12 Robust Mobile Test Automation Using Smart Object Identification

THIS CHAPTER WAS CONTRIBUTED BY UZI EILON, CHIEF TECHNOLOGY OFFICER (US) AT PERFECTO

UZI EILON is the Chief Technology Officer (US) at Perfecto. He joined Perfecto in 2010 after a fifteen-year career as a software developer and manager at IDF, Netrialty, Comverse, and SanDisk. Over the past seven years, Uzi has grown the company by managing expanding R&D teams and leading Sales Engineering teams. His fields of expertise includes mobile application testing, automation tools, defining customer projects, and on-boarding, plus bringing Agile methodologies into the equation. Uzi Eilon speaks regularly on behalf of Perfecto at events, such as AnDevCon, StarWest, HP Discover, and ongoing technical webinars.

INTRODUCTION

It is clear that getting native objects identified properly as part of the test automation process is key to successful continuous test automation. Selenium allows test automation engineers to identify objects[1] by 7 different locators such as XPATH, CSS, ID, Text, Class, etc.

Within this chapter, we will go through some of them and we will introduce a brand new tool[2] that could help you improve the reliability of object identification in Selenium tests. The tool was developed and open-sourced by the Office of the CTO at Perfecto. Think about it as an object analysis grader where the highest score would mean that the test engineer found/identified the best-matched object name to be used in the test code. It also provides valuable recommendations on how the corresponding expressions could be improved in a way that they would identify objects well across different platforms.

OBJECT ID AND AUTOMATION

Getting the most accurate object as part of any automation test code delivers several key benefits, such as:

- Test code that is easy to author and debug
- Highly maintainable test code which is more resilient to changes in the app under test
- Reusable automated tests across different versions of the app under test
- Cross-platform automated solutions (mobile and desktop web) can be built
- Reduce false-negative errors

Stable test code is not the only prerequisite for a continuously working automation. It is also a matter of a well-defined process and collaboration between the various teams. Before diving into the technical talk about XPath, it will be worth it to discuss a process used successfully by agile devOps teams. It will help testers write better automation code that can be easily integrated in the

1 Selenium object locators — http://www.testinginterviewquestion.com/p/object-identification-in-selenium.html

2 XPath Validator tool — http://xpathvalidator.projectquantum.io/

early stages of CI workflows.

Both mobile and web developers will typically define the identifiers of UI objects as part of their application code.

In a mobile web/ HTML app, it will be through adding the ID attribute to HTML elements, as also depicted in **Figure 66.**

```
<input type=”text” placeholder=”ZIP Code” maxlength=”5”
id=”zip” name=”zip”>
```

Figure 66: An Example of a HTML Text Field That Could Be Located By Its ID

In a native mobile app, it would be through adding the ID to the UI control object (Android and iOS), as one can observe in **Figure 67** and **Figure 68.**

```
<TextViewandroid:textAppearance="?android:attr/textAppear-
anceMedium"
    android:text="My ip"
    android:id="@+id/my_ip"
    android:layout_alignStart="@+id/btn_cam"
    android:layout_marginTop="35dp"
    android:layout_below="@+id/mimageView" />
```

Figure 67: An Example of Defining Text Field That Could be Located by ID in an Android App

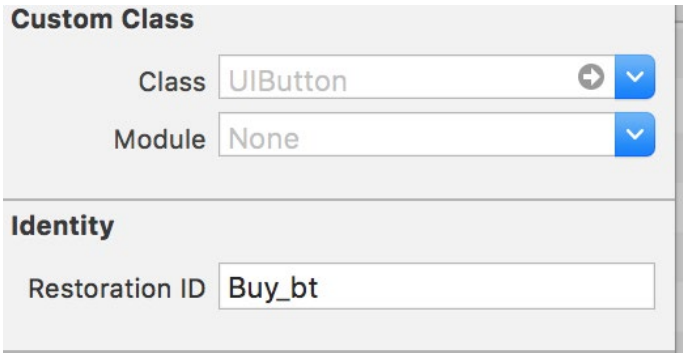

Custom Class
Class UIButton
Module None
Identity
Restoration ID Buy_bt

Figure 68: An Example of Defining UI Control That Could be Located by ID in an iOS App

At the beginning of each sprint, both **developers and testers** would define the UI objects together (including the corresponding objects IDs) and create a joint repository that will be used once by the developers while writing the code and, later on, by test engineers during the test automation development.

Both the code and the tests will be committed together into the main source code management system (SCM) and will be executed as part of the CI cycle.

The process of defining the objects jointly by developers and testers is critical for writing robust automation code. In addition, this process helps developers understand the testing requirements and code with testing in mind which is clearly a win-win situation for the entire mobile team.

XPATH IN A NUTSHELL

The process described above requires developers and test engineers to work together. Unfortunately, in many projects this is not the case and testers usually receive the app under test without any metadata about UI objects, i.e., their IDs are not defined. This is where XPath locators come in handy.

XPath is a syntax of defining parts of an XML objects tree. In Selenium and Appium tests, the structure of the displayed page/screen is described by a hierarchy which is accessed as an XML document in memory. For automation purposes, XPath expressions can be defined and run against this document object in order to locate specific UI objects (often represented as sub-trees in the corresponding XML object trees). Selenium and Appium make object identification through XPath expressions very easy, so you will not need to deal with any in-memory representations of XML object trees. However, this is how it works under the hood and this is why XPath is a relevant tool for all mobile test automation engineers.

XPath structure:

```
//object_type_n1/ object_type_n2[@attribute =value … ]
```

XPath expressions can be very powerful object locators. Table 2 lists three typical examples of identifying certain HTML UI elements (objects) in web pages rendered by web browsers.

Xpath Expression	**Definition**
//button[@type='submit']	Matches any button element of type "submit"
//input[@name='zip']	Matches any input element named "zip"
//div[@dir='ltr'] /input	Matches any input field element within a div with an attribute "dir", which has a value of "ltr"

Table 2: Sample XPath Expressions Used in Web Apps

The ability to locate any given UI object in the tree model is invaluable. However, sometimes test automation engineers fail to recognize the true beauty of XPath, that is, its impressive set of built-in functions. This way, test engineers can make their automated test solutions much more robust and even smarter.

This said, XPath supports functions for working with:

- Strings[3]: e.g. ***start_with, contains,*** and many more.
- Direction[4]: e.g. ***following, preceding,*** etc.
- Standard conditional operators[5], such as ***and, or,*** and others.

XPath and Automation

As described earlier, XPath has been used by automation frameworks such as Selenium and Appium as a means to identify the UI objects of web apps (DOM), native mobile app (Object trees), and hybrid mobile apps (objects tree containing DOM). DOM stands for Document Object Model and every web page has its own instance of it created by web browsers when pages are loaded.

XPATH WARS — WELL DEFINED VS. POORLY DEFINED EXPRESSIONS

Let us start with a practical definition of well and poorly defined XPath expressions by using the web and mobile apps for Bank of America.

Figure 69 shows the login area of the website bankofamerica.com.

3 String functions — http://www.w3schools.com/xml/xpath_operators.asp

4 Direction functions — http://www.w3schools.com/xml/xpath_operators.asp

5 Xpath operators — http://www.w3schools.com/xml/xpath_operators.asp

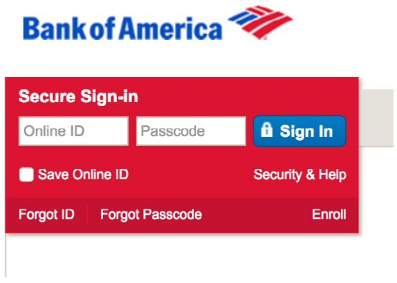
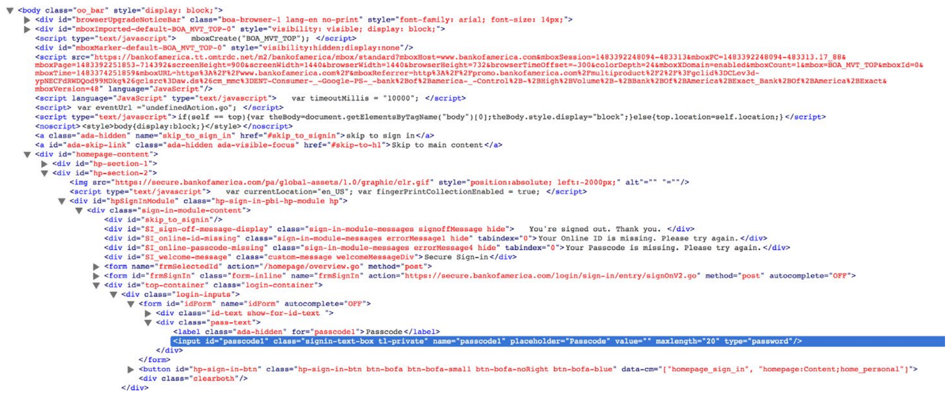


Figure 69: The Sign-in Area of the Bank of America Website

An excerpt from the corresponding DOM can be found in **Figure 70.** Note the marked password field.



```
▼ <body class="oo_bar" style="display: block;">
  ▶ <div id="browserUpgradeNoticeBar" class="boa-browser-1 lang-en no-print" style="font-family: arial; font-size: 14px;">
  ▶ <div id="mboxImported-default-BOA_MVT_TOP-0" style="visibility: visible; display: block;">
    <script type="text/javascript">   mboxCreate("BOA_MVT_TOP"); </script>
    <div id="mboxMarker-default-BOA_MVT_TOP-0" style="visibility:hidden;display:none"/>
    <script src="https://bankofamerica.tt.omtrdc.net/m2/bankofamerica/mbox/standard?mboxHost=www.bankofamerica.com&mboxSession=1483392248094-483313&mboxPC=1483392248094-483313.17_88&
    mboxPage=1483392251853-714392&screenHeight=900&screenWidth=1440&browserWidth=1440&browserHeight=732&browserTimeOffset=-300&colorDepth=24&mboxXDomain=enabled&mboxCount=1&mbox=BOA_MVT_TOP&mboxId=0&
    mboxTime=1483374251859&mboxURL=https%3A%2F%2Fwww.bankofamerica.com%2F&mboxReferrer=http%3A%2F%2Fpromo.bankofamerica.com%2Fmultiproduct%2F2%2F%3Fgclid%3DCLev3d-
    ypNECFdRWDQod99MDkg%26gclsrc%3Daw.ds%26cm_mmc%3DENT-Consumer-_-Google-PS-_-bank%2Bof%2Bamerica-_-Control%2B-%2BHigh%2BVolume%2B-%2BBank%2BOf%2BAmerica%2BExact_Bank%2BOf%2BAmerica%2BExact&
    mboxVersion=48" language="JavaScript"/>
    <script language="JavaScript" type="text/javascript">   var timeoutMillis = "10000"; </script>
    <script> var eventUrl ="undefinedAction.go"; </script>
    <script type="text/javascript">if(self == top){var theBody=document.getElementsByTagName("body")[0];theBody.style.display="block";}else{top.location=self.location;}</script>
    <noscript><style>body{display:block;}</style></noscript>
    <a class="ada-hidden" name="skip_to_sign_in" href="#skip_to_signin">skip to sign in</a>
    <a id="ada-skip-link" class="ada-hidden ada-visible-focus" href="#skip-to-h1">Skip to main content</a>
  ▼ <div id="homepage-content">
    ▶ <div id="hp-section-1">
    ▼ <div id="hp-section-2">
        <img src="https://secure.bankofamerica.com/pa/global-assets/1.0/graphic/clr.gif" style="position:absolute; left:-2000px;" alt"="" "=""/>
        <script type="text/javascript">   var currentLocation="en_US"; var fingerPrintCollectionEnabled = true; </script>
      ▼ <div id="hpSignInModule" class="hp-sign-in-pbi-hp-module hp">
        ▼ <div class="sign-in-module-content">
            <div id="skip_to_signin"/>
            <div id="SI_sign-off-message-display" class="sign-in-module-messages signoffMessage hide">   You're signed out. Thank you. </div>
            <div id="SI_online-id-missing" class="sign-in-module-messages errorMessage1 hide" tabindex="0">Your Online ID is missing. Please try again.</div>
            <div id="SI_online-passcode-missing" class="sign-in-module-messages errorMessage4 hide" tabindex="0">Your Passcode is missing. Please try again.</div>
            <div id="SI_welcome-message" class="custom-message welcomeMessageDiv">Secure Sign-in</div>
          ▶ <form name="frmSelectedId" action="/homepage/overview.go" method="post">
          ▶ <form id="frmSignIn" class="form-inline" name="frmSignIn" action="https://secure.bankofamerica.com/login/sign-in/entry/signOnV2.go" method="post" autocomplete="OFF">
          ▼ <div id="top-container" class="login-container">
            ▼ <div class="login-inputs">
              ▼ <form id="idForm" name="idForm" autocomplete="OFF">
                ▶ <div class="id-text show-for-id-text ">
                ▼ <div class="pass-text">
                    <label class="ada-hidden" for="passcode1">Passcode</label>
                    <input id="passcode1" class="signin-text-box tl-private" name="passcode1" placeholder="Passcode" value="" maxlength="20" type="password"/>
                  </div>
                </form>
              ▶ <button id="hp-sign-in-btn" class="hp-sign-in-btn btn-bofa btn-bofa-small btn-bofa-noRight btn-bofa-blue" data-cm="["homepage_sign_in", "homepage:Content;home_personal"]">
                <div class="clearboth"/>
              </div>
```

Figure 70: The Input Text Field for the Password in the DOM

We mentioned earlier that XPath expressions can be an extremely powerful weapon for identifying every single UI object on the screen in both web and mobile apps. However, XPath locators could be also used to shoot yourself in the leg.

That said, let us get back to our example above. If you go through the list in **Table 3,** you will find out that there are at least eight different XPath expressions matching the **passcode field.** Do you already get it? All of them will work and match our query. However, some of them would prove to be poor decisions during the evolution of the app. They would need to be modified too often and would bring inefficiencies in the maintenance of the automated test solution.

1	//*[@id='passcode1']
2	.//*[text()="Passcode"]
3	//input[@name="passcode1"]
4	//body/div[4]/div[2]/div[1]/div[1]/div[6]/div[1]/form/div[2]/input
5	//form[@id="idForm"]/div[2]/input
6	//form[@id="idForm"]/descendant::input[2]
7	(//input)[25]
8	(//input[@type!='hidden'])[4]

Table 3: Eight XPath Expressions Matching The Password Field in the Login Area

So, now the challenge would be to assess and evaluate each and every one of them in order to filter out the ones which would bring more damage than value. Let us digest the list and briefly discuss the options.

XPath expressions #1 and #3 are the ones mostly recommended to use. They both use a stable attribute ("ID" or "name") to identify the field. Its value will be the same across all platforms or versions of the app. In addition, **XPath expression #3** also specifies the type of the UI element (**"input"**), which increases the accuracy of the expression looking up one specific object.

XPath expression #2 may look like a good candidate, but the text value can be changed and the XPath will stop working as it used to. **Figure 71** illustrates how easy it is to break this one. Once a password is entered by an end user/automated test script, the value of the field would be already changed to many dots.

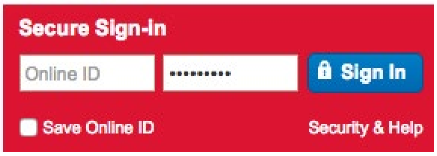


Figure 71: Normal App Interactions Breaking Poorly Authored XPath Locators

XPath expression #4 is based on absolute path which is very hard to maintain and very fragile. Any change in the page will break this XPath. Therefore, automation engineers should never use this type of XPath expression.

XPath expression #5 relies on identifying the *form* object which contains the creational fields. In this regard, it is better than expression #4, but it will not work as expected when the form is changed.

Xpath expression #6 is an interesting one. It looks up the parent object (in this case the "form") by its "ID" because it is defined and known. From there on, the built-in XPath function *"descendant"* is used to get access to the child elements filtered by their type ("input"). Of course, this is precisely the list of text fields representing the username and the passcode. XPath provides access by index to the elements in such lists. This way, the "passcode" text field is picked as the second element in the list. When you have no ID or good attribute to identify your object, try to make use of its "relatives" as stable anchors to locate your object in a more reliable way.

XPath expression #7 pinpoints the input field in the list of "input" objects on the whole page, while **XPath expression #8** does almost the same in a very cryptic manner. It looks for the fourth hidden "input" element available on the page. Both of them will not work if a new text field is added. In addition, #8 is also vulnerable to scrolling in the web browser.

Good	**Medium**	**Never use**
Expressions #1 and #3 stand out for their robustness. You should ideally use similar XPath expressions when automating test scenarios.	Expressions #5 and #6 identify the first parent with ID and as a next step select the needed "object". Expression #6 is more robust than expression #5.	Expression #2 will not work in any scenario when the value of the field is changed. Expression #4 will constantly give you hard times because it is based on a full path expression. Expression #7 will be broken many times when text fields are added or removed to the page. Expression #8 is a last resort.

Table 4: A Summary of the Analysis of XPath Expressions

ASSESSING GOOD AND POOR XPATH EXPRESSIONS IN THE SMART WAY

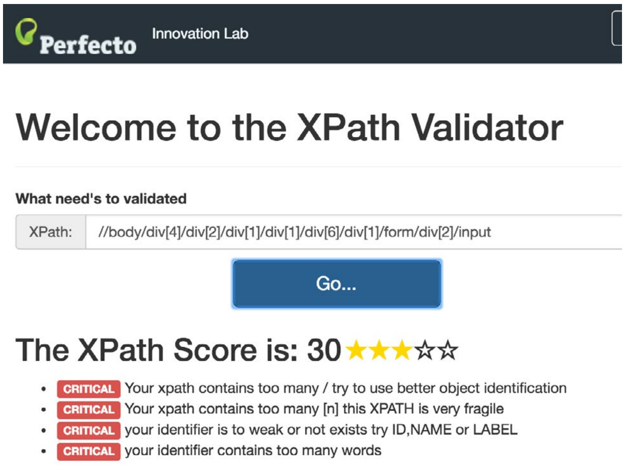
There are rules that can be used to differentiate between well- and poorly-defined XPath expressions. To help you test your XPath expressions, the Office of the CTO at Perfecto built a web tool that checks the quality of any XPath expressions passed through it. It conducts static analysis of the expressions so that the Object trees/DOM are not needed, i.e. it pretty much does what we have covered in our comparison of XPath expressions above.

The tool takes into account certain rules and heuristics in order to analyze XPath expressions and assess their quality. You can find some of them below.

- Does the XPath expression contain attributed definitions and, if yes, which ones?
- What is the length of the expression and how many indexes [n] does it contain? XPath expressions with large number of indexes are weak and can be easily broken across versions and platforms.
- How many tree levels ("/") does an expression contain? Too many levels may indicate "full-path" type of expression.
- How strong is the selected identifier (ID)? For example, in the case of mobile apps, some automation tools generate automatic IDs and name them after numbers that get changed with every build run. This makes certain XPath expressions relying on specific values of identifiers very weak.
- The quality of the XPath expressions is decreased when many words or too many XPath operators are used to get to an object identifier. Complex XPath locators could become troublesome across different platforms and lead to different (and often) unexpected results.
- The number of special characters has an effect on the quality of XPath expressions. In many cases, an excessive number of such symbols indicates that the identifier is generated automatically. Of course, this also means that the corresponding value will most likely be changed with the very next build. For example, the following xpath *(//span[@id="ext-element-140"])[1]* contains a couple of minus (-) symbols which should make automation engineers think twice before using it.

The tool analyzes XPath expressions and generates a quality score. It also highlights any detected issues. This tool is built based on knowledge collected from customers' test automation results. It is the result of analyzing the success rate of thousands of different XPath expressions over time in the context of different versions, platforms, and executions.

Let us quickly illustrate the value of the tool by analysing one of the XPath expressions that we used for the sake of understanding well designed and poorly designed expressions. The result has been displayed in **Figure 72.**



Welcome to the XPath Validator

What need's to validated

XPath: //body/div[4]/div[2]/div[1]/div[1]/div[6]/div[1]/form/div[2]/input

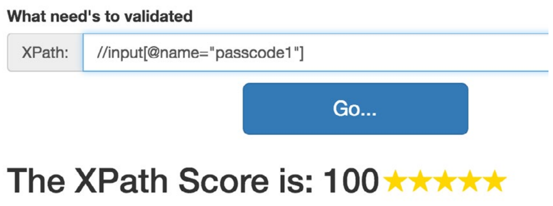
The XPath Score is: 30 ★★★☆☆

- CRITICAL Your xpath contains too many / try to use better object identification
- CRITICAL Your xpath contains too many [n] this XPATH is very fragile
- CRITICAL your identifier is to weak or not exists try ID,NAME or LABEL
- CRITICAL your identifier contains too many words

Figure 72: Quality Analysis of a Poorly Written XPath Expression

As you can see, there are four critical issues discovered. Each of them is described and the general score of the expression is by no means impressive.

If we do the same and pass one of the expressions that we assessed as a "good" one, the tool confirms our understanding of XPath, as evident in **Figure 73.**



What need's to validated

XPath: //input[@name="passcode1"]

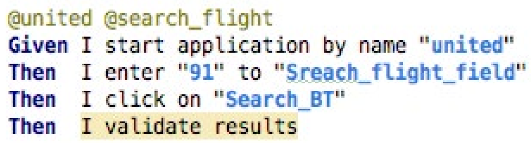
The XPath Score is: 100 ★★★★★

Figure 73: Quality Analysis of a Well Written XPath Expression

XPATH FOR MOBILE TEST AUTOMATION

The XPath expressions used for the sake of automating tests for mobile apps are a bit different from the ones used to look up objects in the DOM in web browsers. This certainly affects the way automation scripts are implemented.

Let us demonstrate these differences through a practical example, so that they may be more easily understood. This time, we will give the mobile app of United a spin. We will search for a flight. There is a ready-made, behavior-driven test illustrated in **Figure 74.**



```
@united @search_flight
Given I start application by name "united"
Then  I enter "91" to "Sreach_flight_field"
Then  I click on "Search_BT"
Then  I validate results
```

Figure 74: Behavior-driven Scenario for Searching a Flight in United

Right now, you can safely disregard the validation step. We will get back to this topic later in this chapter. Other than that, the flow is the same for both iOS and Android devices. Even the UI of the corresponding apps looks the same, as also confirmed in **Figure 75.** However, the two object trees are different and consequently the XPath locators used to identify UI objects will differ.

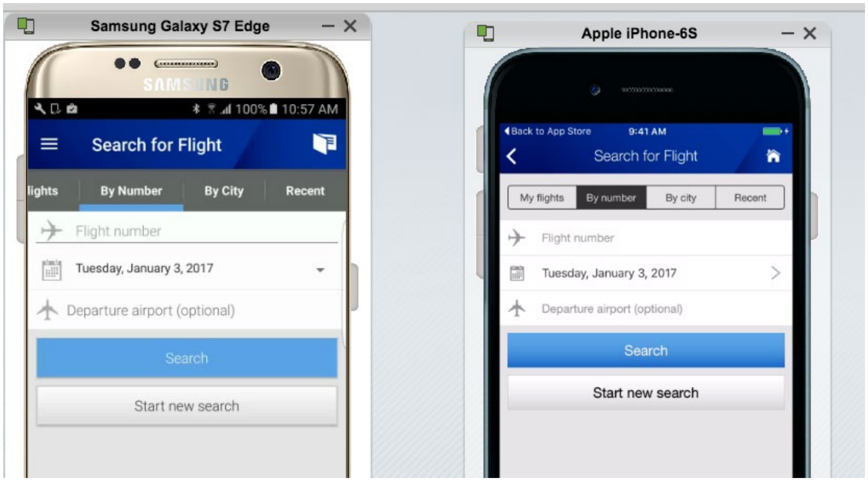


Figure 75: Unified Look-and-feel of the App of United for Both Android and iOS

Objects in Android are identified by ID, while in iOS they are not. The iOS objects contain few attributes.

The difficult part for automation engineers is to write and maintain one test script (and with the same flow) with different locators used for object identification across platforms.

There are two ways to address such a challenge:

- Build two external object repository files (one for iOS and one for Android) and provide them as parameters to the test script.
- Find common XPath expressions for both iOS and Android by using advanced functions like "contains" and **"starts-with"**. You could also make use of logical operators (*"AND", "OR"*) to enhance the lookup logic in terms of platform adaptability.

Keep in mind that the differences are substantial, as depicted in **Table 5.**

Object	**Android**	**IOS**
search_field	//*[@resource-id="com.united.mobile.android:id/flightNumber"]	//*[@value="Flight number"]
Search_BT	//*[@resource-id="com.united.mobile.android:id/searchButton"]	//*[@label="Search"]

Table 5: XPath Locators for Android and iOS — Comparison

That said, it will pay off to list the major differences between the models of the object trees in iOS and Android. The implications to the respective XPath expressions will become clear as well.

- Android objects contain a unique identifier ("ID"), while the corresponding UI elements in iOS do not.
- The names of the types of UI objects have absolutely nothing in common. Take, as an example, text fields. In iOS they are instances of the *"UIATextField"* class, while in Android they would be instances of *"android.widget.EditText".*
- Each type of UI object has a different set of attributes that can be queried.

- The hierarchy of composing UI objects, i.e. the object trees, have different structures.

In order to explore how both approaches work for United's app, let us try to come up with an XPath locator that is able to look up objects on both platforms. It is much better to work a bit more on a smarter query than parameterizing the whole scenario, isn't it?

At first, we need to figure out how to look up the "Search" button. The only common attribute between the iOS and Android apps turns out to be the label/text of the corresponding buttons. The label cannot be changed by end users as it is static for the corresponding resources. Thus, the only expression which works has been illustrated in **Figure 76.**

```
//*[@text="Search" or @label="Search"]
```

Figure 76: The XPath Locator for Looking Up the "Search" Button for Android and iOS

There are at least a couple of limitations relevant to this locator. If the app supports different languages, the expression will not work in languages different from English. The query is too generic and it basically describes any object labeled "Search". What would happen if a text field with this placeholder (i.e. search bar) would be introduced? That is right, there is no guarantee whatsoever that the button would be returned and not the text field.

If we quickly do the same exercise in order to identify the field where the flight number needs to be entered, we get the results shared in **Figure 77.**

```
//*[@value="Flight number" or @text="Flight number"]
```

Figure 77: XPath Query Looking Up the Field for Flight Numbers in the App of United

The aforementioned issues are applicable to this expression as well. Even worse, the value of this field is modifiable by end users, so when they enter a flight number, our expression will be unable to locate anything.

These expressions look good, but they are not stable and can be quite restrictive. In this particular case, we cannot not find good XPath locators which work flawlessly on both platforms. Therefore, we need to figure out different expressions that will bring better results when run against the corresponding platform-specific objects tree.

Let us start with Android. The first rule in UI automation is: if there is an ID, use it. The resulting expressions are listed below.

Flight number:

//[@resource-id="com.united.mobile.android:id/flightNumber"]*

Search button:

//[@resource-id="com.united.mobile.android:id/searchButton"]*

When it comes to iOS, the solution will be a bit more complicated. As you probably remember, there is no notion of strictly defined object IDs in iOS. There is a convention of using the accessibility label of UI elements as an object ID, but this is totally optional. Some development organizations do not become aware of it until it is too late. That said, the resulting expressions for iOS could be found below.

Flight number:

(//UIATextField)[1]

The concrete screen of the United app contains only one text field and the only attribute that could be used for automation purposes is its value. However, we have already discussed the downsides. This attribute turned out to be useless in terms of object identification. Therefore, it would be safer to fall back to the index of the text field in the corresponding list in the objects tree.

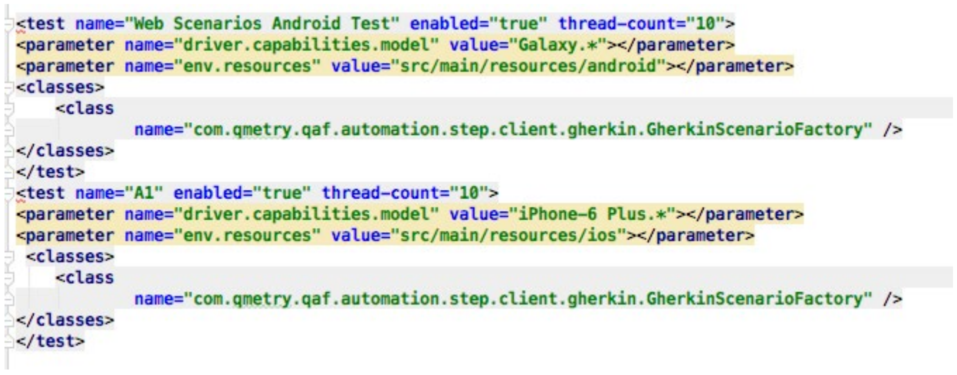
Search button:

//UIAButton[@label="Search"]

This is a good example of having the chance to benefit from the accessibility label of the field which was pre-set during the development of the app. It is stable and cannot be changed by end users. The type of the UI element is also clear, which makes it impossible for the underlying XPath implementation to look up and pick an incorrect element in the hierarchy.

In this example, we could not find a good common base for aligning the object identification for our automated solution on iOS and Android. The best implementation will rely on different expressions for object identification. They could be injected as a parameter in the script describing the flow of the scenario. There is a good chance that your automation framework already supports this automation use case. If not, you could implement it on your own.

For example, the open-source framework Quantum supports parameterization of object identifiers, as illustrated in **Figure 78.** In the testNG example, you can see how to pass the parameter **env.resources,** which points to different objects.



```
<test name="Web Scenarios Android Test" enabled="true" thread-count="10">
<parameter name="driver.capabilities.model" value="Galaxy.*"></parameter>
<parameter name="env.resources" value="src/main/resources/android"></parameter>
<classes>
    <class
            name="com.qmetry.qaf.automation.step.client.gherkin.GherkinScenarioFactory" />
</classes>
</test>
<test name="A1" enabled="true" thread-count="10">
<parameter name="driver.capabilities.model" value="iPhone-6 Plus.*"></parameter>
<parameter name="env.resources" value="src/main/resources/ios"></parameter>
 <classes>
    <class
            name="com.qmetry.qaf.automation.step.client.gherkin.GherkinScenarioFactory" />
</classes>
</test>
```

Figure 78: Quantum and TestNG Descriptor Featuring Parameterization of Object Identifiers

Note that the parameter named “env.resources” is defined twice in the scenario definition descriptor. This way, different XPath expressions could be injected and used in a test script, while the automated user flow would stay the same.

XPATH FOR VALIDATION PURPOSES

As part of a test script, we would typically like to validate certain requirements by analyzing and asserting the intermediate results of different interactions. Unlike flow objects, validation objects are more complicated to define, because:

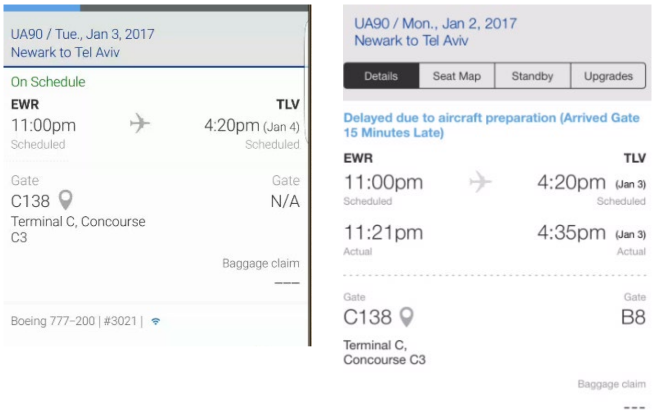
- User interactions may have more than one valid and acceptable result.
- Results can vary between different executions.

- Most of the time the elements do not have an ID defined.

If we continue with our flight search scenario, after entering the flight number and pressing the “Search” button, the script will need to validate the results. However, most of the results can change over time and different executions. Flights tend to be delayed all the time, so:

- Flight status may be “On Schedule” or “Delayed”.
- Time, dates, and gates may be different during different executions.

This is also depicted in **Figure 79.**


UA90 / Tue., Jan 3, 2017
Newark to Tel Aviv

On Schedule

EWR 11:00pm Scheduled — TLV 4:20pm (Jan 4) Scheduled

Gate C138 Terminal C, Concourse C3 — Gate N/A

Baggage claim ---

Boeing 777-200 | #3021 |

UA90 / Mon., Jan 2, 2017
Newark to Tel Aviv

Delayed due to aircraft preparation (Arrived Gate 15 Minutes Late)

EWR 11:00pm Scheduled — TLV 4:20pm (Jan 3) Scheduled

11:21pm Actual — 4:35pm (Jan 3) Actual

Gate C138 Terminal C, Concourse C3 — Gate B8

Baggage claim ---

Figure 79: Different and Dynamic Results Between Different Scenario Executions

There are means for automation engineers to introspect the object trees of apps while running on different platforms. Using basic tooling, one can reach the right conclusions about how a certain UI object could be looked up. You can pinpoint the flight status field during such introspection, as also illustrated in **Figure 80.**

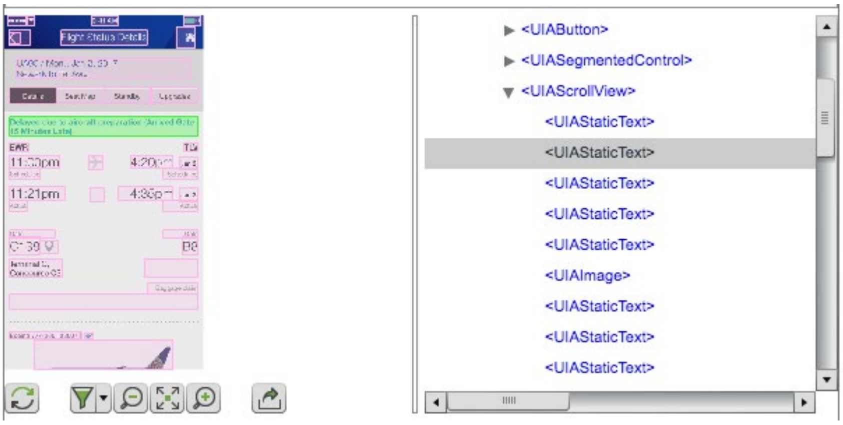


Figure 80: Introspecting the Objects Tree of the iOS Version of the United App

Thus, one will learn that the status field is the second instance of *"UIATextField"* within a concrete "UIAScrollView".

There are a few ways to validate the results after interacting with these fields. They have been explained and discussed in more detail below.

Option #1

Identify the element with the expected string or strings. If it is not found, catch the exception and raise an error, as also done in the code snippet in **Figure 81.**

```
@Test
public void checkFlightsStatus()
{
    String xpath = "//UIAStaticText[contains(@value,\'Delayed\') or @value=\'On Time\']";
    try {
        getDriver().findElement(By.xpath(xpath));

    } catch (NoSuchElementException e) {
      // Raise an error
     }
}
```

Figure 81: Raising an Exception When an Element Is Not Found

Option #2

Get all the elements by calling the method "findElements" of the Selenium/ Appium driver instance. If iterated upon and checked, the results can be verified and an error can be raised when an unexpected value is found. Such an approach has been illustrated in **Figure 82.**

```
@Test
public void checkFlightStatusLoop()
{
     String xpath = "//UIAStaticText";
     List<WebElement> elements = getDriver().findEleme
     nets(By.xpath(xpath));
     Boolean found = false;
     for (WebElement element : elements) {
           if (element.getText().equals("Delayed")) ||
                 element.getText().equals("On Time")
           {
                 found = true;
           }
           if (!found) {
                 // raise an error
           }
}
}
```

Figure 82: Iterating over Text Fields on Screen to Validate Flight Status

Option #3

A third option would be to use automation tools supporting means of visual validation. In this case, you will not need XPath expressions at all.

If it is not possible to use a visual validation tool, Option #1, described above, is superior, because:

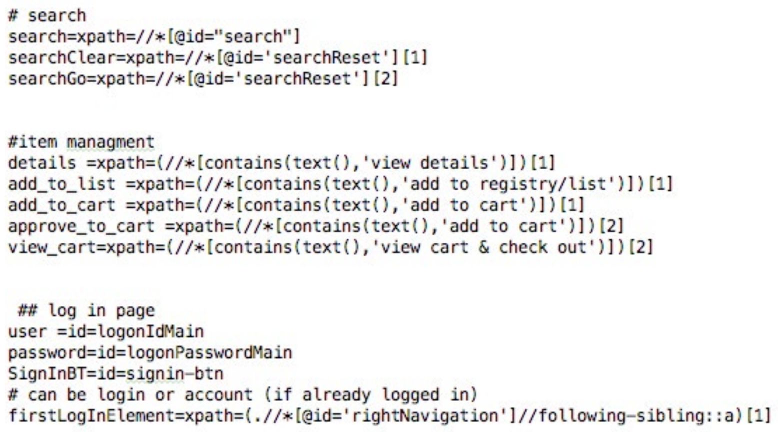
- The validation done by the XPath expression and the Java code is very simple and easy to understand.
- It is easier to maintain, since only the XPath locator would need to be modified, in cases of UI changes.

- The script queries the relevant field and not all the elements on the screen. This is how execution time is reduced, while the accuracy of the interaction results is increased.

SUMMARY

To wrap up and summarize this chapter, we have prepared a list of industry-proven good practices that you can find below.

- Use an ID whenever you can.
- Build processes inspiring the collaboration between developers and automation engineers at a very early stage during the creation of scripts.
- If your objects do not contain IDs, but the objects' parents or siblings have such, use them as anchors in conjunction with built-in XPath functions, such as:
  - "following-sibling"
  - "parent"
  - "descendant"
- Use automation frameworks that support external object repositories. Add and maintain all of your XPath locators within a set of external resource files (refer to **Figure 83** as an example) that can be injected within test scenarios depending on the context/ platform.



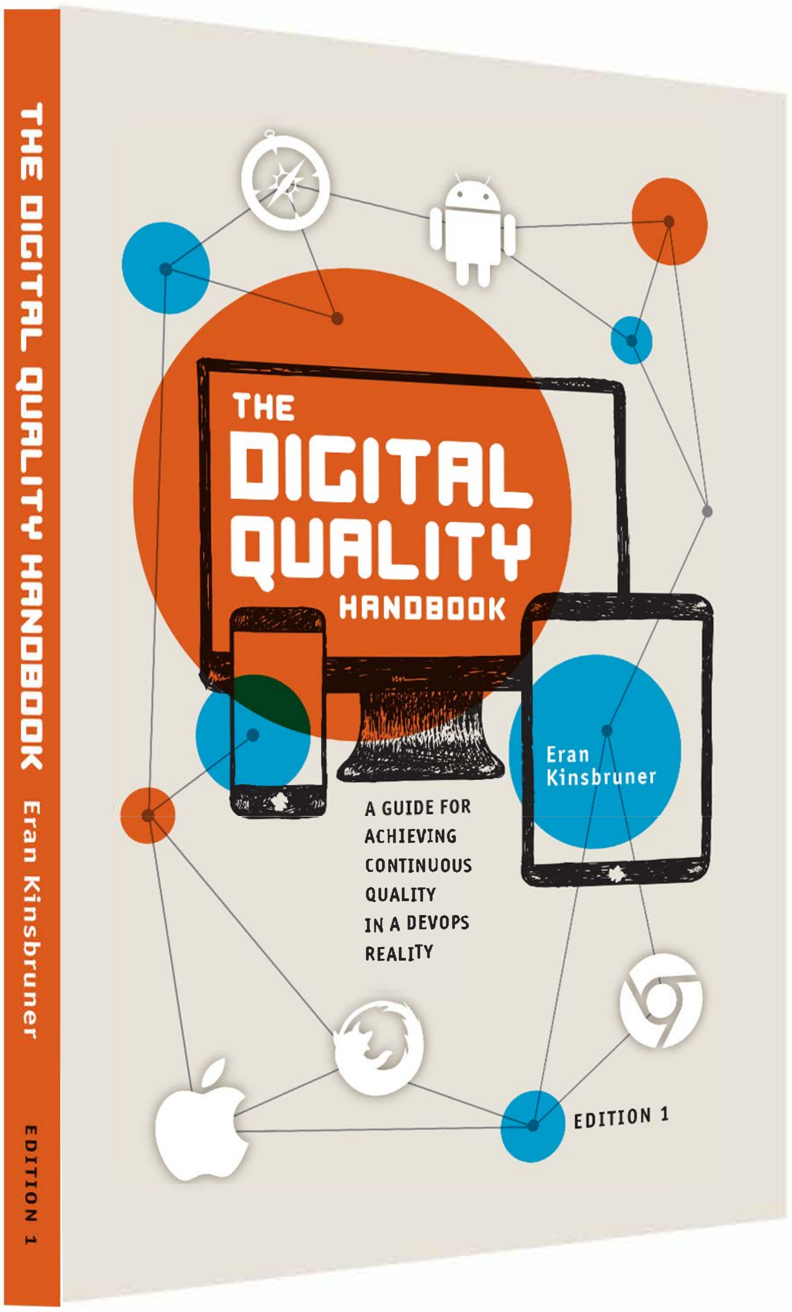
```
# search
search=xpath=//*[@id="search"]
searchClear=xpath=//*[@id='searchReset'][1]
searchGo=xpath=//*[@id='searchReset'][2]


#item managment
details =xpath=(//*[contains(text(),'view details')])[1]
add_to_list =xpath=(//*[contains(text(),'add to registry/list')])[1]
add_to_cart =xpath=(//*[contains(text(),'add to cart')])[1]
approve_to_cart =xpath=(//*[contains(text(),'add to cart')])[2]
view_cart=xpath=(//*[contains(text(),'view cart & check out')])[2]


 ## log in page
user =id=logonIdMain
password=id=logonPasswordMain
SignInBT=id=signin-btn
# can be login or account (if already logged in)
firstLogInElement=xpath=(.//*[@id='rightNavigation']//following-sibling::a)[1]
```

Figure 83: An Example Object Repository

- For **mobile** app/web, add the object repository as **parameter** to the script to execute one test/flow on all platform (iOS, Android, web).
- Try to add the **validations** to the XPath expression, build XPATH with the expected results, and look for the element as part of the script.
- For mobile apps, try your XPath expressions on all platforms. For responsive web, try it on three different screen sizes (desktop, tablet, and phone) before you commit your tests in the source control system.
- Try to limit expressions to a depth of two levels. For example, //o1/o2/o3 is not a good XPath.
- If an XPath locator contains more than one index id [n] and n > 1, the XPath is not strong enough and can change between different versions of the app under test or platforms.

If your XPath expressions still look like this, please re-read this chapter again:

```
//body/div[4]/div[2]/div[1]/div[1]/div[6]/div[1]/form/
div[2]/input
```


THE DIGITAL QUALITY HANDBOOK
A GUIDE FOR ACHIEVING CONTINUOUS QUALITY IN A DEVOPS REALITY
Eran Kinsbruner
EDITION 1
THE DIGITAL QUALITY HANDBOOK Eran Kinsbruner EDITION 1