[Print](javascript:window.print())

**Course Transcript**

Advanced Features of Selenium Automated Web Testing

**JavaScript, Ajax and Cookies**

[1. Course Introduction](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t2)

[2. Using JavaScript](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t6)

[3. Ajax Calls](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t10)

[4. Testing Visibility](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t14)

[5. Cookies](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t18)

**Extending and Managing Tests**

[1. assertEval](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t23)

[2. Listening for Events](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t27)

[3. Extracting Code](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t31)

[4. Multiple Browsers](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t35)

[5. Selenium Grid](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t39)

**Planning and Executing Optimized Tests**

[1. Test Planning](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t44)

[2. Testable Site Design](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t48)

[3. Test Suite Design Tips](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t52)

[4. Extended Example](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t56)

**Exercise: Work with an Extended Test Case**

[1. Practice: Implement and Manage an Extended Test Case](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#t61)

Course Introduction

Learning Objective

*After completing this topic, you should be able to*

* *start the course*

**1. Introduction to the course**

As web sites are getting more complex in their capabilities and implementations, we must also have our testing be smart and capable. I'm Tony Lowe, a Java architect with over 15 years of experience building and teaching IT solutions. Selenium provides us the capabilities to test for the most basic of pages up to pages with complex Ajax and JavaScript features. It helps us create smart test cases and to implement test plans. This course will help you with some of the advanced features as well as preparing how to use Selenium to design and improve your web site testing.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Using JavaScript

Learning Objective

*After completing this topic, you should be able to*

* *tweak Selenium tests with JavaScript*

**1. Using JavaScript for testing**

The Selenium IDE gives us basic controls but no real dynamic language in which to create complex test cases. Selenium does however provide us the ability to execute JavaScript as part of the test solution; to gather data, validate behavior, or just tweak the site as needed in order to get the test state we want. So on this page we are looking at here, we have a field inside of here. Let's just say for whatever reason, we need to tweak this field as part of our test solution involved in changing the value. So to change that value here, we can come inside of here and we could actually change that using a straight up Selenium command. So we can say runScript. And so as part of the runScript inside of here, we can add in any dynamic JavaScript we'd like in order to be able to change its behavior. So in this case, what I want to do is add some JavaScript to go to the document; the Document Object Model here, and do a getElementById. The name of this field right here, the name of this input box this get, is ('hiddenToStart'). And I want to change the value of that, you can see the .value inside of there to be equal to the value changed, all right. So I can go through and execute the script. Now, let me just double-click this guy and you can see it runs the script and look, the value is changed. I can put it back to some junk value here, run the script again and you can see, it executes that script and it changes the value to be changed.   
*In a browser, a locally stored web page is open. It contains the title Visibility, a text box set to Show Me, and two buttons – Show Hidden and Hide.  
  
In the Selenium IDE, the Base URL is set to the URL of the Visibility page. The Test Case pane contains the entry Untitled and the Test tabbed page is open. It contains a table with Command, Target, and Value columns, as well as Command, Target, and Value text boxes. The Target text box is associated with Select and Find buttons.  
  
The presenter enters runScript in the Command text box. In the table, an entry is added for the runScript command. A pane at the bottom of the IDE contains Log, Reference, UI-Element, and Rollup tabs. The Reference tabbed page lists information about the runScript command, such as the arguments it supports.  
  
In the Target text box, the presenter types  
  
document.getElementById('hiddenToStart').value="changed";  
  
This value is automatically added to the entry for the runScript command in the table.  
  
The presenter double-clicks the table entry for the runScript command. He then returns to the web page. The value in the Show Me text box is now the word "changed."  
  
The presenter types an arbitrary string of letters in the Show Me text box – jlkfdasjfkl. In the Selenium IDE, he double-clicks the table entry for the runScript command again and the value in the Show Me text box changes back to the word "changed."*   
  
So this opens up a whole world of possibilities. Now, anything I can do in JavaScript I can also do for my test script. I can go through and change a value. I can make something visible or invisible. Any of these attributes inside of here, I can go through and I could change as far as I want it to go. Now, that is not even limited to just simply using little commands I come up with here. I can actually interact with other JavaScript elements on the page. So if we go over here and look at the page source of this guy, the page source shows there's a callout here called remoteCall. And this actually does pretty much what we just saw right there: it goes and changes that hidden value to be changed. So if I – again, we'll go off and change this to be some junk value inside of here – I'm going to go back to my call here and I am going to add another runScript just to show you – runScript these guys side by side. And now I can make a call to remoteCall. And this is calling this existing JavaScript formula. So if I execute this guy, you see it's changed it in here to be changed.   
*Next the presenter opens the source code for the web page. He points out the following segment of the code:  
  
function remoteCall()  
{  
   document.getElementById('hiddenToStart').value="changed";  
}  
  
The presenter returns to the web page and types an arbitrary string of letters in the Show Me text box. In the Selenium IDE, he then selects a new row below the existing entry in the table. The presenter then types runScript in the Command text box and types remoteCall() in the Target text box.  
  
He again points out the source code  
  
function remoteCall()  
{  
   document.getElementById('hiddenToStart').value="changed";  
}  
  
In the IDE, the presenter double-clicks the new entry in the table. He returns to the web site to show that the value in the Show Me text box has changed from the arbitrary string of letters to the word "changed."*   
  
So we don't really have the ability to control everything within Selenium IDE. But what we can do is tweak whatever we need to do past that using JavaScript. It's not necessarily as powerful as using maybe a Java, or C, or Perl, or Python implementation of Selenium. But it does allow us to take the IDE and add some behavior to it, some dynamic behavior. And not only go through and change the little elements; we can pull back elements that might be complex to pull back; or change the behavior of what's going on within the flow of the test site to create error states, or whatever we would need to do in order to control our tests. So just be aware of this capability to be able to run scripts within the IDE and use that to great effect to create your testing.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Ajax Calls

Learning Objective

*After completing this topic, you should be able to*

* *manage Ajax calls in Selenium tests*

**1. Managing Ajax calls**

When a web site integrates Ajax update features we add a wrinkle to our testing. Most of the steps in a test assume that there's data content and that can keep running immediately after the previous step. In the Ajax world though, I often have to wait for an update from the server to get the desired results. Thankfully Selenium accounts for this as part of our test design options. So in this page right here I'm going to record a test. And it's going to record just fine, it's going to be real simple to record the test. I can go through here and I can put in Hello Server inside of here and I hit the **Update** button. So, it's going to go off, it's going to send that to the reserve server. In this case it's actually just a JavaScript pause in between, and I can validate this comes back. Okay, so I can assert here the text says Hello Server. Right, very simple test, I open the page, type something in, click a button and assert the results. So, if I go run this test again, it's going to fail miserably. You see my test has failed, assert results. Why? Because again it assumed this text was already there. I didn't tell it to wait anything, I did a click. Now I could do clickAndWait but that's actually not the intent of this functionality. So, I do a clickAndWait and it's going to sit there and wait, and it's going to wait, it's going to wait, it's still waiting. It's actually going to time out here because it's waiting for the page to refresh.   
*A locally stored web page is open. It contains the heading Wait for Response, a text box, and two buttons – Update and Update Fancy.  
  
The Selenium IDE is also open. The Base URL text box is set to http://localhost:8080, the Test Case pane contains the entry Untitled, and the Table tabbed page is open. The Table tabbed page contains a table with Command, Target, and Value columns, as well as Command, Target, and Value text boxes. The Target text box is associated with Select and Find buttons.  
  
In the bottom pane, the Log tabbed page is open. It lists various log entries categorized as [info], as well as an entry that reads  
  
[error] Element id=toolbox-close not found  
  
On the Wait for Response web page, the presenter types Hello Server in the text box and clicks the Update button. The output "Results from site Hello Server" then displays below the text box.  
  
The presenter right-clicks the link to open a shortcut menu and selects the following option:  
  
assertText id=results Results from site Hello Server  
  
In the Selenium IDE, the Table tabbed page now contains four entries. The first row lists the command open and the target website/content/wait.html. No value is specified. The second row lists the command type, the target id=entry, and the value Hello Server. The third row lists the command click and the target id=go. No value is specified. The fourth row lists the command assertText, the target id=results, and the value Results from site Hello Server.  
  
In the table, the presenter selects the row for the assertText command and clicks the Run button on the toolbar. In the IDE, the Log tabbed page then displays the following entry:  
  
[error] Actual value " did not match 'Results from site Hello Server'  
  
On the Table tabbed page, the row for the assertText command is highlighted in pink to indicate an error.  
  
Next the presenter selects the row for the command click. He changes the value in the Command text box from click to clickAndWait and, with the row still selected, clicks the Run button.  
  
After an interval, the Log tabbed page displays the entry  
  
[error] Timed out after 10000ms  
  
In the table, the row for the clickAndWait command is highlighted in pink.*   
  
We don't get exactly what's going on here. So you see this step fails inside of there. The test fails even though it actually successful. It actually asserted the text was there, but just waited for an indeterminate amount of time. So, I do definitely need to stay with click here because that's the behavior that's going on. I don't want to time out, I want it to take however long it's going to take to get back. So, the way I will need to do this instead is I need to add in a check. So, after this I'm going to insert a step and the step is going to be waitFor. Now, there's a lot of things I can wait for. I can wait for alerts, I can wait for things to show up on the page. In this case I'm going to wait for text to show up. And so inside of here, I'm looking for text to show an id of here, the id is results. And I know this because I can actually inspect this or I could do the **Select** inside of here. I'm going to just do a **Find**, just to show you, see that's indeed the one that's out there. I can do a **Select** the same way and say that way, the id=results. And the results need to match the exact text that's coming back here. So, I'm going to copy it here and I'm going to say waitForText. I'm waiting for this text to show up. If I know what the exact match of the results are, but I can do it this way. So, again, I can run my test, I'm going through, I'm clicking on the results, I'm waiting for the text to show up and then it continues on. So, I can do that really fast too. I have my speed turned down there a little bit, but it goes through. And whenever the results shows up, my test passes just fine.   
*The presenter changes the value in the Command text box from clickAndWait back to just click. He then right-clicks the row for the assertText command to open a shortcut menu and selects the option Insert New Command. A new, empty row is added above the selected row in the table.  
  
The presenter types WaitFor in the Command text box and a drop-down list suggests possible options for completing the command. The presenter selects waitForText to enter it in the text box. He then types id=results in the Target text box.  
  
The presenter returns to the web page titled Wait for Response and right-clicks the output Results from site Hello Server. In the shortcut menu that opens, one of the options is   
  
assertText id=results Results from site Hello Server  
  
The presenter returns to the Selenium IDE. He clicks the Find button alongside the Target text box and on the web page, the output Results from site Hello Server is highlighted in yellow.  
  
Next the presenter copies the text Results from site Hello Server from the web page into the Value text box in the IDE. He then clicks the Run button on the toolbar.  
  
The test case completes successfully and, to indicate this, is highlighted in green in the Test Case pane. The pane specifies the number of Runs as 1 and the number of Failures as 0.*   
  
Now, I can also wait for the opposite of that, instead of waiting for the exact text. Maybe I don't want the exact text to show up. What I can do is I can say waitForNotText inside of here. And when I say waitForNotText, what I'm saying is I'm waiting for the text to no longer whatever the value is right here. So, in this case, I can leave it blank and I can say waitForNotText inside of here. And if I run my test, it'll go through, it'll wait for the text to no longer be blank and the second it's no longer blank, it passes. That works pretty well. The last thing we can talk about here is what if instead of just saying blank this time, this is a really bad user interface. When I click something, I don't want it to be blank, it looks like I did nothing here, it's actually doing something. You know, but it doesn't look like I am doing anything. If I say, it looks like it's doing nothing, nothing, nothing and all of a sudden, it updates. What I would prefer to have happen here is that when I hit the update, give me some visual indication. Hey it's loading. So, that adds a little wrinkle inside of here. But I can change this to be waitForNotText and put the word Loading inside of here. I actually, let me hit the other button. And when I click the button, instead of say go, I want to goFancy, that's the name of the button inside of here. Let me just show you, I hit the **Find** here. I'm updating the **Fancy** button instead, I'm going to click that button instead. And so now if I run this test this way, I go click the **Fancy** button, it goes to Loading. And once the text is no longer says Loading, then I can go assert whatever that text comes up outside of there.   
*The presenter selects the row for the command waitForText in the table. He then changes the entry in the Command text box from waitForText to waitForNotText, and deletes the entry from the Value text box. He clicks the Run button and the test passes.  
  
On the web page, the presenter changes the entry in the text box from Hello Server to HelloServer2222. After an interval, the output Results from site Hello Server2222 displays below the text box.  
  
Next the presenter clicks the Update Fancy button. This time the message "Loading" displays before the result is returned.*   
  
So, again, there's a lot of options inside of here, the wait is – the waitForText, waitForNotText, wait for a bunch of different things that could show up inside of here. Let me just show you one more time, and we wait for the Selenium IDE, actually let's do it this way. Wait for a lot of different options here. waitForLinks to show up, I **waitForBodyText** to show up, I can wait for an attribute to show up, I wait for cookies to show up, a CSS count to show up of some element. There's all sorts of things that I can wait for, that's pre-programmed into my IDE. And that's going to allow me to handle most of these Ajax behaviors and be able to respond once the site goes away and comes back with some refresh. So, you can use this and test your Ajax and still not lose any functionality.   
*In the IDE, the presenter selects the row for the waitForNotText  command and types the word Loading in the Value text box. He then selects the row for the click command and changes the entry in the Target text box from id=go to id=goFancy. The presenter clicks the Find button and the Update Fancy button is highlighted in yellow on the web page.  
  
The presenter clicks the Run button in the IDE. On the web page, the message Loading displays, followed by the message Results from site Hello Server. The IDE shows that the test case has passed.  
  
The presenter types waitFor in the Command text box to demonstrate the options that are available. A drop-down list includes options such as waitForAllWindowIds, waitForAllWindowNames, waitForAllWindowTitles, waitForAttribute, waitForAttributeFromAIWindows, and waitForBodyText, waitForAllButtons, waitForCookie, waitForAllLinks, and waitForCssCount.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Testing Visibility

Learning Objective

*After completing this topic, you should be able to*

* *test visibility in the Selenium IDE*

**1. Visibility testing**

As we move into Ajax-based applications, our pages no longer are loaded all at once and remain static for the visitors. Each user interaction, or even without a user interaction, may cause updates on the page. This becomes challenging for testing as we can no longer assume a linear behavior and need to accommodate for time as part of our test. Let's look at how Selenium gives us the ability to wait for changes in the page as part of the test flow. So, in this little page we have getting up, we're kind of mimicking maybe an auto-complete functionality or a remote search, or something like that, where I'm hiding a button, that box is going away, and then I'm just in JavaScript making a comeback a couple of seconds later with some updated results. So, from this page, if we're going to do a test in this page, we have to be very careful how we construct it. So, we can start here on this page and we can say, okay let's validate here, verify here, the text it starts with. So, we open the page and we're going to verify the text to start with. We're just going to record the steps as we go along here. And then we'll hit this **Hide** button and then it'll run and it will come back, and okay so we did that, and let's validate here at the end, let's assert here that this text at the end indeed says what we think it is and in this case it should say - it didn't fill it in for me - so it should say I'm Back! We'll double-check this line right here. Ah nope, I didn't… here, let me do this real quick. Let me grab this line again, just to make sure.   
*In a browser, a locally stored web page is open. It contains the title Visibility, a text box set to Show Me, and two buttons – Show Hidden and Hide.  
  
In the Selenium IDE, the Base URL is set to http://localhost:8080. The Test Case pane contains the entry Untitled and the Test tabbed page is open. It contains a table with Command, Target, and Value columns, as well as Command, Target, and Value text boxes. The Target text box is associated with Select and Find buttons.  
  
On the web page, the presenter clicks the Hide button. This causes the text box to disappear briefly. When it displays again, it contains the message "I'm Back!"  
  
The presenter selects the page URL in the address bar and presses Enter. The text box again contains the text "Show me."  
  
On the web page, the presenter selects and right-clicks the text "Show me." In the shortcut menu that opens, he selects the option verifyValue id=hiddenToStart Show me.  
  
In the Selenium IDE, the table on the Table tabbed page now contains two rows. The first row lists the command open and the target /website/content/visible.... No value is listed. The second row lists the command is verifyValue, the target id=hiddenToStart, and the value Show me.  
  
The presenter clicks the Record button in the toolbar.  
  
On the web page, the presenter clicks the Hide button. The text box disappears briefly and then returns with the message "I'm Back!"  
  
A new row has been added to the table in the IDE. It specifies the command click and the target id=makeInvisible. No value is specified.  
  
On the web page, the presenter selects and right-clicks the message "I'm Back!" and then, in the shortcut menu, selects the option assertText id=hiddenToStart.  
  
In the IDE, a new row has been added to the table. It lists the assertText command and the target id=hiddenToStart. No value is specified.  
  
The presenter types I'm Back in the Value text box.*   
  
There you go. Let's do there, let's just do a verify again on this, let's just… yes, it's indeed right there. So there's our test. Again, the test looks straightforward, it looks very linear but if I run the test as it sits there, it fails. Why? Well, if we run it a little bit slower, we can see what happens inside of here. So we play this, here it is, that opens it up, it verifies the value, it clicks on it and then it verifies the value but it's not for another couple of seconds that that value comes back. Because the test is running so quickly, it's actually running right past. It's ignoring the fact that that box disappears. It doesn't know any better. The interesting thing inside of here is this. If I look at the source code inside of this a little bit, if I go through and let me do the page source, I can actually see inside of the JavaScript - I am going to just jump right to it - that what happens is when I hit the makeInvisible there, it changes the value to be Poof! and then it comes back a little bit later and says I'm Back!. And so the cool thing about our ability here is we can actually validate text even if it's not shown, so it's made invisible inside of here, it's not showing any more. And I can actually go through and I can insert a command inside of here and I could say verify the value. I'm going to copy this one down here. And at this point, let's verify it goes to be Poof! Okay, so in this sequence, after I click, it is going to indeed change. I'm going to run the test here and let's speed that up.   
*The presenter double-clicks the row for the command. It's highlighted in pink to indicate an error.  
  
The presenter returns to the web page, selects and right-clicks the text "I'm Back!", and selects the option verifyValue id=hiddenToStart I'm Back!  
  
In the IDE, the row for the assertText command has been replaced by a row that lists the verifyValue command, the target id=hiddenToStart, and the value I'm Back!  
  
The presenter then clicks the Run button. The test fails and the Log tabbed page lists the following entry:  
  
[error] Actual value 'Poof!' did not match 'I'm Back!'  
  
On the toolbar, the presenter drags the slider for determining the speed of testing to the right, towards the Slow end. He then clicks Run and you can see that the Hide button on the web site is clicked and the text box temporarily disappears before displaying with the message "I'm Back!" The test fails again.  
  
Next the presenter selects and right-clicks the message "I'm Back!" on the web page and, in the shortcut menu that opens, selects View Page Source.  
  
In the source code for the page, the presenter navigates to the code for the makeInvisible() function and points out the code  
  
hideMe.value = "Poof!";  
  
He then navigates to the code for the makeVisible() function and points out the code  
  
hideMe.value = "I'm Back!";  
  
The presenter returns to the Selenium IDE. He selects and right-clicks the last row in the table, which is for the verifyValue command, and then selects Insert New Command from the shortcut menu. A new row is added above the row for the verifyValue command.  
  
The presenter then copies the entries in the second row of the table, which lists the command verifyValue, the target id=hiddenToStart, and the value Show me, into the new row. He then changes the entry in the Value text box from Show me to Poof!  
  
The presenter clicks the Run button and drags the slider back towards Fast. The test fails again and the last row in the table, which lists the command verifyValue, the target id=hiddenToStart, and the value I'm Back, is highlighted in pink to indicate an error.*   
  
And it's fine. It's actually verifying the value does change from Show me to Poof! inside of there. But it still fails on the start. So, I have to insert one other command inside of here. Now, the command is quite simple; it's called waitForVisible. Alright, there's a lot of waitFor options inside of here. So, in this case, the command is waitForVisible. There are other waitFor options: I can waitForAlert, I can waitFor buttons or fields or I can waitFor attributes or other things inside of here. We're just going to waitForVisible here. So, when that guy becomes visible again. So, maybe it was invisible but when it becomes visible again. Now, which field am I talking about? Well, we can say **Select** the field; that one right there. Alright, so once this guy is visible, then it's going to continue on. That should be all I need to do at this point. Again, it will be able to validate this. So, if I run this test one more time, you can see it pauses at this line right here. So it runs everything real quick, it says it's Show me, it changes to Poof!   
*The presenter right-clicks the last row in the table and selects Insert New Command. A new, empty row is added above the last row. The presenter types waitForVisible in the Command text box.  
  
The presenter points out options that display in a drop-down list once waitFor has been typed. Examples are waitForAlert, waitForAllButtons, waitForAllFields, waitForAllWindowNames, and waitForAttribute.  
  
The presenter clicks the Select button alongside the Target text box and then selects the text box that contains the message I'm Back on the web page. The entry id=hiddenToStart is then automatically entered in the Target text box.  
  
The test case passes. The row for the waitForVisible command, which is currently the fourth command in the table, is highlighted in blue to indicate a pause. All the other rows are highlighted in green.*   
  
Then it goes through and waits for it to become visible and then it validates/verifies the value. You know I could even, if I wanted to, insert a command here to just say verifyNotVisible, at this point. And we can **Select** just to show, yes indeed that box does away, and it's visible right now but if run the test case, you can see it actually does verify, it's not visible. It waits for it to be visible, then it verifies the value in that box. And so now, using the waitFor command and some of the waitForVisible command, I can test my Ajax features and get back to the point of being able to see all my tests pass even though it might take some indeterminate amount of time for it to make the call and get a response back.   
*Next the presenter inserts a new row above the row for the waitForVisible command. He types verifyNotVisible in the Command text box and selects the text box on the web page to add the entry id=hiddenToStart in the Target text box.  
  
The presenter clicks Run and the test passes, as indicated by green highlighting.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Cookies

Learning Objective

*After completing this topic, you should be able to*

* *work with cookies in Selenium tests*

**1. Working with cookies in Selenium tests**

A key part of the technical design of some web sites is the use of cookies. Since our cookies are browser-based, we need to use Selenium to reach into the browser and help us determine that our site is seeing, and managing cookies, as we expect. Selenium IDE gives us a few quick commands to manage cookies. So in this test we're going to see the use of cookies as part of this web site in its use of menus. And so for this test we're going to start a recording here, and we're going to log into the site. Alright and so we log into the site and over here, on the side, we can see there's a number of these menu items. Now from page to page, the browser doesn't know whether or not you've clicked these before, and so we have two choices in the design. We can store on the server side or we can store on the client side and this design, the key part of it is it's using cookies to determine that these have been clicked on. So now that I'm logged in, I want to go through and I can do some checks on cookies. So inside of here I can add a new command: assertCookie. And so we're going to see if the cookie is present. So we can say CookieNotPresent in this case. So the cookie name is going to be based off of whatever the site name is and we know, in this design the cookie name is going to be based off of the name of the link,   
*In a browser, a locally stored web page is open. It contains Username and Password text boxes, and a Login button.  
  
In the Selenium IDE, the Test Case pane contains the entry Untitled. The Table tabbed page is open. It contains a table with Command, Target, and Value columns, as well as Command, Target, and Value text boxes. The Target text box is associated with Select and Find buttons.  
  
The presenter clicks the Record button in the Selenium IDE. On the web page, he then types admin in the Username text box, types a password in the Password text box, and clicks the Login button.  
  
On the web page, four expandable menu options become available. They are Your Account, Admin Functions, Site Content, and Interactive Content. The page displays the message "Choose a feature from the list to the right." It also now contains the heading "New and Uncompleted Notices" and, below that, the message "No Notices to Handle."  
  
The table in the IDE now contains entries in four rows. Row 1 lists the command selectWindow and the target name=mainWindow. Row 2 lists the command type, the target id=usernameInput, and the value admin. Row 3 lists the command type, the target id=passwordInput, and the value test. Row 4 lists the command clickAndWait and the target name=submit.  
  
The presenter adds a new row. He types assertCookieNotPresent in the Command text box.*   
  
and so Your Account over here. And so we can actually run this command right here and assert, okay yeah it's not present exactly as we expected. So I typed in my command here, CookieNotPresent, I gave it the target value – which is the name of the cookie – and I went ahead and double checked it, hey it's not present right now. So let me now click on the link here and I actually have to go back to my recording here, turn my recording back on, click on the link here and you can see this opens up. So now I can go back and check to assertCookiePresent afterwards, and the cookie should be Your Account inside of here. And we can validate that, and look here, the cookie is now indeed present. And we can do more than that – we can assert that the cookie is not only present or not present, we can assert it by name. Or we can just say the cookie, "Your Account" has the expected value we want to, in this case, of true. And so we can assert this one as well. Now, you can see in this case it's saying the Actual value true did not match 'Your Account'. So, there's a little bit of an issue inside of here, and we actually need to use ByName in this case,   
*The presenter types Your Account in the Target text box. The presenter clicks the Record button and, on the web page, clicks the Your Account option to expand it. This reveals a Change Password option.  
  
The presenter returns to the IDE and adds a new row to the table. He types assertCookiePresent in the Command text box and types Your Account in the Target text box. The presenter double-clicks the row to run it and the Log tabbed page displays the entry  
  
[info] Executing: |assertCookiePresent | Your Account | |  
  
In the IDE, the presenter enters a new row in the table. He types assertCookie in the Command text box, Your Account in the Target text box, and true in the Value text box. He then double-clicks the row and the Log tabbed page displays the entry  
  
[error] Actual value 'Your Account=true' did not match 'Your Account'  
  
The presenter changes the entry in the Command text box from assertCookie to assertCookieByName.*   
  
and then that guy is true. So you can see the little trick inside of there. So, we're just saying in one case you're saying, assert that the cookie is there, we say assertCookieByName, we can assert the value is inside of there. Now, if we expected the value here to be "On" or something like that, we could've put the word On inside of here. The fact that it's showing true is again a design decision. And in this case, you can see that the Actual value 'true' did not match 'On'. We need to know how the design of this site is working, this testing is very much a white box test, not a black box test. We're not just, you know looking at the interface alone, we need to know behind the scenes how this thing's designed – how this thing is planned. But at this point you can see, we have a really basic test and so we can actually go through here, and let's actually close out our test. We know we need to log out in order to be able to login inside of here. And let's try and run our test and see what happens. Oh, sorry I, we actually are, we did not start with the correct commands, so let me just go ahead and take this line out. It added a line in the recording that we didn't really mean to, because of the way it started up. But again, we'll try this again. Okay, so now it came through, it ran the tests and it – well it failed, why did it fail here? Well it says the CookieNotPresent, we had that check in here to make sure the cookie was not present. The reason that was the case is because we started this test within a browser and the cookie stays through the session.   
*The presenter double-clicks the row and the Log tabbed page displays the entry  
  
[info] Executing: |assertCookieByName | Your Account | true |  
  
The presenter changes the entry in the Value text box from true to On and double-clicks the row. The Log tabbed page then displays the entry  
  
[error] Actual value 'true' did not match 'On'  
  
The presenter changes the value back to true, double-clicks the row, and the Log tabbed page again displays the entry  
  
[info] Executing: |assertCookieByName | Your Account | true |  
  
The presenter switches to the web page and clicks the logout option. Once he has done this, the option changes to login.  
  
The presenter returns to the IDE and clicks the Run button, and the test fails. He then removes the first row, which lists the command selectWindow and the target name=mainWindow, from the table and clicks Run again.  
  
The test fails again. The first row that lists the command assertCookieNotPresent is highlighted in blue, indicating that this is where the test failed.*   
  
We have to be aware of the state of everything going along, we can't just trust. And that's why we have all these extra commands. And so the other little option I can show you inside of here – and we're not going to go and fix this test case a 100%, to make it a 100% working – the easy answer is here to start from a fresh browser each time, start with no session. But we can also manipulate cookies just to show you, if we needed, we can say we can store a cookie, we can store cookie by name, we can store if a cookie is present. We can go through and store these values as well within our logic, and so we can store a cookie and use logic on our side to determine what the state of it would be from there. So you can see if we say storeCookie this says Generated from getCookie() here, all cookies of the current page under test Return all of cookies under test. Or I can say storeCookieByName and then we can say hey, we want to go and return a cookie. And again the store inside of here, stores inside of our Selenium IDE, and then we can go and add in JavaScript or something else to determine what's supposed to be going on. So again if you have very simple cookies, they're there, they're not, you can check them, but if it's more complex, I need to go into storeCookie, so I can design test cases appropriately. So again Selenium IDE gives us the ability to look at cookies and then make our tests to go past just straight black box testing of user interface, into the white box testing and validate that this site is designed, and behaving as we expect.   
*The presenter adds a new row to the table and types the letters storeCo in the Command text box. A drop-down list contains options for completing the command, such as storeConfirmation, storeCookie, storeCookieByName, and storeCookiePresent.  
  
The presenter enters storeCookie in the Command text box. The Reference tabbed page then lists information about the command, as follows:  
  
storeCookie(variableName)  
Generated from getCookie()  
Returns:  
             all cookies of the current page under test  
Return all cookies of the current page under test.  
  
The presenter changes the entry in the Command text box to storeCookieByName. The Reference tabbed page provides the following information:  
  
storeCookieByName(name,variableName)  
Generated from getCookieByName(name)  
Arguments:  
name - the name of the cookie.  
Returns:  
             the value of the cookie  
Returns the value of the cookie with the specified name, or throws an error if the cookie is not present.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

assertEval

Learning Objective

*After completing this topic, you should be able to*

* *use deeper Selenium IDE testing with assertEval*

**1. Testing with assertEval**

Out of the box, Selenium IDE gives us many basic validation options that can even be added by simply selecting items on our page and right-clicking. At times though, our test evaluation must be more detailed in analyzing the changes on our pages. When that happens, we can take advantage of Selenium's ability to create variables in the IDE. And then evaluate the variables to determine the testing outcomes. So for this test that's already recorded, we're going to go walk through this. And I have it set to pretty slow, so I am going to start it walking through. We're adding a poll. And then we're not testing the ability to add a poll. We've tested that before. But we can see we're adding in two options, Yes and No. And we're going to save it but what we want to test is the ability to edit this poll. So as we go into edit this poll, we're going to keep working inside of here. We're going to add additional options. So we're adding one, two, three options inside of here. Let me pause this real quick. So we've added those options in. But initially up here, after we completed the first part of the poll, we went through and we used this variable inside of here called – this command inside of here called storeCssCount. And so what this storeCssCount does is it goes and does a CSS Selector in this case. And the CSS Selector is saying, "Give me all of the input options with the name of optionBeingEdited." And it's going to give us a count from there. And as you can see from here, if we actually look at the log,   
*In a browser, a locally stored web page is open. The web page includes the title Add a Poll and contains a Poll Name text box, Active Date and Inactive Date text boxes – each with a button for accessing a calendar, Question and Possible Answer text boxes, and two buttons – Save Poll and Cancel.  
  
In the Selenium ID, the Test Case pane lists a test named assertEval. The Table tabbed page contains a table with Command, Target, and Value columns, as well as Command, Target, and Value text boxes. The Target text box is associated with Select and Find buttons.  
  
On the Table tabbed page, the table lists the steps in the assertEval test case. For example, the first row lists the command open and the target /website/addPoll.vpd?dispatch. The second row lists the command type, the target name=name, and the value Edited Poll. The third row lists the command type, the target name=question, and the value Can I edit it successfully. Other rows list various commands, such as type, clickAndWait, click, storeCssCount, and echo commands.  
  
The presenter drags the slider for setting the speed of the test to slow and clicks the Run button on the toolbar. Each row in the table is highlighted as it executes and relevant changes display on the web page.  
  
As the test case runs, the question "Can I edit it successfully" is added to the Question text box, Yes and No radio buttons are added below the Possible Answer text box, and then additional radio buttons – Maybe, Sometimes, and Of Course – are added. Edit, Delete, Up, and Down buttons are added below the radio buttons, and three buttons are added below that – Save Poll, Duplicate Poll, and Cancel.  
  
The web page then changes. It contains Poll Name and Number of Responses columns. The Poll Name column contains Edited Poll and Unedited Poll radio buttons, each with associated Edit, Clear, and Delete buttons. The Number of Responses column lists a number for both edited and unedited polls.  
  
In the Selenium IDE, the presenter clicks the Pause button. In the table, he selects a row that lists the command storeCssCount, the target css=input[name=optionBeingEdited], and the value initialCount.  
  
The presenter then selects the next row, which lists the command echo and the target ${initialCount}.*   
  
it gave us the count right here. It said 2, we have 2 of these items right here. The initial count of these guys was 2. Now, I could put other text inside of here, I'm just printing out the count in this case. But I could say the initial count is 2 or whatever I want to but I just wanted to show you the value being printed out. And this being stored inside of this variable called initialCount right here. You can see the initialCount is the name of the variable. So now that we've added the additional items, we're going to go through it. And I'm just going to go through each page one at a time at this point. And so I click the button and it comes up here. So now I'm going to do another storeCssCount and in this case, it's going to go through and it's going to look underneath the rows. It's going to go through and it's going to pull all the rows that have a CSS of a table row of the class pollResultLine. And it's going to look for the td inside of there. It's going to look for each of those. So it's going to get the final count off for that. This is my CSS Selector inside of there. And the final count is going to be stored and the result you can see here is 10. Now, there aren't actually ten in there. It's actually counting all of the rows and all the cells. So 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 and that's why we get 10 inside of there. And so we can accommodate for that, it's okay, we don't have to have perfect numbers here. We could probably work very hard to get a perfect selector inside of there. But we know we're going to get double the number of columns inside of here because there are two columns instead of one before the row. And so that's fine. We'll just grab each of that.   
*On the Log tabbed page below the table, the presenter points out the entry [info] echo: 2.  
  
The presenter selects the previously selected row that lists the command storeCssCount again and points out that the value for the row is set to initialCount.  
  
Because of where the test case was paused, the web page currently contains Poll Name and Number of Responses columns. The Poll Name column contains Edited Poll and Unedited Poll radio buttons, each with associated Edit, Clear, and Delete buttons. The Number of Responses column lists a number for both edited and unedited polls.  
  
In the table in the Selenium IDE, the presenter double-clicks a row that lists the clickAndWait command and the target documentViewPollsForm.buttonPressed[6].  
  
The web page changes. It now contains the heading Delete Poll, a table titled Can I edit it successfully, and two buttons – Confirm Delete and Cancel. The table lists five possible responses – Yes, No, Sometimes, Maybe, and Of course – each followed by a zero in brackets.  
  
Next the presenter selects a row that lists the storeCssCount command, the target css=tr.pollResultLine > td, and the value finalCount.  
  
He then selects the next row, which lists the echo command and the target ${finalCount}.  
  
The presenter points out that the Log tabbed page includes the entry [info] echo: 10. On the web page, the presenter then points out that together, the five responses listed in the table plus the value listed for each response add up to 10 items.*   
  
And then we press the button inside of there and then we can go through and finish our evaluation. So we can use the storeEval to allow us to do some math inside of here. So we have the stored variables. And we're going to go through finalCount as a stored variable. That's the second one we grabbed inside of here. And for finalCount, we're going to divide it by two and that's where we get the 5. You know, the total number that was out there. And then we can subtract from that the value of the stored variable which was the initialCount. So the storedVars is all the items we have created and stored using the storeCssCount. Any other store command that's out there as an option to us. And then we can store that in what's called $(added Options). And so you can see here added option now is equal to 3. So 10 divided by 2 which is 5; minus 2 which is the initial count there and it gives me the answer 3. Okay, and finally then I can do an assertEval. And this is where I can go through and look at my variables out there and compare them and say okay, given all that data collected, what's my results. So in this case my storedVars, I'm going to go look for addedOptions, I'm going to say does that equal 3,   
*The presenter selects the next row in the table in the Selenium IDE. It lists the command clickAndWait and the target name=buttonPressed. The web page now lists the Poll Name and Number of Responses columns with Edited Poll and Unedited Poll radio buttons, and associated Edit, Clear, and Delete buttons.  
  
The presenter selects the next row in the table in the Selenium IDE. It lists the command storeEval, the target storedVars['finalCount']/2-storedVars['initialCount'], and the value addedOptions.  
  
The presenter selects the next row in table, which lists the command echo and the target ${addedOptions}. On the Log tabbed page, the presenter points out the entry [info] echo: 3.  
  
The final row in the test case lists the command assertEval, the target storedVars['addedOptions'], and the value 3.*   
  
because I am expecting it: that I added three options. So after I've done all that evaluation, you could see I stored some variables, I've done some clicking. I can say yes, I have added three options indeed inside of there. So it's a pretty simple test, but you can see how it can scale up to allow me to do a lot of different things. For instance I could be looking in the database. I could say when I go look at this specific user's number of transactions for the last month, there should be 27 of them, so I could do some variables and delete some items and move some items around and play with some formatting. And then compare it to the answer 27 that I'm expecting to find or whatever that number might be. So using the storeVars plus the assertEval allows me to go through and make whatever kind of collections I want to and do much deeper testing than just the simple are these two values equal, is this equal to hard code value. You can really get in there and know that my web site is working properly.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Listening for Events

Learning Objective

*After completing this topic, you should be able to*

* *listen for Selenium events*

**1. Listening for Selenium events**

While we can control the flow of testing very well with basic Selenium, we could also listen for general events within Selenium to customize behavior, every time a mouse is clicked, a page is visited or any other number of other features. This gives us greater control over testing, and a creative way to expand what is being tested on our site in a very universal manner. So, adding in the event fire is really easy. We can take any base driver, in this case we are using the FirefoxDriver, or we could be using the Chrome driver, or we could be using a remote driver, it doesn't matter. And we wrap that driver in the EventFiringWebDriver. So, this guy here, this driver we are going to use allows us to register event listeners. And then go about whatever test we are going to go do, we have a couple of commands right here. We will come back to those in a little bit, but I could run 50 test cases after this, it doesn't matter. As long as I'm using this driver I will be listening for these events. So the events basically are fired by the driver, and I listen by implementing this AbstractWebDriverEventListener out here. And it gives me a bunch of options, we can see inside of here a couple of options that we have from the base. So, before we click on something, we can intercept and do something we want to do. After we click on something, the same sort of thing, if there is an exception or you do the same sort of thing. And let me just show you real quick if I do override implement inside of here, you can see there is a lot of different options. So, before I change a value or after I change a value, find by before and after, I go backwards, forwards or navigate to, these are the commands inside of Selenium.   
*In the Selenium IDE Package Explorer, the listener node is expanded. It lists two files – ListenerExample.java and UniversalChecks.java – which are open on separate tabbed pages in the main pane.  
  
In the ListenerExample.java file, the first part of the code is shown, as follows:  
  
package listener;  
  
import org.openqa.selenium.By;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.firefox.FirefoxDriver;  
import org.openqa.selenium.support.events.EventFiringWebDriver;  
  
public class ListenerExample  
{  
       public static void main (String[] args) throws Interrupted Exception {  
  
               WebDriver baseDriver = new FirefoxDriver ();  
               EventFiringWebDriver driver = new EventFiringWebDriver(baseDriver);  
               driver.register(new UniversalChecks());  
               try  
               {  
               driver.navigate().to("http://localhost:8080/website/home.vpd");  
               driver.findElement(By.linkText("Another Menu Item")).click();  
               driver.navigate().back();  
  
               Thread.sleep(500);  
               driver.findElement(By.linkText("Menu Bar Text")).click();  
               driver.navigate().back();  
  
       } finally  
  
The presenter highlights the following two lines in the code:  
  
               WebDriver baseDriver = new FirefoxDriver ();  
               EventFiringWebDriver driver = new EventFiringWebDriver(baseDriver);  
  
Next the presenter opens the UniversalChecks.java file. The top part of the code is shown, as follows:  
  
package listener;  
  
import org.openqa.selenium.By;  
import org.openqa.selenium.NoSuchElementException;  
import org.openqa.selenium.NotFoundElementException;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.WebElement;  
import org.openqa.selenium.support.events.AbstractWebDriverEventListener;  
  
public class UniversalChecks extends AbstractWebDriverEventListener;  
{  
       private String url;  
  
       @Override  
       public void beforeClickOn(WebElement element, WebDriver driver)  
       {  
               url = element.getAttribute("href");  
       }  
  
       @Override  
       public void afterClickOn(WebElement element, WebDriver driver)  
       {  
               try  
               {  
                       WebElement footer = driver.findElement(By.id("footer"));  
                       if (! footer.getText().contains("©")  
                       {  
                       System.err.printline("Footer not found on page " + url);  
                       }  
               }  
  
       @Override  
       public void onException(Throwable throwable, WebDriver driver);  
       {  
               if (throwable instanceof NotFoundException)  
               {  
                       System.err.printline("Exception found that may be due to test failure ");  
               }  
       }  
}  
  
In the code, the pointer highlights the items beforeClickOn, afterClickOn, and onException.  
  
The presenter right-clicks an area on the file to open a shortcut menu and selects the option Override/Implement Methods. The Override/Implement Methods dialog box contains a list of methods, each with a checkbox. Examples are afterChangeValueOf(WebElement, WebDriver), afterFindBy(By, WebElement, WebDriver), afterNavigateBack(WebDriver), afterNavigateForward(WebDriver); afterNavigateTo(String, WebDriver), afterScript(String, WebDriver), and beforeChangeValueOf(WebElement, WebDriver).*   
  
Driver.navigate.back or forward or to, this is not some sort of clever page workaround. This is just simply an intercept when your test calls the Selenium commands, it will actually let you just sneak something inside of here. So, let's see how we can use this. So in this test, what I'm doing is before I click on some element, I am going to go off and save the URL. Now, I need this for my logging more than anything else. This is important though, because once I get to here this WebElement is basically read-only. If I do this line of code afterwards, if I do this line of code over here, you will actually get an exception because this is failed, because I clicked on something. It's gone to the link, it's moved forward. And so that, you know that information about what was in this tag is gone at that point, and that tag is no longer around. So, what this test is doing by the way, is it's going every time we click on the link, I want to investigate the page it's on, because maybe I have a rule in my organization that says every page on our site must have a footer, and that footer must contain a copyright notice. So here, when I click on the link then I'm going to check hey, is there is a footer out here. And if there is not a footer out here I am going to get a NoSuchElementException, and we can say there is not a footer on this page, that's strike number one. If there is a footer on the page, then it must contain a copyright notice. Now I could do more specific copyright notice, I could check this year's date, you know there is a number of things I could do with regular expressions or whatnot. The idea is just to say hey, you can check it here.   
*The presenter highlights the following lines of code:  
  
       @Override  
       public void beforeClickOn(WebElement element, WebDriver driver)  
       {  
               url = element.getAttribute("href");  
       }  
  
He then highlights the two lines immediately below this, which are  
  
       @Override  
       public void afterClickOn(WebElement element, WebDriver driver)  
  
Next the presenter scrolls down in the UniversalChecks.java file. The code that's visible now is as follows:  
  
       @Override  
       public void beforeClickOn(WebElement element, WebDriver driver);  
       {  
               url = getAttribute("href");  
       }  
  
       @Override  
       public void afterClickOn(WebElement element, WebDriver driver);  
       {  
               try  
               {  
                       WebElement footer = driver.findElement(By.id("footer"));  
                       if (! footer.getText().contains("©"))  
                       {  
                               System.err.println("Copyright not in footer on page " + url);  
                       }  
               } catch (NoSuchElementException e)  
               {  
                       System.err.println("Footer not found on page " + url);  
                }  
       }  
  
In the code, the presenter highlights the following lines:  
  
public void beforeClickOn(WebElement element, WebDriver driver);  
       {  
               url = getAttribute("href");  
       }  
  
       @Override  
       public void afterClickOn(WebElement element, WebDriver driver);  
  
The presenter then highlights the following lines:  
  
                      WebElement footer = driver.findElement(By.id("footer"));  
                       if (! footer.getText().contains("©"))  
                       {  
                               System.err.println("Copyright not in footer on page " + url);  
                       }  
               } catch (NoSuchElementException e)  
               {  
  
The presenter then highlights the following lines:  
  
                      WebElement footer = driver.findElement(By.id("footer"));  
                       if (! footer.getText().contains("©"))  
                       {  
                               System.err.println("Copyright not in footer on page " + url);  
                       }  
               } catch (NoSuchElementException e)  
               {*   
  
And again if there is not a copyright notice I say, hey, there is not a copyright notice. And in my case, I'm not failing the test, I'm not adding failures, I am logging it and not even just logging into System.err, but I could do any number of things here. Mostly, I probably just want to note it and be able to have somebody to address it later on. Now, here as well I don't have to deal with the exception, I could say find elements and if there is no footer elements of the size equal to zero, I could put the error out there. I am doing this, because I want to show you any time there is an exception I can catch it as well. Now in this case, if I see a NotFoundException, based off of the example above, it's probably something wrong with my test case in most cases. In this case, it's not. In this case, it is the NotFoundException is an indication of something is missing on the page, but I really should code my test cases to avoid that. If there is a chance that something is not out there, I should probably use you know with an s, plural s and then double check it that way. But I just want to show you every time there is an exception inside of here, and this guy will get called even when I am handling the exception, and I just want to be able to show you that example inside of here. So, let's go ahead and run our test here, now as it runs it's going to go up to the page, it's going to do some navigation, the two pages I'm going to click on they are not correct, because it wouldn't be interesting if they were. So, the first one we click on here you can see, we are clicking on "Another Menu Item" outside of here. It goes to a page that indeed does have a footer, but the footer does not have a copyright. So, I get this first error off of the check,   
*The presenter scrolls further down in the file to show the following code:  
  
       @Override  
       public void oneException(Throwable throwable, WebDriver driver);  
       {  
               if (throwable instanceof NotFoundException)  
               {  
                       System.err.println("Exception found that may be due to test failure");  
               }  
       }  
  
The presenter runs the test and the Console pane below the script returns the following messages:  
  
Copyright not in footer on page http://localhost:8080/website/pages/drag.html  
Exception found that may be due to test failure  
Footer not found on page http://localhost:8080/website/nowhere.html*   
  
because it has a footer, but it does not have a copyright so I get that to check. And the second one has no footer at all inside of it. And so that page again the exception out there and say hey, the page is not found in nowhere.html. Now, in this case nowhere.html is actually nowhere at all, our 404 page is the one not getting it, but this line message could drive you to that. So, anyway the point being is Selenium event listeners give you that little extra burst you need, to be able to check things on your site. So, whether you want to validate that click is going to be valid. Maybe you're doing Ajax stuff and you want to, you know do something a little bit different inside of there, you want to set something up, you want to save something, you want to log something. Or you can check is this set across to all the site. Use this to creatively expand out your testing, and be able to get the most out of your testing across all tests, without even having to plan it into every single call.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Extracting Code

Learning Objective

*After completing this topic, you should be able to*

* *extract source code from the Selenium IDE*

**1. Extracting source code**

The ability to record test steps while just being a user of the site is so tempting, it has to make you think it's preferable to use the Selenium IDE over a coded solution. Well, you'd be half right. While recording is a great, quick way to start tests, it's also very limited in the scope and power compared to a coded test. So, we could take advantage of this little cool feature in the IDE to start our coded tests within Selenium – and then customize our tests inside of code from there. So, I have this little test here and, just to show you I've recorded it already, I can **Play** it real quick – it logs in. You put in a username, a password, submit…again, I could have many, many steps here; it doesn't matter the number of steps – it's just a…showing I have a sequence of steps that seems to work pretty well. Now, once I have this to a point where I'm happy with it and I know it's what I want to do, I can copy this over. So, if you right-click – you see here, you can see there's a **Copy** option, **Ctrl+C** copy option. But, when I copy it, the question is what's getting copied? Well, underneath the **Options** menu, there's something called the **Clipboard Format**, and there's many different ways I can go and capture this. So, I can do **HTML** and, if I do **HTML** and I **Copy** this HTML here, I basically am copying this out as if it was just straight code that's going to go for Selenium. And this is what it would look like here – doing it inside of a Word doc, just to kind of keep it simple inside of there. But it's just...   
*A web page at the address /localhost:8080/website/login.vpd is open. It contains Username and Password text boxes, and a Login button.  
  
In the Selenium IDE, the Test Case pane lists a test case named Untitled. The Table tabbed page is open. It includes a table with Command, Target, and Value columns that lists the steps in the test case, as well as Command, Target, and Value text boxes.  The Target text box is associated with Select and Find buttons.  
  
The table currently contains five rows. Row 1 lists the open command and the target /website/signon.vpd. Row 2 lists the command type, the target id=usernameInput, and the value admin. Row 3 lists the command type, the target id=passwordInput, and the value test. Row 4 lists the command clickAndWait and the target name=submit. Row 5 lists the command assertText, the target css=h2, and the value Administrator Home.  
  
The presenter clicks Run to run the test and the web page changes to show a successful login.  
  
Next the presenter selects the rows in the table, right-clicks to open a shortcut menu, and points out the Copy option.  
  
He then expands the Options menu and selects Clipboard Format. This opens a submenu with the options HTML, Ruby / RSpec / WebDriver, Ruby / Test::Unit /WebDriver, Ruby / RSpec / Remote Control, Ruby / Test::Unit / Remote Control, Python2 / unittest / WebDriver, Python2 / unittest / RemoteControl, Java / JUnit4 / WebDriver, Java / JUnit4 / WebDriver Backed, Java / JUnit4 / Remote Control, Java / JUnit3 / Remote Control, Java / TestNG / Remote Control, C# / NUnit / WebDriver, and C# / NUnit / Remote Control.  
  
The presenter selects the HTML option. With all the steps in the test case selected, he then right-clicks and selects Copy.  
  
The presenter opens a Word document and pastes the copied content in. It appears as HTML code. For example, the first part of the code is as follows:  
  
<tr>  
  
       <td>open</td>  
       <td>website/signon.vpd</td>  
       <td></td>  
  
</tr>  
  
<tr>  
  
       <td>type</td>  
       <td>id=usernameInput</td>  
       <td>admin</td>*   
  
a table where I'm showing the steps as recorded. This is the file format that gets saved when I save these Selenium tests. You might have noticed here, though, there's formats for Ruby tests, Python tests, Java tests, C# tests. I want to show you quickly a few of these. Let me go pick the Java WebDriver version. So, I'm going to **Copy** this, and then I'm going to go over to my Java IDE and I can just take those steps and I can paste it inside of there. Now look, all of a sudden, it's pasted this in with the driver inside of here, a (baseUrl) – so these are things I've actually had to set up beforehand. If I paste it in without presetting up my code…it has several compiler errors, but I've imported everything, I've set up a driver, I've set up a baseUrl, I've even…you know, set up all the imports I need. And it's going to do all the steps it needs as if I was quoting them. So it does find By.ids, find By.ids, find By.names, and it even adds in JUnit code to basically check what is the test that's going on inside of here. So, it says assertEquals is the finding of the item inside of here. I go find the element and I'm getting the Text element, matching the text I wanted to have out here. Now, the advantage to this, is I could customize this much further. The "Administrator Home" is hardcoded here. I could look this up in a database.   
*The presenter closes the file without saving it and returns to the Selenium IDE. With the steps in the test case still selected, the presenter selects Options - Clipboard Format - Java / JUnit4 / WebDriver. He then right-clicks the table and selects Copy.  
  
The presenter opens the Java IDE. In the Selenium Package Explorer, a recording node contains the file Starter.java. The contents of the file display in the main pane.  
  
The presenter clicks below the following, existing lines of code in the file:  
  
@Test  
public void recordedStarter()  
{  
  
He then pastes in the copied content, which appears in Java code. The code is as follows:  
  
       driver.get(baseUrl + "website/signon.vpd");  
       driver.findElement(By.id("usernameInput")).clear;  
       driver.findElement(By.id("usernameInput")).sendKeys("admin");  
       driver.findElement(By.id("passwordInput")).clear;  
       driver.findElement(By.id("passwordInput")).sendKeys("test");  
       driver.findElement(By.name("submit")).click();  
       assertEquals("Administrator Home", driver.findElement(By.cssSelector("h2")).getText());*   
  
I could go and find some numbers of items. I could get the keys I want to send – username, passwords – I can get that from a database or from a…you know, back-end source. I could go off to an LDAP server, or something like that, to be able to get a valid username/password without having to hardcode this inside of my tests. But I'm not limited to Java. So, if I come back to my solution here…so if I hide my Java one here, I could go through and say, What's my clipboard format? Well, let's say we want to do some C# testing. So, I can **Copy** this again – don't forget to Copy it again, it doesn't actually change what's on the clipboard – and this…I've come over here to my C# solution. Again, I've imported the items I need. I have set up my…my driver, I've set up my URL to the baseURL inside of there, and I paste this in and look. It's now in C# format…the capital F for FindElement. It looks a lot like Java code, but you can see it actually is C#, and it has the ng test that's searched on inside of here. And so it's really very smart; it's going to paste this inside of here. But I'm not going to leave you at that. Let me go one more of these. I can go to the **Clipboard Format**, I can switch it over to my Python test. And now I can **Copy** this over and I can go to my Python option inside of here. And here's my…oops, that's the wrong one…here is my...   
*The presenter returns to the Selenium IDE and, with the steps in the test case still selected, selects Options - Clipboard Format - C# / NUnit / WebDriver. He then right-clicks the table and selects Copy.  
  
The presenter opens an existing C# file. Code already in the file includes the following segment:  
  
{  
       [Test Class]  
       public class Recording  
       {  
               [TestMethod]  
               public void TestMethod1()  
               {  
                       IWebDriver driver = new FirefoxDriver();  
                       String baseURL = "http://localhost:8080";  
  
Immediately below this code, the presenter pastes in the copied test case, which appears as C# code. The code is as follows:  
  
                       driver.Navigate().GoToUrl(baseURL + "/website/signon.vpd")  
                       driver.FindElement(By.Id("usernameInput")).Clear();  
                       driver.FindElement(By.Id("usernameInput")).SendKeys("admin");  
                       driver.FindElement(By.Id("passwordInput")).Clear();  
                       driver.FindElement(By.Id("passwordInput")).SendKeys("test");  
                       driver.FindElement(By.Name("submit")).Click();  
                       Assert.AreEqual("Administrator Home", driverFindElement(By.CssSelector("h2")).Text);  
  
The presenter returns to the Selenium IDE and, with the steps in the test case still selected, selects Options - Clipboard Format - Python2 / unittest / WebDriver. He then right-clicks the table and selects Copy.*   
  
PyDev solution, all right? And so, again, I have the imports, I have the driver, I have the baseURL. I paste this inside of here – and this, obviously, looks different from the other two. Python code looks a lot different, but I have the driver, I have the find\_element\_by, I have all the stuff inside of here. My test…in theory, might work straight out of the box. And so, I can go through and I can **Run** this, and it's going to pop it up and it's going to run the test, and here it comes. Oop…and something failed. So, I'm not necessarily all the way down…let's see what failed inside of here – just out of curiosity. Oh, it's because I named it wrong; baseURL. That's my fault inside of here; base\_url. Okay, so that's a quick little fix, just to see that this is real code. I just don't want to, you know, fake it off. This is real code and, in this case, I'm doing my Python test. And…oh, I'm still failing here.   
*The presenter opens an existing Python file. It already contains the following code:  
  
import unittest  
  
from selenium import webdriver  
  
class Test(unittest.TestCase):  
  
       def testName(self):  
               driver = webdriver.Firefox()  
               baseURL = "http://localhost:8080"  
  
Immediately below this code, the presenter pastes in the copied test case. It appears in Python code, as follows:  
  
driver.get(self.base\_url + "/website/signon.vpd")  
driver.find\_element\_by\_id("usernameInput").clear()  
driver.find\_element\_by\_id("usernameInput").send\_keys("admin")  
driver.find\_element\_by\_id("passwordInput").clear()  
driver.find\_element\_by\_id("passwordInput").send\_keys("test")  
driver.find\_element\_by\_name("submit").click()  
self.assertEqual("Administrator Home", driver.find\_element\_by\_css\_selector("h2").text()  
  
The presenter opens a dialog box and selects Python unittest to run the code. The Console pane below the code returns an error. The relevant part of the error message is  
  
AttributeError: 'Test' object has no attribute 'base\_url'  
  
The presenter returns to the code in the Python file and changes the line baseURL = "http://localhost:8080" to  
  
base\_url = "http://localhost:8080"  
  
He then runs the code and the same error is returned.*   
  
Oh, it has it as self. Let me change that real quick. I can change this code real quick and that's my fault. I should've thought forward on this a little bit better. And here's my test and, just like that, you can see the whole thing ran and worked. So, just like that, I've gone from a recorded test inside of my Selenium IDE, to either a C# or a Ruby or a Python or a Java test. Just copy and paste it over and there's the foundations. Now, there are some commands that won't copy over. There are some things that are in the IDE that don't exist inside of the WebDriver, but you can just find coded solutions around them. It's not that it's not possible; it's just that it's not those simple commands. You might have to put a couple of things together, but it certainly is a great way to get started and not have to go through each line and figure it out as you want…as you need to build your tests. So, use that if you choose, and really take advantage of the two of them – the Selenium IDE to capture tests quickly, and the power and flexibility there is within your coded solutions.   
*The presenter changes the line driver.get(self.base\_url + "/website/signon.vpd") to  
  
driver.get(base\_url + "/website/signon.vpd")  
  
The presenter runs the code again. This time it executes successfully. The username and password are entered on the web page, the Login button is clicked, and the Administrator Home page opens.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Multiple Browsers

Learning Objective

*After completing this topic, you should be able to*

* *run Selenium tests in multiple browsers*

**1. Testing multiple browsers**

Our web pages may work great in one browser but, to be totally sure, we ideally will test across many browsers. We can utilize Selenium to help with cross-browser testing, by enabling different drivers for our common tests. So, we've a really, really basic test here, and the first driver we can look at is the HtmlUnitDriver. Now, this is – by far – going to be the quickest browser that's out…the driver that's out there. You can see; the test is done almost instantly when I hit the button. The problem is I don't get to see anything that happened. If it fails, I have no idea what happened; it just ran through everything and was done. So, for a quick check – once you know everything's working – that works well, but isn't necessarily the best browser for development…the best driver for development, I should say. Now, getting to a real browser, we can go through and switch to the FirefoxDriver(). This is the one that's the easiest to work with; it's the one that Selenium's built on. And, as we **Run** this one, we'll see here; it's going to actually launch an instance of Firefox. It's going to go through and, you can see, a bunch of stuff just happened really quick there. It logged into the web page and it did the little test. If we go back here, it actually typed in a Username, it typed in a Password, hit the **login** button, and went through and ran that. And so that's the Firefox browser; we'll see that one quite a bit. Now, there's many other browsers out there, but they require a little more config. So, for instance, we can switch over to the Internet Explorer browser.   
*In the Selenium Package Explorer, a file named LocalTest.java is open. The code in the file is as follows:  
  
import org.openqa.selenium.By;  
  
public class LocalTest  
{  
       public static void main(String[] args)  
       {  
               WebDriver driver = new HtmlUnitDriver();  
  
//              WebDriver driver = new FirefoxDriver();  
  
//              System.setProperty("webdriver.ie.driver", "E:\\Apps\\Dropbox\\Selenium\\Java\\IEDriverServer.exe");  
//              WebDriver driver = new InternetExplorerDriver();  
  
//              System.setProperty("webdriver.chrome.driver", "E:\\Apps\\Dropbox\\Selenium\\Java\\chromedriver.exe");  
//              WebDriver driver = new ChromeDriver();  
  
//              new OperaDriver();  
//              new SafariDriver();  
  
               driver.get("http://localhost:8080/website/signon.vpd");  
  
               WebElement element = driver.findElement(By.id="usernameInput"));  
               element.sendKeys("admin");  
  
               element = driver.findElement(By.id="passwordInput"));  
               element.sendKeys("test");  
  
               element = driver.findElement(By.name="submit"));  
               element.click();  
  
               System.out.println("Test Completed")  
       }  
}  
  
In the code, all lines identifying specific web drivers, except for the line WebDriver driver = new HtmlUnitDriver();, have been commented out.  
  
The presenter clicks the Run button and the Console pane below the code immediately returns the message "Test Completed."  
  
Next the presenter comments out the line WebDriver driver = new HtmlUnitDriver(); and removes the comment symbols from the line WebDriver driver = new FirefoxDriver();. He clicks the Run button again.  
  
A web page titled Administrator Home opens in Firefox. The presenter navigates back in the browser to show that on the page at http://localhost:8080/website/signon.vpd, the test entered a username and password, and then clicked the Login button – causing the Administrator Home page to load.  
  
Next the presenter comments out the line WebDriver driver = new FirefoxDriver();. He then removes the comment symbols from the following lines:  
  
               System.setProperty("webdriver.ie.driver", "E:\\Apps\\Dropbox\\Selenium\\Java\\IEDriverServer.exe");  
               WebDriver driver = new InternetExplorerDriver();*   
  
So, the driver for Internet Explorer requires you to tell it where this driver lives on your machine. So, you have to go out and download this executable…the executable file, and put that on your machine somewhere – and this is the place I've chosen to put it in. I've just put the system property out here, and then I can go and **Run** this test. Now, as we're running this test, let me tell you upfront – this is going to be a really slow test. For whatever reason, Internet Explorer…like, each keystroke is painful going into this. And so, while the other test was instantaneous…for Firefox moved right along, you can see we've done three characters of the five-character word on the first one inside of here.Just know that this is great for checking stuff out; there might some tweaks you can do to speed it up but, for the mind, it's just terrible. I'm glad I'm using a very unsafe password there – only four characters. Now, you can see the test is finished; it's rolled up everything, and the Internet Explorer test has worked just fine. We can move along then. Another very common browser out there is Chrome.   
*The presenter clicks the Run button. The test opens the page at http://localhost:8080/website/signon.vpd in Internet Explorer. Very slowly, it enters a username and a password, and clicks the Login button. The Administrator Home page then opens.  
  
The presenter returns to the code, comments out the lines related to Internet Explorer, and removes the comment symbols from the following lines:  
  
               System.setProperty("webdriver.chrome.driver", "E:\\Apps\\Dropbox\\Selenium\\Java\\chromedriver.exe");  
               WebDriver driver = new ChromeDriver();*   
  
So, again, we have to go to get the chrome.driver out there. And we have to go and get a driver file – an executable file we download – and we can **Run** on top of that. And so, here pops up Chrome; Chrome pops over, runs the whole test almost instantly – very, very fast inside of there. But, ideally here, you can see how easy it is to actually switch out the test. The test steps are the same in every single one of these. We open a site, we find the element based off of the ID, typing in some keys, finding the next ID, typing in some keys, hitting…finding the next button outside of there, clicking the button, and that's our test. We're not really doing a lot inside of there, but we can do it across other browsers. Now, there's other browsers out there you can use. You can use the Opera browser, you can use the Safari browser. I don't have those locally on my machine; it's a Windows machine, not a Mac. But, if you're a Mac user, you can totally use these; if you're a Windows user, you can totally use these. There's an Opera version for Windows out there as well – it's just very uncommon. So, I'll leave that to you to test out and use your imagination. The whole point being, though, is once you have your test suite up and running, it's quite simple to switch out the driver and be able to use this across multiple tests. You'll at least be able to know, ideally, that your web sites are working perfectly across all browsers or, at least, where there's some variances. Where there's variances, you can choose whether it's a test issue or an actual issue. But, at least, you can see the variances and then be able to focus your testing on those browsers accordingly. And, in many cases, you can eliminate a lot of browser testing by not having to do full tests on multiple browsers.   
*The presenter clicks the Run button, the test runs, and the Administrator Home page opens in Chrome.  
  
Finally the presenter scrolls down to focus on the part of the code that runs the test. This code, again, is  
  
               driver.get("http://localhost:8080/website/signon.vpd");  
  
               WebElement element = driver.findElement(By.id="usernameInput"));  
               element.sendKeys("admin");  
  
               element = driver.findElement(By.id="passwordInput"));  
               element.sendKeys("test");  
  
               element = driver.findElement(By.id="submit"));  
               element.click();  
  
               System.out.println("Test Completed")  
       }  
}*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Selenium Grid

Learning Objective

*After completing this topic, you should be able to*

* *use Selenium Grid to manage multiple tests*

**1. Using Selenium Grid**

After we've constructed a fantastic suite of tests, we need to be able to use them in the most flexible way possible. Selenium Grid is a tool that gives us this flexibility to run our tests in parallel, select details of different test environments, and so much more. It takes a little bit of configuration to get started, but it opens a world of capabilities. So let's take a look. So here we're going to talk about some testing where, instead of using a local driver, we're going to use a remote driver. And we're going to point to a remote driver that actually is a hub, and will spread out the testing across multiple servers, multiple environments, based off of what our desired capabilities for that environment is. And that's based off of what our need for testing is. So to start off the Hub, we have a little script here. But we, basically, are going to our standalone server, Selenium standalone server. We just download a jar file, and we're running that jar file underneath the role of hub, as we can see right there. Now, this starts it up, but it doesn't really do any testing itself. It will then distribute to a bunch of nodes that will register themselves with the server. So those nodes can be run on any machine. I'm actually going to be running them on the same machine right here, just because it's more convenient. But I can be running this over on a LINUX machine, or over on a UNIX machine, or over on a Mac, or any environment. I could just have a whole fleet of Windows machines, if I wanted to.   
*The presenter opens a Java file named GridRunner.java. It contains the following code:  
  
public class GridRunner  
{  
       @Test  
       public void runOnGrid() throws MalformedURLException;  
       {  
               DesiredCapabilities capability = DesiredCapabilities.firefox();  
               capability.setBrowserName("firefox");  
               capability.setPlatform(Platform.WINDOWS);  
               capability.setVersion("29");  
  
//              DesiredCapabilities capability = DesiredCapabilities.chrome();  
//              capability.setBrowserName("chrome");  
//              capability.setPlatform(Platform.WINDOWS);  
  
               WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capability);  
  
               new LoginTest().loginSteps(driver);  
  
               driver.quit();  
       }  
}*   
  
Now, here I'm going to have another script here, that's going to run this for a Firefox version. So it's the same server executable, it's just a jar file. So when you have Java and the jar, you can run this on whatever environment you want to. In this case so, the role is node, you can see underneath here, and we're going to specify for the browser that's going to be used for this node, we're going to use the Firefox version number 29. Okay, and we've registered ourself as a hub, and you can see I'm pointing to the URL that I want to register myself as. So in this case, it's local. It's the same name machine I'm on, and the host is 4444 by default. That's what it starts up as. But I could be pointing to another machine on my network, and then that could be acting as the hub, and then you can come to, and farm it off to me as the node that could be running on a Windows machine, a LINUX machine, a UNIX machine, as we said. Now, we can do more than one hub here. We're going to do nodes…excuse me…here, and I'm going to do another one here on my machine. You can only have one hub. You can have more than one node; this is going to be a Chrome version. So in this case, since I'm using Chrome, I have to point to – you can see up above my text here – the chrome driver. And that Chrome Driver is going to be the guy who connects to the Chrome executable. It's a download you get from Selenium and you use that there. And similarly, I use the exact same jar file underneath the role of node, but now we're going to set up, instead of the hub alone, we're going to register ourselves to a browser.   
*In the command prompt window for C:\Windows\system32\cmd.exe, the current directory is E:Apps\Dropbox\Selenium\Server. The presenter enters the command hub. He then enters the following command:  
  
java - jar selenium-server-standalone-2.41.0.jar -role hub  
  
The output is as follows:  
  
May 20, 2014  10:38:11 PM org.openqa.grid.selenium.GridLauncher main  
INFO: Launching a selenium grid server  
2014-05-20 22:38:11.913:INFO:osjs.Server: jetty-7.x.y-SNAPSHOT  
2014-05-20 22:38:11.944:INFO:osjsh.ContextHandler:started o.s.j.s.ServletContextHandler[/,null]  
2014-05-20 22:38:11.950:INFO:osjs.AbstractConnector:Started SocketConnector@0.0.0.0:4444  
  
In a new command prompt window for C:\Windows\system32\cmd.exe, also with the prompt set to E:Apps\Dropbox\Selenium\Server, the presenter enters the command gridFirefox. He then enters the command  
  
java -jar selenium-server-standalone-2.41.0.jar -role node -browser browserName=firefox, version=29 -hub http://localhost:4444/grid/register  
  
The output specifies that the system is launching a selenium grid node, adding the browser name and version, and registering the node to the hub at http://localhost:4444/grid/register.  
  
In a new command prompt window with the prompt set to E:Apps\Dropbox\Selenium\Server, the presenter enters the command gridchrome. He then enters the command  
  
java -Dwebdriver.chrome.driver="E:Apps\Dropbox\Selenium\Java\chromedriver.exe" - jar selenium-server-standalone-2.41.0.jar -role node -hub http://localhost:4444/grid/register -register browser browserName=chrome -port 5556  
  
The output specifies that the system is launching a selenium grid node, adding the browser name, and registering the node to the hub at http://localhost:4444/grid/register.*   
  
And the browser is going to register itself as chrome. I didn't put in some of the details, so I didn't say whether I'm a version, or what not, just Chrome in general. So we don't care on this version. So going back to our test, our desired capability then says "for this test execution, I want you to run, in this case, you see on a firefox browser". The browser name is going to be Firefox, I'm on a Windows platform, version 29. Maybe I have multiple versions of Firefox I want to test this on, so I can specify it out here. If I don't do the versions, it's going to pick any Firefox instance to run it on. We don't know where it runs, we just know that it runs out there. So now let's go ahead and execute this guy, and we go off and run it. It's on some of the machines. So somewhere it's going to pop up a Firefox instance, and that machine is going to have that, you know, window going to pop up. It's going to go through the test cases, and then it's going to come back to us. So in this case, it's going to go off to our instance, right here…sorry, the instance right here, and it's going to start that testing. Sorry, I've a little bit of pause right there, because I had the mouse selected inside of there for my highlighting. But now we can go back to our Firefox instance, and our Firefox instance shows that it ran those steps. You can see we have finding element, finding element, finding element, and then a click right here. You can see this; I clicked right here, those are all the steps that is part of my login test. Again, the steps aren't important here. I can see the steps if I want to, the steps are just over here. Again, I find an item, I send some keys, I find the button, then I click it. Again, the test is unimportant. I'm sending these now off to the remote server.   
*The presenter returns to the code in the GridRunner.java file, which again is as follows:  
  
public class GridRunner  
{  
       @Test  
       public void runOnGrid() throws MalformedURLException;  
       {  
               DesiredCapabilities capability = DesiredCapabilities.firefox();  
               capability.setBrowserName("firefox");  
               capability.setPlatform(Platform.WINDOWS);  
               capability.setVersion("29");  
  
//              DesiredCapabilities capability = DesiredCapabilities.chrome();  
//              capability.setBrowserName("chrome");  
//              capability.setPlatform(Platform.WINDOWS);  
  
               WebDriver driver = new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capability);  
  
               new LoginTest().loginSteps(driver);  
  
               driver.quit();  
       }  
}  
  
Note that in the code, the lines for the Chrome browser are currently commented out. The presenter clicks the Run button.  
  
The console where the presenter originally entered the command gridFirefox returns the results of running the login test. The presenter points out lines in the output, indicating that the system was finding each of three elements – an entered username, an entered password, and then a submit element. He also points out a line showing that the system executed a click.  
  
In the GridRunner.java file, the presenter selects loginSteps in the line  
  
new LoginTest().loginSteps(driver);  
  
He then right-clicks to access a shortcut menu and selects Open Declaration. This opens a separate file named LoginTest.java, which contains the following code:  
  
package login;  
  
import org.openqa.selenium.open.By;  
  
public class LoginTest  
{  
       public void loginSteps(WebDriver driver);  
       {  
               driver.get("http://localhost:8080/website/signon.vpd");  
  
               WebElement element = driver.findElement(By.id="usernameInput"));  
               element.sendKeys("admin");  
  
               element = driver.findElement(By.id="passwordInput"));  
               element.sendKeys("test");  
  
               element = driver.findElement(By.name="submit"));  
               element.click();  
       }  
}*   
  
Now, this ran within the Firefox option. I could very easily, instead of my test, you know, say "let's run our chrome option real quick, let's see how that goes", and then go off and run this test. And now, instead of running over on the Firefox instance, it ran on the Chrome instance. It opened up a chrome browser somewhere, on some remote computer somewhere, and it executed everything, and it came back, and my test passed. So I can switch instances that easy. This opens up a world where now I can create a test and, from my desktop, test it on many different machines, many different browsers, and not even have to use up my resources. I could have a suite of tests, where each test that executes goes off and hits a different node and runs in a different node. That way, I can have the optimized machines out there with the versions installed, and nothing else running, or whatever I want to. And it even opens up the ability for me to run these tests in parallel. Since I am distributing across many computers, I can test faster because I can run tests in parallel that way. And even better, I can hit many instances on my machine at the same…on my web site at the same time. And thus, I can actually do some performance testing, driven by the same tests I have a lot of quality and checks on. So I can not only have performance testing for slamming on a machine, I can know it's behaving properly because I'm checking each step, and validating – through my unit tests – all of the steps inside of…I've coded inside of Selenium. So adding Grid into the mix gives us some powerful runtime options and capabilities, after I built all my tests. So take a look at that, and consider it for rounding out your testing and finishing up your test plan.   
*In the GridRunner.java file, the presenter comments out the lines that relate to Firefox and removes the comment symbols from the lines for Chrome, which are as follows:  
  
               DesiredCapabilities capability = DesiredCapabilities.chrome();  
               capability.setBrowserName("chrome");  
               capability.setPlatform(Platform.WINDOWS);  
  
The presenter then clicks the Run button and opens the command prompt window where he originally ran the command gridChrome. The console returns the results of running the login test.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Test Planning

Learning Objective

*After completing this topic, you should be able to*

* *plan Selenium tests*

**1. Planning Selenium tests**

Selenium testing focuses on automated testing through a web browser. To that end, we need to have a better understanding of the elements required to test a web site and how Selenium can assist us in testing it or not. Let's take a little time to look at a generic web site and to see how we can plan to test it, and where Selenium comes in. So on this site here, we just have some general stuff you can see. I mean there are some dynamic elements, there are some static elements, and there is actually a lot of blank space that we would fill in with something more interesting. The first thing we'd want to understand on any web site is, "What is the static content and is the static content appropriate?" Now, the static content could be something you choose to put within your Selenium testing, it could be something you don't. If it's truly static, it's truly just a one-time look at it and won't change, then there's not much point to automating it. But if it's something that's static content out of a database or out of a server where files are uploaded, it might very well be useful to have tests that would compare the two sources. And so I could go through in my Selenium testing and say: okay, these menu items up here are they labeled properly, or maybe some text that comes in here? Now, one big thing in Selenium is that we can handle is links. Well there are a lot of links inside of here and it takes a lot of time for a human to go through and click every link and make sure it goes some place.   
*A web page at the address localhost:8080/website/home.vpd is open in a browser. It contains a top-level menu with the links home, news, calendar, contact us, login, and no popups. Along the side of the page are three links – Menu Bar Text, Another Menu Item, and Menu Bar Option.  
  
The main part of the page contains a pane with the question "What language will you choose?", five radio buttons – Selenium IDE, Java, C#, Python, and Ruby, and a Vote button. The page also contains an Upcoming Activities section and a News section, each with various links.*   
  
And so what we can do inside of Selenium is we can actually automate this. And let me just show you a little test that I came up with, it's just kind of a quick way to test this site specifically. And so I have a Selenium link, a Selenium test here that really is going to go find on a page every single one of the "a" text. And this is a link text. So we're going to go through that and we are going to loop through every link inside of there. And there's a lot of logic, convoluted logic that's going around inside of here to make that happen. Because going back and forth on a page makes it difficult. But when it gets down to it, I'm going to count all the bad links that would show up. And then I can make a report here that the following links are bad. So if I went off and ran this test here, we can see I have now done with a tiny, tiny bit of Selenium code a whole lot of work. And so here is my test running. You can see it's clicking through every single one of the links that's on that page. And it comes back and says, "Oh my gosh! I found two failed links inside of here." There's a link that goes to an empty and a link that goes to nowhere. Now, I could report better and tell you exactly what the text was on top of there. I'm just showing you as a demo. But again, if I click this you know that works but these two over here take you to, in this case our canned 404 page.   
*In Selenium Package Explorer, the presenter opens a file named AllLinks.java. The file contains the following code:  
  
public void checkAllLinks()  
{  
       WebDriver driver = new FirefoxDriver();  
       driver.get("http://localhost:8080/website/");  
  
       Set<String> clickedLinks = new HashSet<String>();  
       Set<String> badLinks = HashSet<String>();  
  
       int = linksTested = 0;  
       All\_Links:  
       while (true)  
       {  
               Link<WebElement> links = driver.findElements(By.Id.tagName("a"));  
               if (linksTested == links.size())  
               {  
                       break All\_Links;  
               }  
               for (WebElement link : links)  
               {  
                       String href = link.getAttribute("href");  
                       if (clickedLinks.contains(href))  
                       {  
                               continue;  
                       }  
  
                       if(href.trim().length()==0)  
                       {  
                               linksTested = linksTested + 1;  
                               continue;  
                       }  
  
                       link.click();  
                       clicked.Links.add(href);  
                       linksTested = linksTested + 1;  
                       List<WebElement> headers = driver.findElements(By.tagName("h2"));  
                       for (WebElement header : headers);  
                       {  
                               if (TEXT.404.equals(header.getText());  
                               {  
                                       badLinks.add(href);  
                               }  
                       }  
                       driver.get("http://localhost:8080/website/");  
                       continue All\_Links;  
               }  
       }  
       driver.quit();  
       if (badLinks.size() > 0)  
       {  
               StringBuilder message = new StringBuilder();  
               message.append("The following links are bad\n  ");  
               for (String link: badLinks);  
               {  
                       message.append(link);  
                       message.append("\n  ");  
               }  
               fail(message.toString());  
               }  
        }  
}  
  
The presenter clicks Run. The web page opens and each of the links it contains is clicked automatically.  
  
The test completes and the Selenium IDE lists the following two failed links:  
  
http://localhost:8080/website/empty.html  
http://localhost:8080/website/nowhere.html  
  
The presenter opens the web page and clicks two of the links – Menu Bar Text and Menu Bar Option – to show that they both lead to a page that contains the message "The page you are trying to load does not exist!"*   
  
Now again, this has to be very specific to the site. You have to know, in this case, what your 404 page or your link looks like. See where it ended up going to in order to be able to evaluate it. But it's something we can do in Selenium for a static element like a link. Now, obviously within Selenium, we can do all sorts of functional matters. So if we are going to login. I can type any username. I can type in a password. I can type in invalid ones. I can check if the validation is working properly. So I can make sure it works as desired. I can validate that. I can make sure it's working in obvious ways that's not working. I can even look for subtle errors. So a real subtle error that comes up is a little bit of site hacking. So what if I went over here to the adminHome? I have it in my history page right here and I go to that page. Well, notice this site here has a little bug; it actually takes me to a page called admin home. It actually gives me your account over here and there's really nothing else over here. But it gives me the chance to go to the change password page. Now, who knows what this functionality does and it certainly should not be available when there's a login here. So this type of test can help you test your security matters. Just having your browser randomly jump to different pages might be a useful thing to see if your security is set up properly on your site.   
*Next the presenter opens a page at localhost:8080/website/login.vpd. It contains Username and Password text boxes and a Login button. The administrator demonstrates entering values – including ones that are invalid – and clicking the button to test the functionality of the page.*   
  
And then obviously a big area is Ajax. Every time we add Ajax onto a page, we add another dynamic feature. And if we have Ajax on many of these pages, then all of a sudden we have to go and test across all those pages "Is the Ajax working?" Well, if I can create an automated set of steps to go through that, then all of a sudden, I can execute those steps on any page if that Ajax element shows up. So within Selenium, I can check static elements to get static elements. I can add code. I can even go off and check items in the database or on a file server or other sources, load them up in my test and then compare them to what's being shown on the site. And as we saw, I can see links or jumping around navigation. These are all ways you might not think of using Selenium but can add a lot of value to your automated testing, by taking away hours and hours of time of having somebody doing a repetitive task of typing things in the browser and jumping around randomly. And instead let the computer do that for you with the little bit of clever design and a little bit of clever testing.   
*The presenter clicks home to open a page at /localhost:8080/website/adminHome.vpd. It contains the title Administrator Home, a set of menu options – home, news, calendar, contact us, login, and no popups, an expandable Your Account node, and a New and Uncompleted Notices section, which currently contains the message "No Notices to Handle." Expanding Your Account reveals a Change Password option.  
  
The presenter clicks Change Password. This opens a page with Old Password, New Password, and Old Password Repeated text boxes, as well as a Change Password button.   
  
The presenter clicks home to return to the page at localhost:8080/website/home.vpd.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Testable Site Design

Learning Objective

*After completing this topic, you should be able to*

* *design testable sites*

**1. Designing testable sites**

While we're mostly talking about automated testing here, the level of effort of building automated tests can be reduced by understanding, and using, testable design patterns in building our websites. Let's talk about a few of these ideas that can be applied in our architecture design and implementation, even to the lowest details, that can have a positive impact on the outcome and effort it takes to build tests. So at a high level, when we architect our site, we want to be able to do things like using reusable frameworks. Now, they could be frameworks for purchasing, it could be frameworks for building, but this little demo site we're using here, you can see, has built-in capabilities to manage security centrally. The temptation might be to just code in security on each link; checking has the user logged in, has the user done this, that, or the other. But doing that means I have to do security tests, all over the place…security as one example. Or it could be validation, or it could be file uploads, or it could be image management, anything inside of there, something that we can, basically, centralize a service that's going to be used in many spots on our site, allows us, as we can see, to manage that centrally. I can manage the groups, I can manage the roles, I can manage the resources people want to be able to use, and stuff like that. So I can manage this all, and know it's going to be done at a URL level, not at a page level.   
*A web page at the address localhost:8080/website/viewAllUsers.vpd is open. It contains the heading View Existing Users, an Add User button, a Username column with an admin radio button and two buttons – Edit User and Delete User, and an Active Date/Inactive Date column, currently set to 12/29/2013/No inactive set.  
  
The page also contains a menu with options for managing the site, organized in a tree structure with expandable nodes. The top-level nodes are Your Account, Admin Functions, Site Content, and Interactive Content. One of the options nested under Admin Functions is Site Security. In turn, it contains the options Manage Users, Manage Groups, Manage Roles, and Manage Resources. The Manage Users node contains View Users and Add User options.  
  
The Site Content node includes the nodes Manage News, Manage Calendar, Manage Bookmarks, Google Maps, Content Lists, Menus, and File Content. The Menus node includes Manage Menus and Add New Menu options.  
  
The presenter expands the Manage Roles node, which contains View Roles and Add Role options. He also expands the Manage Resources node, which contains View Resources and Add Resource options.  
  
The presenter then expands the Manage Bookmarks node. It contains View Bookmarks and Add Bookmark options. He clicks View Bookmarks and the View Bookmarks page opens. It contains an Add Bookmark button, Edit and Delete buttons, and a table with Page and Active Date/Inactive Date columns, for listing bookmarks. The menu options for managing the site remain visible on the left of the page.*   
  
That really helps. In general, we want to…like this menu over here on the side, this menu shows up across all of these pages. So I can go look at any of these items inside of here, and our menu is going to show up consistently, and it's going to be reused. In this case, the web site is actually taking a HTML page, and it's including that HTML content on every page that comes up. And so once I've tested on this menu on one site, I've tested this menu on all sites. I don't have to worry about it over, and over, and over again inside of here; I only have to test it the one time. So another feature we can talk about here is, we want to externalize as much of the static content as we can from the HTML code we're producing. In this case, this site, again, has pages, and pages, and pages of all the text you would see on the web site; all the prompts, all the error message, everything else you can see here is something that could be managed by users, not by the developers building the code. The advantage of this is, if they want to change something, if there's something wrong, they can actually do that as part of their user acceptance testing. It's not on the development team to go back, change the code, update the HTML, republish it, check it again, and go through that cycle every time the code is changed. If we externalize the features, we can really help letting administrators and owners manage the site on themselves. Again, another design tip, it'll eliminate the need to do static text checking on your site, possibly. Now, within there, though, we want to make sure we're following good usability standards. So in this case, you can see we have the back and forth color. It's not necessarily a bad thing, maybe a questionable color combination inside of here, but we want to be very careful about how we're applying these things inside of our browser.   
*The presenter expands the Manage News node, which contains three options – View Stories, Add Story, and Add Slideshow. He clicks View Stories and the View Stories page opens. It contains a table with four columns –  Title of Story, Type, Number of Pages, and Active Date/Inactive Date. It also contains four buttons – Edit Story, Delete News Item, Add New Story, and Add Slideshow.  The menu options for managing the site remain visible on the left of the page.  
  
The presenter expands the Site Maintenance node, which contains the options View Monthly Stats, View Logs, and Manage Text options. He clicks View Monthly Stats and a page displays a graph that represents the numbers of visitors to the site per month.  
  
Next the presenter clicks Manage Site Text and a page titled Manage Text Resources provides options for adding new resources and a long list of all existing text resources on the site, along with options for editing or deleting them. In the list, every second row is colored purple.*   
  
We want to apply them in a way that it meets usability standards, as if you were doing something for a browser that's for sightless people, or something like that. Because then we can consistently apply style sheets and elements like that across our site, we don't want to necessarily hand code…or, you know, either within a automated solution or within HTML… the style elements inconsistently through the site. We want to be able to take advantage of style sheets and design principles like that, that are also a part of usability. It's going to improve our performance, as far as testing goes. All right, so the other thing we can talk about here is, here is a little test that we got started. It's adding an item inside of here, and it's dynamically generating based off of here. It's adding items inside of here. So as we're doing our testing here…and what happened was, I was on a different page so it restarted. I'm adding in elements inside of here, but if I look at my Selenium helper, I go to the IDE here…and let me actually delete all the stuff here. And I wanted to click on one of these, for instance, I've got to go off and I can select – I want to click on this one. Now, if I go back to here, notice I have a really nasty bit of…in this case it's DOM, or XPath, or whatever I have going on inside of here. I could be using CSS Selectors, or whatnot. None of these are really all that pleasant. It's very deeply embedded inside of here, it's not very testable at all. And again, you can see, it's optionBeingEdited, number two.   
*The presenter closes the Manage Text Resources web page. He then opens a page with the heading Add a Poll. It contains a Poll Name text box and Active Date and Inactive Date text boxes – each with a calendar button. It contains a Question text box that currently contains the question "Is this helpful?", a Possible Answer text box with an Add button, three radio buttons – Yes, Could be better, and Nope, and Edit, Delete, Up, and Down buttons. The page also contains a Language drop-down list box with a Translate button, and Save Poll and Cancel buttons. A menu organized into a tree structure is available down the left side of the page.  
  
The presenter types What in the Possible Answer text box and clicks Add. A radio button titled What then replaces the three original radio buttons, Yes, Could be better, and Nope. He types I'm not sure in the Possible Answer text box, and a radio button titled I'm not sure is added below the What radio button. Finally the presenter types Yes in the Possible Answer text box and clicks Add, and a Yes radio button is added.  
  
The presenter opens the Selenium IDE, deletes existing rows, and, in the Command text box for a blank row, types click. He then clicks the Select button alongside the Target text box and, on the web page, selects the I am not sure radio button. The entry document.addPollForm.optionBeingEdited is automatically added to the Target text box.  
  
The presenter clicks the down-pointing arrow alongside the entry and a drop-down list of alternative targets displays. It describes the currently entered target as dom name and also provides options described as xpath:attributes, xpath:idRelative, and dom:index. The presenter selects the xpath:attributes option and the value in the Target text box changes to xpath=(''input[@name='optionBeingEdited'])[2].*   
  
So this is really a micro element you could do in your design, where what I'm going to name these items inside of here, what I am going to name these individually, if I go and look at it here, its value is number 1. Well, a better option is to put something, such as the text that we just typed in, as that value. By putting that inside of there instead – as the value – it allows us to, within our testing, be able to closer say  "I typed this text, I validated this text", as opposed to having to take a best guess and a shell game of what the order of this was. So the rule inside of here is use as specific of data as possible. As much as you can, make sure you're naming the elements. You can see this element is not named, put an ID on the element. Again, maybe you don't put the value, the value can be 1, but you put an ID on the element that matches the text that's inside of there. But give testers the option and the opportunity to be able to tie things specifically on the screen, without having to do complicated searches for them. The last thing we'll say is just, as you're building websites, know that there's going to be bleeding edge technology that's just not going to be automated. You can't just do automated testing around it. Selenium has a lot of issues around testing HTML5 elements, because it's so new, and because it's not consistently used across browsers. Again, if you're building a site that only works on a limited set of browsers; it'll only work in Chrome, it will only work in IE, it'll only work in whichever, then you're certainly limiting your audience, and you're also limiting your testability. So take all of these ideas, and think about it, and give that feedback. If you're not a developer, give that feedback to a developer to say "hey, can you try this, can you try this", because often it's really just something they haven't thought of. And with a little bit of some tweaks to the high level and movable design, we can reduce the number of tests we have to execute, as well as improving the efficiency and the ease at which we can do the testing we do have to do.   
*The presenter right-clicks the I am not sure radio button on the web page and selects Inspect Element from the shortcut menu that opens. Along the bottom of the page. the Inspector pane then lists the HTML for the element.  
  
In the HTML, the presenter highlights the line  
  
<input type="radio" value="1" name="optionBeingEdited"></input>*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Test Suite Design Tips

Learning Objective

*After completing this topic, you should be able to*

* *apply design tips for Selenium test suites*

**1. Design tips for test suites**

Selenium has a lot of great tools, but applying them correctly, and in smart ways, can make our testing better, quicker, and easier to maintain. While the goal of Selenium is testing, automated testing is also the development of code. So we can talk about some best practices used for constructing tests. Let's take a look at some helpful tips that can help in designing better tests. So the first thing that might seem obvious to you, but wouldn't necessarily come up right away, is if you have repeated steps that might be something you're testing directly, it might be something you test in many different ways, or it might just simply be a prerequisite towards doing some tests. You can actually create methods and classes, and whatnot, to help you reexecute those tests. So we have some really simple steps inside of here, and we're talking login steps. We're passing them into a driver inside of here. We could actually pass in the drivers to get structure, you can pass them in the method, it doesn't really matter. There's a lot of different ways to construct it. But we can repeat these steps. In fact, we're kind of static inside of here, but we should be really passing in a username and a password to this functionality. And if we did that, and replaced this out here, then we have a truly reusable test. So I had a partially reusable test here, but you can see, now I can truly get away from coding for one specific situation,   
*A file named Login.java is open. It contains the following code:  
  
package login;  
  
import org.openqa.selenium.open.By;  
  
public class LoginTest  
{  
       public void loginSteps(WebDriver driver);  
       {  
               driver.get("http://localhost:8080/website/signon.vpd");  
  
               WebElement element = driver.findElement(By.id="usernameInput"));  
               element.sendKeys("admin");  
  
               element = driver.findElement(By.id="passwordInput"));  
               element.sendKeys("test");  
  
               element = driver.findElement(By.name="submit"));  
               element.click();  
       }  
}  
  
The presenter changes the line public void loginSteps(WebDriver driver); to  
  
public void loginSteps(WebDriver driver, String username, String password);  
  
He then changes the line element.sendKeys("admin"); to element.sendKeys(username); and changes the line element.sendKeys("test"); to element.sendKeys(password);.*   
  
and code this functionality as a reasonable feature. If I want to test it, I can give it a good username and good password, I can put a bad username and password. The same steps, though, of finding the element for username, finding the element for password, hitting the **Submit** button, can be repeated and done over, and over. And then afterwards, I can either login successfully and move on, or I can validate "hey, this is a bad login", let's look at the screen, let's look at what failed, let's look at what's successful. And so it's up to this method, just simply, to execute some common steps, and then everybody else to figure out how it's used, save some copy and paste time. It's one really key item. Now, another here is, don't be limited to how you design your test suite. Now right over here, I'm showing you Java. You could be doing C#, you could be doing Ruby, you could be doing Python, doesn't matter. But within your code suite, use your unit testing framework. In this case, I have a unit testing framework that's going to allow me to use a singleton here, to start up my driver. And then I can close my driver at the end. I'm not going to manage that outside of everything else. I could use exception handling, or whatever, to ensure that that guy gets handled properly. Now, past that, in this case using Java, I'm using JUnit. I'm throwing a listener inside of here. Now, the listener is a very specific purpose, because when I go and run my test case here, let's say I get, for instance, a NoSuchElementException. So when I run this test case, I'm looking up something that doesn't exist in the page. Now, is that a failure on the web site? maybe, but only if it truly doesn't exist there. It might very well be a test failure, itself. And so this might be my opinion coming from a development side, but my opinion is if you fail because of a test failure, most likely a test failure like this, somebody who wrote the test should go off and double-check it.   
*The presenter opens a file named Runner.java. It contains the following code:  
  
package design;  
  
import org.junit.runner.JUnitCore;  
import org.junit.runner.Result;  
  
public class Runner  
{  
       public static void main(String[] args);  
       {  
               DriverManager.getInstance().start(DriverManager.FIREFOX);  
  
               JUnitCore core = new JUnitCore();  
               core.addListener(new SeleniumListener());  
               Result results = core.run(ImprovedDesign.class, Sample.class);  
  
               DriverManager.getInstance().end();  
       }  
}  
  
The presenter highlights the line   
  
DriverManager.getInstance().start(DriverManager.FIREFOX);  
  
He then highlights the line  
  
DriverManager.getInstance().end();  
  
The presenter opens a file named SeleniumListener.java. It contains the following code:  
  
package design;  
  
import org.junit.runner.notification.Failure;  
import org.junit.runner.notification.RunListener;  
import org.openqa.selenium.NoSuchElementException;  
  
public class SeleniumListener extends RunListener  
{  
  
       @Override  
       public void testFailure(Failure failure) throws Exception  
       {  
               if (failure.getException() instanceof NoSuchElementException)  
               {  
                       System.err.println("Selenium Test Design Failure");  
                       System.err.println(failure.getTestHeader() + "=> \n" + failure.getMessage());  
               }  
       }  
}  
  
The presenter highlights NoSuchElementException in the line  
  
if (failure.getException() instanceof NoSuchElementException)*   
  
Just to make sure that the test itself isn't broken, as opposed to being the site broken. And the test author should be responsible for that. And so again, we go through, and we run this inside of here, and we…and as we run this inside of here…and I've caused some errors later, because I changed that method. But as I run this inside of here, it's going to go off. It's going to execute this site, it's going to fail the test, and then it's going to come back, and it's going to say "hey, by the way, you need to go and look at this error that we came up inside of here". So as we go off, we run the test, and it did run, and it didn't print out what it was supposed to in the console. But it would come over here, and it would, you know, give you, say "hey, by the way, System.err", and you could do a login, or whatever you want to. We think there's a design failure out there. So this is an interesting strategy you can wrap around Selenium with other elements. Don't feel bound by just what Selenium provides. Also, as we move into code, again, whether it's C#, or Java, or Ruby, or Python, use the other features you develop within the code, as well. In this case, for my database test, I'm kind of cheating off the one that's on there. I've the same steps, but instead of putting in a hard-coded username password, I'm actually going to go to my manager (some code I've written that talks to the database), I'm going to get all the active users, and then pull out the first user, and use that user for this test.   
*The presenter returns to the Runner.java file. He clicks the Run button. In the Save and Launch dialog box, LoginTest.java is listed in the Select resource to save pane. The presenter clicks OK.  
  
The Errors in Workspace dialog box, which includes Proceed and Cancel buttons, contains the following message:  
  
Error(s) exist in required projects:  
SeleniumIntro  
Proceed with launch?  
  
The presenter clicks Proceed and the test fails.  
  
The presenter re-opens the SeleniumListener.java file and points out the lines  
  
System.err.println("Selenium Test Design Failure");  
System.err.println(failure.getTestHeader() + "=> \n" + failure.getMessage());  
  
The presenter opens a file named UseDB.java. It contains the following code:  
  
package design;  
  
import org.junit.Test;  
org.openqa.selenium.By;  
org.openqa.selenium.WebDriver;  
org.openqa.selenium.WebElement;  
  
public class UseDB  
{  
       private UserManager manager;  
       private WebDriver driver;  
  
       @Test  
       public void dbTest()  
       {  
               User user = manager.getActiveUsers().get(0);  
  
               driver.get("http:localhost:8080/website/signon.vpd");  
  
               WebElement element = driver.findElement(By.id("usernameInput"));  
               element.sendKeys(user.getUserName());  
  
               element = driver.findElement(By.id("passwordInput"));  
               element.sendKeys(user.getPassword());  
  
               element = driver.findElement(By.name("submit"));  
               element.click();  
       }  
}  
  
The presenter points out the line  
  
User user = manager.getActiveUsers().get(0);*   
  
I mean, I could use a test that runs through all the users, or part of the users. The point being is, I can actually look into my database, see what data is there, and compare that or use that to drive my web site, or validate what's on the web site. And so in this case, I pass in the username, and I'm passing the password from a real user object, so it should work, because I'm cheating, I'm going behind the scenes. And again, I'm testing the web site, so that's what, you know, makes it all useful out there. Now, again, let me go back and fix this here, you know, let me take this out, so we can look at our last test. The point being is, when we get down to it, even I've done it here in this little demo. Most of the time, you're going to see examples of things that aren't really great design. Good design should do things like this, when I'm going off and I'm checking text, I'm saying "hey, this is the text that should show up on the menu". Well, when I check this text, do I really want to hardcode this in my Java solution? not necessarily. What if the text changes? I don't want to have to go recompile, and recode, and rechange these tests. I want to, again, look up this test…this text in a file, or in the database, or just simply externalize it from the code itself. The locator is inside of there, so I have this very complicated XPath locator that's going to go through and find where this menu item we are going to validate on top of there is, that should be externalized.   
*The presenter switches to the LoginTest.java file and changes the line public void loginSteps(WebDriver driver, String username, String password); to just  
  
public void loginSteps(WebDriver driver);  
  
The presenter then opens a file named ImprovedDesign.java. It contains the following code:  
  
package design;  
  
import org.junit.Before;  
import org.junit.BeforeClass;  
import org.junit.Test;  
org.openqa.selenium.By;  
org.openqa.selenium.WebDriver;  
org.openqa.selenium.WebElement;  
  
@SuppressWarnings("deprecation")  
public class ImprovedDesign  
{  
       // Externalize text on the screen  
       private String menuText     = "Manage Calendar";  
       // Externalize complex locators  
       private String menuItemLocator = "//table[@id='ygtvtableel23']/tbody/tr/td";  
  
       private WebDriver driver;  
  
       @BeforeClass  
       public void getReady()  
       {  
               DriverManager.getInstance().start(DriverManager.FIREFOX);  
       }  
  
       @Before  
       public void setup()  
       {  
               DriverManager.getInstance().getDriver();  
       }  
  
       @Test  
       public void testSuccessfulLogin()  
       {  
               //Move common steps in many tests to its own file  
               new LoginTest().loginSteps(driver);  
  
               WebElement menuItem = driver.findElement(By.xpath(menuItemLocator);  
               assertEquals("Menu item is shown and text matches", menuText, menuItem.getText());  
       }  
  
       @AfterClass  
       public static void finish()  
       {  
               DriverManager.getInstance().end();  
       }  
  
       public String getMenuText()  
       {  
               return menuText;  
       }  
  
       public void setMenuText(String menuText)  
       {  
               this.menuText = menuText;  
       }  
  
       public String getMenuItemLocator()  
       {  
               returnMenuItemLocator;  
       }  
}  
  
The presenter highlights the line  
  
private String menuText     = "Manage Calendar";  
  
He then highlights the code   
  
//table[@id='ygtvtableel23']/tbody/tr/td";*   
  
I can use my BeforeClass before my testing framework things to be able to set up my driver, to be able to get a clean instance of each time. And then within my test, I should only be focusing on the test steps, and then closing down things afterwards. Again, you can see my getters and setters, so I can inject items into my test. Either I'm using a dependency injection framework, or just simply doing it by hand. This test design will vastly improve the way I'm going to be able to create tests and modify tests, because as my web site adjusts, I need to have my test adjust as well. Otherwise, it's just going to fail. And if I want my test to always be working, making them easier to fix is really a great thing. So take these thoughts and put those in your test design, and really engineer your test design, as opposed to just spitting it out for one-time use only.   
*In the ImprovedDesign.java file, the presenter points out the @Before, @BeforeClass, and @AfterClass sections of the code.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Extended Example

Learning Objective

*After completing this topic, you should be able to*

* *work with an extended sample test case in the Selenium IDE*

**1. Working with an extended test case**

While Selenium gives us a bunch of tools to help execute tests, we still must plan out the flow and data we need to use as part of testing. Where some test organizations, they capture detailed step-by-step instructions for testers to walk through each time, as to make the test reusable and repeatable. For other organizations, they just simply play with the site and see what happens. Selenium allows to do a bit of each as we can record our actions and go back and formalize the tests. We can edit them. Let's take a look at how we can create a test in long form and use a real functionality as an example from our site. So, this functionality we're going to look at here allows us to add a poll. Now, I'm not going to get super formal inside of this. I'm just going to basically dive right in and start testing. So, I have my Selenium IDE up and it's set up to record. So let me just start interacting with it. So I'll give the poll name Test Poll inside of here. And then I can play with dates and what not. I'm not going to mess with that at this point, but there's a lot of interactive features right here on the page. So, Is the, so as before Is the test working? And so one of the answers is **Yes**, yes the test is indeed working. The other answer is **No**, no the test is not working. And we're going to say **Maybe**, maybe the test is working, we'll figure it out from there. Now, I've done a bunch of steps here. And notice Selenium has got all the stuff recorded for us: what we've typed in, opened the page, what we've typed in, the buttons we have clicked, all that sort of stuff. It's happening, it's going behind the scenes. I don't have to worry about Selenium, it's keeping up with me.   
*In a browser, a locally stored web page is open. The web page includes the title Add a Poll and contains a Poll Name text box, Active Date and Inactive Date text boxes – each with a button for accessing a calendar, a Question text box, and a Possible Answer text box with an associated Add button and Edit, Delete, Up, and Down buttons. It also contains a Language drop-down list box with a Translate button, and Save Poll and Cancel buttons.  
  
In the Selenium IDE, the Test Case pane contains the entry Untitled. The presenter clicks the Record button. On the web page, he then types Test Poll in the Poll Name text box, types Is the test working? in the Question text box, types Yes in the Possible Answer text box, and clicks Add. A Yes radio button is added below the text box. The presenter types No in the Possible Answer text box and clicks Add. He then Maybe in the Possible Answer text box and clicks Add again. Three radio buttons – Yes, No, and Maybe – now display below the text box.  
  
In the Selenium IDE, the table on the Table tabbed page contains a row for each test step that the presenter has performed. Row 1 lists the command open and the target /website/addPoll.vpd?dispatch=edit. Row 2 lists the command type, the target name=name, and the value Test Poll. Row 3 lists the command type, the target name=question, and the value Is the test working? Row 4 lists the command type, the target name=answer, and the value Yes. Row 5 lists the command clickAndWait and the target name=buttonPressed. Row 6 lists the type command, the target name=answer, and the value No. Row 7 lists the command clickAndWait and the target name=buttonPressed. Row 8 lists the command type, the target name=answer, and the value Maybe. Row 9, the final row, lists the command clickAndWait and the target name=buttonPressed.*   
  
Now, there's different things I want to test here. So I'm going to check here, did our text actually show up here, and then did our text actually show up here, and then did our text actually show up here? Okay we want to make sure that after we've hit those buttons, did the text actually show up? Alright, so that is a first good test. Basically interactions, did the things go put inside of there, go inside of there and then I can save it. Alright so we save it here. And then I can confirm that it did save. I can say "Okay, let's insert that there, that did save." And maybe just to make sure everything is working here, I can go over and click **Edit**. And then I can see is that still out here? Okay, **Yes** and assert and assert. Alright, let me just **Cancel** just to close that out and there you go. And, you know what, I'm going to be clean inside of here. I'm going to go through and click there and hit **Delete** and then I'm going to confirm that. So, these are out here and we confirm that. Okay so now I'm going to go back to No Polls defined. So there's my first test. Now, I'm going to turn off my recording here and I can give it a shot, see if it all runs and works. So, run through and look at that, so we added it, we confirmed it, it tested it, it deleted it. All seems to work, there's a test. Great thing about this is I can run this test a 1000 times. I can run it over and over again, but I'm not stuck with it. You know, when I go through, you know as I was doing it, I didn't necessarily plan ahead perfectly. I did a bunch of asserts inside of here. You can see asserted **Yes**, **No**, and **Maybe**. And were they all there, I asserted it after I added them all.   
*On the web page, the presenter selects and right-clicks the Yes radio button and, in the shortcut menu that opens, selects the option assertText css = p Yes.  
  
Similarly, he selects and right-clicks the No radio button and selects assertText css = p No, and then selects and right-clicks the Maybe radio button and selects assertText css = p Maybe.  
  
On the web page, the presenter clicks the Save Poll button. This opens a page with the title View All Polls. It contains an Add New Poll button and a table with three columns – Poll Name, Number of Responses, and Active Date/Inactive Date. The Poll Name column contains a Test Poll radio button and, above and below it, two sets of Edit, Clear, and Delete buttons.  
  
The presenter selects the Test Poll radio button, right-clicks it to open a shortcut menu, and selects the option assertText //div[@id='content']/form/table/tbody/tr[6]/td[2] Test Poll.  
  
Next the presenter selects the Test Poll radio button and clicks the Edit button. This opens the Update Poll page, which contains the same controls and user entries as the Add a Poll page.  
  
The presenter selects and right-clicks the Yes radio button and again selects the option assertText css = p Yes. He selects and right-clicks the No radio button and selects assertText css = p No, and then selects and right-clicks the Maybe radio button and selects assertText css = p Maybe.  
  
The presenter clicks Cancel to return to the View All Polls page. He then selects the Test Poll button and clicks the Delete button. This opens the Delete Poll page, which contains a table that lists the question Is the test working? and contains the options Yes, No, and Maybe, each with a zero in brackets to indicate number of responses. The page also contains Confirm Delete and Cancel buttons. The presenter clicks Confirm Delete. The View All Polls page re-opens.  
  
The presenter returns to the Selenium IDE. The table contains a new row for each of the steps that the presenter performed. For example, the second-last row lists the command clickAndWait and the target document.viewPollsForm.buttonPressed. The last row lists the command clickAndWait and the target name=buttonPressed.  
  
The presenter clicks the Record button to stop the recording and then clicks the Run button. The test executes and green highlighting indicates that the test was successful.*   
  
What if I wanted to actually assert them as they went? Well, that's no trouble, I can go back and fix that. I can say you know, after I click on **Yes**, let's assert the Yes is there, and after I click on **No**, let's assert that No's there. And then **Maybe**, okay and then I can run this again just make sure. Okay, I tweaked it, it's running well. Alright, now that doesn't test everything out there, I haven't tested a lot of validations. But I can go through and let me just save my test case inside of here. And we can call this first test inside of here, we'll call it happyPath as something going on inside of here. And then I can create another **New Test Case** here. So I'm going to create another test case and just put next to the other one. And so now I'm going to start recording again. And I'm going to say **Add New Poll**. Alright, and so now we test some validation, I'm going to hit the **Save Poll** button. Alright, oh look there are some errors that have happened here. So let's assert that this happened. Alright and then we can assert that this happened here, and assert, and go on like this. And so we can throw in whatever additional tests we want to have happen here. I can go through and check it again. And I could set up and assert that that text is no longer there. I can do whatever I want to add inside of here. So, I could add, you know, over here directly if I want to, if I want to add the right steps or what not. I can hit the **Add** button here, okay, and say "Okay, Possible Answer is required, let's go ahead and assert that, that's going on." Okay, so that's good too, that's working inside of there. And so we're doing a bunch of testing inside of here. And then again I can go through and I can execute this test. So, let me just **Play** this specific test right here. And I can do that, sorry my bad, I messed up my test here because we didn't start it at the right location. So let's start here, and we'll do **Cancel** and then now we're in the right position, so then I can run this test. So, that's another error I'd have to go through. I'd have to go through and. See and I'm starting at the wrong page there, because I'm just recording things. So, let's go back to my **admin home**, let's go to **View Polls** right here and then update this because there's some extra stuff I need to do inside of here, so I can go through and fix that. And now I can try and run it again. Okay so it's a little bit better. So we got the poll open, we hit the button. Oh you see, we have an exact inside of here. And so maybe, we don't want an exact inside of here. So this is the type of thing that we would be doing inside of our testing. We're doing a bunch of assertions inside of here and we need to go through and it saves some stuff that may not be the saved the way we want to. And so again, I can go through and tweak these inside of here, and I'd have to do more. So, I'm going to call it right there. But before I do that let me go through and actually save my second test case.   
*The presenter drags a row that lists the assertText command and the value Yes to below the rows containing the type and clickAndWait commands for the Yes radio button. He does the same with the rows containing assertText commands for the No and Maybe radio buttons, moving them to below the rows containing the relevant type and clickAndWait commands.  
  
The presenter clicks the Run button and the test executes successfully.  
  
In the IDE, the presenter selects File - Save Test Case. In the Save As dialog box, he accepts the current folder location, types happyPath in the File name text box, and clicks Save.  
  
In the IDE, the Test Case pane now lists the entry happyPath.  
  
Next the presenter selects File - New Test Case. The table is cleared and the entry Untitled appears below happyPath in the Test Case pane.  
  
The presenter clicks the Record button. On the View All Polls web page, he clicks the Add New Poll button. This opens the Add a Poll page. The Poll Name, Question, and Possible Answer text boxes are blank.  
  
The presenter clicks the Save Poll button and the following message appears above the page heading:  
  
Please correct the following errors  
Poll Name is required.  
Question is required.  
At least one option is required  
  
The presenter selects and right-clicks the text "Please correct the following errors" and, from the shortcut menu that opens, selects the option assertText css=form[name="addPollForm"] exact:Please correct the following error...  
  
The presenter selects and right-clicks the text "Poll Name is required" and, from the shortcut menu that opens, selects the option assertText css=li Poll Name is required.  
  
The presenter then selects and right-clicks the text "Question is required" and selects the option assertText css=//div[@id='content']/form/ul/li[2] Question is required.  
  
The presenter selects and right-clicks the text "At least one option is required" and selects the option assertText css=//div[@id='content']/form/ul/li[3] At least one option is required.  
  
On the Add a Poll page, the presenter then types fads in the Poll Name text box and clicks the Save Poll button. The following message displays:  
  
Please correct the following errors  
Question is required.  
At least one option is required  
  
The presenter switches to the IDE and notes that he could choose to add steps directly to the table. He then returns to the Add a Poll page and, without entering a value in the Possible Answer text box, clicks the Add button. The following message displays:  
  
Please correct the following errors  
Possible Answer is required.  
  
The presenter selects and right-clicks the text "Possible Answer is required" and, in the shortcut menu that opens, selects the option assertText css=li Possible Answer is required.  
  
The presenter returns to the Selenium IDE. The table contains a row for each step that the presenter performed. For example, the second-last row lists the command clickAndWait and the target name=buttonPressed. The last row lists the command assertText, the target css=li, and the value Possible Answer is required.  
  
The presenter clicks the Run button. The test fails and the second row in the table, which lists the command clickAndWait and the target name=buttonPressed is highlighted in pink to indicate an error. An error message also displays on the web page.  
  
The presenter returns to the Add a Poll page and clicks Cancel. The View All Polls page displays.  
  
The presenter returns to the IDE and clicks the Run button. The same error is returned.  
  
The presenter clicks the admin home menu option on the page to navigate to the Administrator Home page. He then clicks View Polls to open the View All Polls page.  
  
In the IDE, the presenter changes the target listed in the first row from /website/addPoll.vpd to /website/addPoll.vpd?dispatch=edit to match the actual address of the All Polls page. He clicks Run again and the test fails. The fourth row in the table, which lists an assertText command  is highlighted in pink.  
  
The presenter selects the row and removes the word exact from the Value text box.  
  
The presenter selects File - Save Test Case.*   
  
This is going to be my errorHandling test case and now I have a test suite, and I can go through and I could actually save the test suite itself. And so for this test suite, this is going to be my pollTesting inside of there. And so now, I have all this great stuff up and running. This is the way I can start with constructing my tests. I record it and it gives me the foundation to go through and then tweak it from there and figure out exactly what's going on. I can go step by step and say hey, what's working and what's not working. And this one is not failing, I tweak it till it is working – this is failing and then tweak it till it's working again. But again, in some cases it works perfectly and your testing goes through just fine. In other cases it fails along the way. So, this is a good foundation for you to be able to go off and build tests using the tool. But understand you will need to understand the behind the scenes to really make full use of testing and the really most interesting tests happen.   
*In the Save As dialog box, the presenter accepts the current folder location, types errorHandling in the File name text box, and clicks Save.  
  
In the Test Case pane, errorHandling is now listed below happyPath.  
  
Next the presenter selects File - Save Test Suite. In the Save As dialog box, he accepts the current folder location, types pollTesting in the File name text box, and clicks Save.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

Practice: Implement and Manage an Extended Test Case

Learning Objective

*After completing this topic, you should be able to*

* *implement and manage an extended test case*

**1. Exercise overview**

So let's take a few minutes now to try out a few of the skills we've been discussing.

You are working in the IT department of EasyNomad Travel Inc. Your latest task is to extend your company's Selenium test scripts to provide better test coverage of your company's web site. Put your knowledge to use and answer the following questions.

**2. Working with an extended test case**

Question

In order to use Selenium IDE to test a web page that contains Ajax and cookies, you will require some advanced features of Selenium IDE.  
  
Match the following Selenium IDE commands with their descriptions.

**Options:**

1. assertCookieByName
2. storeCookieByName
3. waitForVisible
4. waitForText

**Targets:**

1. Verifies that a cookie exists and the cookies value matches a supplied value
2. Can be used to create a Selenium variable containing a cookies value
3. Waits for an element to become visible on the web page
4. Pauses the Selenium script until a provided string appears

Answer

*assertCookieByName will match the cookie name with a provided value for that cookie.*

*storeCookieByName can store cookies and their values to a variable for more advanced processing in the Selenium IDE test script.*

*waitForVisible will wait for an element to become visible, this is generally needed when waiting for Ajax elements to appear on the web page.*

*waitForText will wait for a string, or text, to appear on the screen. Ajax calls can sometimes be delayed and this will pause the script until the results can be displayed.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Question

Some of the new features of your company's web site use advanced JavaScript and you need to write tests that can access these JavaScript functions from within your Selenium IDE test cases.  
  
Enter the Selenium IDE command to run the JavaScript element referenced in the Target field.

**Code**  
INSERT THE MISSING CODE

Answer

*The Selenium IDE command to run a JavaScript function on the web page is the runScript command.*

**Correct answer(s):**

1. runScript

Question

You need to evaluate the functions of a complex web page using Selenium IDE commands. Match the following actions with the Selenium IDE command that should be used.

**Options:**

1. The web page will add elements with CSS Ids for new calculations and you need to ensure the correct number of elements are created
2. You need to calculate a percentage value to test a calculation on a stored variable
3. The calculation has completed and you need to check if your stored test value matches the result from the web site
4. You need to access a stored variable to do a comparison to ensure the test was successful

**Targets:**

1. storeCssCount
2. storeEval
3. assertEval
4. storedVars

Answer

*storeCssCount can take an XPath query and use that to count the number of matching elements.*

*storeEval allows operations, like division and multiplication, on stored variables.*

*assertEval can be used to compare a stored variable to a value to ensure it is correct.*

*storedVars can be used in your Target to access variables that have already been stored.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Question

You want to use the Selenium Code generation feature to copy and paste script into Java.

Options:

1. Clipboard Format
2. Options
3. Format

Answer

***Option 1:****The****Clipboard Format****option will present you with a list of software languages in which you can copy and paste the Selenium script as code. If you do not select a software language, the default "HTML" is usually used.*

***Option 2:****The****Options****option lets you configure and even add formats for export to source code, but it is not the option that enables you to choose the language to cut and paste the code to Java.*

***Option 3:****The****Format****option has been disabled in recent Selenium versions, but can be turned on to allow recording in other languages than Selenium. However, this is high experimental and not recommended.*

**Correct answer(s):**

1. Clipboard Format

Question

You want to make changes to reduce the testing and test case writing time. Which testable design patterns should you implement?

**Options:**

1. Give each element a unique Name or ID
2. Externalize HTML code that is produced so it can be tested once rather than on each page it is used
3. Add comments with unique names for easy identification
4. Stop using CSS so that XPATH queries will not be necessary in the test scripts

Answer

***Option 1:****Correct. Giving each element a unique name makes it easier for Selenium to identify elements and reduces the need for complex XPATH or DOM queries that can easily fail in the future as items are added to pages.*

***Option 2:****Correct. Externalizing pages, data, or even sections of pages, like menus, means those sections and data only need to be tested once rather than on each page it may appear.*

***Option 3:****Incorrect. Although adding comments is always a best practice, it will not help much with writing test cases, and, giving comments a unique name will not provide much benefit.*

***Option 4:****Incorrect. CSS is a fundamental requirement for web development and testing will have to work with it, even if there are times when complex queries may be required.*

**Correct answer(s):**

1. Give each element a unique Name or ID  
2. Externalize HTML code that is produced so it can be tested once rather than on each page it is used

Question

You have decided to use a Java interface to run your Selenium tests. What best practices should you implement?

**Options:**

1. Use methods and classes to reduce the amount of code required by reusing them in multiple tests
2. Use a database to store validation and input data for the tests
3. Use Java arrays within the source code for the input validation data
4. Use a single Java source file for each test case and do not reuse code to prevent the tests from interfering with each other

Answer

***Option 1:****Correct. The principle of DRY (Don't Repeat Yourself) should be used to minimize code and have functions that can handle multiple situations for multiple tests.*

***Option 2:****Correct. A database, or an external file, should be used to store validation and input data so that test scripts do not have to be recompiled, or require a software developer, in order to run.*

***Option 3:****Incorrect. Although Java arrays can be used to store the input validation data, it means even small changes will require recompiling the test script, versus using something like an external file where only a text editor is required.*

***Option 4:****Incorrect. Code reuse and using common functions between many test cases should always be a priority. Even a simple function that can "login" a user to a web site can be used by many different test cases.*

**Correct answer(s):**

1. Use methods and classes to reduce the amount of code required by reusing them in multiple tests  
2. Use a database to store validation and input data for the tests

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a02_it_enus/output/html/course_transcript.html#top)

© 2017 Skillsoft Ireland Limited