automation Java library can be found on https://github.com/mmarquee.