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

**Course Transcript**

Selenium Automated Web Testing Fundamentals

**Getting Started with Selenium**

[1. Course Introduction](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t2)

[2. The Selenium IDE](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t6)

[3. Assert versus Verify](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t10)

**Test Cases, Test Suites and IDE Controls**

[1. Basic Test Cases](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t15)

[2. Test Suites](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t19)

[3. Selenium IDE Controls](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t23)

**Locators**

[1. XPath Locators](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t28)

[2. DOM Locators](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t32)

[3. CSS Selectors](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t36)

[4. Locators in the IDE](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t40)

**Patterns and Variables**

[1. Pattern Matching](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t50)

[2. Using Variables](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t54)

**Debugging and Running Tests**

[1. Debugging](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t59)

[2. Using Any Browser](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t63)

**Testing Page Elements**

[1. Input Controls](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t68)

[2. List Boxes](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t72)

[3. Checkboxes](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t76)

[4. Checking Popups](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t80)

[5. Multiple Windows](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t84)

[6. Drag and Drop](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t88)

[7. Clicking Links](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t92)

**Practice: Basic Automated Web Testing using Selenium**

[1. Exercise: Basic Automated Web Testing using Selenium](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#t97)

Course Introduction

Learning Objective

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

* *start the course*

**1. Introduction to the course**

Selenium provides an automated testing framework for testing web sites. It gives us options for creating tests in many programming languages as well as an interactive development environment as part of Firefox. I'm Tony Lowe, a Java architect with over 15 years experience in building and teaching IT solutions. This course will focus on getting started in Selenium using the IDE, though many of the concepts are common across all of the Selenium implementations choices. Through this course, we'll see how we can create test cases, find elements on a page, and manage user interactions in a few examples of tests.

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

The Selenium IDE

Learning Objective

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

* *recognize features of the Selenium IDE*

**1. Installing and using Selenium IDE**

The Selenium IDE is the easiest way to get started in automated web site testing, as you don't even need any programming experience to create your first test. The goal of the IDE is to watch you interact with a browser and capture those steps for later playback. There is, of course, more to it than that. But this is a solid start, so let's take a look. So, I don't even have the Selenium IDE installed in this browser yet, I'm in Firefox. That's one requirement of Selenium and the IDE is you have to use the Firefox browser. The plug-in only works in Firefox. And so I am in docs.seleniumhq.org/download/. You can actually search on Download Selenium it'll get you to this page. And you can see there's a **Selenium IDE** download and here is the link for it right here. It's not super obvious, but there's a link for it right there. If you look down at the bottom-left here, I can't move the mouse to point at it because it's going to disappear. But it's going to show up down here where the mouse is. It says selenium-ide/2.5.0.xpi. XPI is a plug-in format for Firefox. So, if you do it on Chrome, if you do it on IE, it won't do anything, it'll download a file you can't use. So, you have to do it in Firefox. Alright, so I'm going to click on this and it's going to say, "Do you want to install this?" And I'll say yes, **Allow**. And poof, just like that, it downloaded everything. And then here, it has several things it's going to install. These are all related to the IDE. These are formatters for source code, if you want the coded options later on. But here is the core IDE. I can say **Install Now**, and it took that quick. That was real-time there, it took less than a second right there. I can restart my browser, and there you go. Now it's back, and I have this little button up in the top-right corner called **Selenium IDE**.   
*The Firefox web browser is open on the docs.seleniumh.org/download web page. The page includes a Selenium IDE section with a 2.5.0 link for downloading the latest version of the software, as well as Release Notes and plug-in installation links.  
  
These links are embedded in the following sentence in the second paragraph of the Selenium IDE section: "Download latest released version 2.5.0 released on 01/Jan/2014 or view the Release Notes and then install some plugins." Active links in this text are "2.5.0," "Release Notes," and "install some plugins."  
  
When the presenter mouses over the 2.5.0 link, the full path to the latest version – release.seleniumhq.org/selenium-ide/2.5.0.xpi – appears in a box at the bottom left corner of the interface.  
The presenter clicks the 2.5.0 link and a dialog box with an Allow button asks if the presenter wants to install the software. The presenter clicks Allow and a download progress bar displays, followed by a Software Installation dialog box. The dialog box includes a list of five add-ons related to the Selenium IDE - namely, Ruby Formatters, Selenium IDE (the core IDE), Python Formatters, C# Formatters, and Java Formatters. The dialog box also includes an Install Now button, and a Cancel button. The presenter clicks Install Now.  
  
A dialog box with a Restart Now button specifies that five add-ons will be installed after Firefox is restarted. The presenter clicks Restart Now and Firefox closes and then opens again on the same docs.selenium.org/download web page. A Selenium IDE button now appears in the right-hand side of the address bar.*   
  
So, when I click on this, it opens it up and I have my control panel here. We'll learn more about this control panel as we go. The only thing you need to know to get started with is this little button up here in the top-right, the **Recording** button. You can see since I started it up, it's now recording. It's going to track everything I do over here on my web site. Now, I'm going to go to a very popular search site inside of there, and then I'm going to search on selenium ide, alright. And I'm going to hit **Enter** right there. And as I go back to my page, you can see it's opening the page /. Now, I didn't open /, I opened google.com. Well, the **Base URL** shows up here. This is where it's going to start. And if you're going to start somewhere underneath that, this'll show you where. But since I'm starting at the **Base URL** and I'm typing something in, it shows up just like that. Now, inside of here it shows I've typed into a box. I typed the word selenium ide. We can see that showing up over here. And as I did that, it's put it in the box that has this id. Now, this id is something that's on the web site. It's over here, I can actually look inside of here in the source code. And you can see I have an input box that's called gbqfq.   
*The presenter clicks the Selenium IDE button..  
  
The Selenium IDE opens. It contains three main panes – a Test Case pane, a pane with Table and Source tabbed pages, and a pane with Log, Reference, UI-Element, and Rollup tabs. It includes a menu bar with File, Edit, Actions, Options, and Help options, a Base URL text box that's set by default to www.google.com, and a toolbar with a Playback Speed slider and Play entire test suite, Play current test case, Pause/Play, Step, and Record buttons. The Table tabbed page contains a table with Command, Target, and Value columns, as well as Command, Target, and Value text boxes.  
  
The presenter clicks the Record button on the Selenium IDE toolbar and then switches to Firefox. He enters www.google.com in the Firefox address bar to open the Google search engine, types selenium ide in the Search text box, and presses Enter. The Google search results display.  
  
The presenter returns to the Selenium IDE control panel. The table on the Table tabbed page now contains entries in two rows. The first row lists the command open and the target /. The second row lists the command type, the target id=gbqfq, and the value selenium ide.  
  
The presenter highlights the command open and the target / in the IDE control panel. He then highlights the base URL - https://www.google.com/ - in  the Firefox browser. He indicates that what he's typed in the browser shows up in the Selenium IDE as the command open and target / and the command type, the target id=gbqfq, and the value selenium ide.  
  
The presenter then highlights the command type, the target id=gbqfq, and the value selenium ide, He points to selenium ide (the value) and type (the command) to explain the value was typed into a box in the browser.   
  
The presenter highlights the target id=gbqfq in the control panel and then returns to the browser, where he right-clicks to open a shortcut menu, and selects the Inspect Element option. A pane opens at the bottom of the Firefox interface. The pane contains a number of tabs, including Inspector, Debugger, and Style Editor. The Inspector tabbed page is open. It contains the page source code.  
  
The presenter highlights "gbqfq" in the following tag in the source code:  
  
     <input id="gbqfq" class="gbqfq" type="text" value="" autocomplete="off" name="q" style="border":  
     medium none; padding: 0px; margin: 0px; height: aut...tion: absolute; z-index: 6; left: 0px  
     outline: medium none;" dir="ltr" spellcheck="false"<>/input>*  
  
I don't know what that means. Somebody at Google coded that up, and that's where we got that inside of there. Actually, I didn't mean to keep recording that. Let me take these steps out real quick. And by the way I just hit **Delete**, took those steps out. So if your recording doesn't go the way you want it to, you do just what I just did, you can just delete those steps. But you can see now I've done, I've typed this in. Now, that's just executing the steps inside of here, that does not get me to checking what's on here. If I want to make a test I have to check what's on here. And so, what I want to do is maybe select some over text here and right-click. And you can see there's a lot of different options inside of here. I can assert that the text is there. I can open something and source something, we don't know what those mean yet. But the assert is interesting. Or I can go look for all these different things that are going on inside of here. So, I could say "Let's verify that there's some text inside of here." Or I can say, "Let's assert some text inside of here." I can assert the title page of this guy. Whatever I need to do. So, I can do again, let's verify the title, and then let's right-click again, and it'll go to the All Commands. Let's go to verify some text of inside of there. And as I go back to my Selenium IDE window, you see I have some checks here.   
*The presenter closes the pane and switches back to the Selenium IDE, where two additional steps have been recorded and added to the table in the Table tabbed page. He removes those steps. He then highlights the Command, Target, and Value entries in the table.  
  
The presenter returns to Firefox, selects the text Selenium IDE from the first search result, right-clicks to open a shortcut menu, where he points to different options, including assertText css = em Selenium IDE, open/#q = selenium + ide, and storeText css = em Selenium IDE.. He then selects Show All Available Commands. This opens a submenu that contains options such as assertElementPresent css=em, verifyText css=Selenium IDE, and verifyTitle selenium ide – Google Search. The presenter selects the verifyTitle selenium ide – Google Search option. He then reopens the shortcut menu and selects Show All Available Commands - verifyText css=em Selenium IDE.  
  
The presenter returns to the Selenium IDE, where three new rows have been added to the table in the Table tabbed page. He removes the first new row - with the command selectWindow and the target null. One of the remaining new rows lists the command verifyTitle and the target selenium ide – Google Search. The other row lists the command verifyText, the target css=em, and the value Selenium IDE.*  
  
So, after the page is loaded, sorry I need to take that step out again. After the page is loaded, I want to verify that the title of the page, which shows up here in the tab bar, says selenium ide – Google Search. Because that should match what came up here, so we'll see if that continues to work. And then verify text here. The inside of this em box inside of there, I have the word Selenium IDE somewhere, inside of there. Now, this is our test. Let's execute our test. The way we can do that is this button up here, **Play current text** case. It starts it over again. And some of the stuff worked and some of the stuff didn't work inside of there. Okay, the reason behind it not working actually, is because of all these Ajax feature. And because this guy goes so quickly inside of here, this is going to fail. Because this is not actually refreshing the page and it goes to verify this stuff right away. Now, if I slow my test way down. This is the other thing I can show you, the control button, slow way down. I can open the page, I go and okay, now this one failed because it went too quick, but then this one worked. Now, if I go and double-click on it here, you can see the page title is now there. I can actually go and execute any one of these lines individually as well as running it. There's the page title that finally shows up and goes from there. Now, there's ways to fix that, but that's well beyond the scope of what we're doing here. Just as a little teaser, if you keep looking in, you'll find out different ways to make the sites wait and all sorts of great things. There's so much more you can do with this Selenium IDE browser and tool. And now you're started – now you have it installed and now you can go make those things happen.   
*The presenter highlights the title in the page tab in Firefox and then highlights the corresponding entry in the control panel in the Selenium IDE. He then highlights the row in the control panel that lists the command verifyText, the target css=em, and the value Selenium IDE.   
  
The presenter clicks the Play current test case button on the toolbar in the Selenium IDE. The recorded steps automatically run in the Firefox window, beginning with the opening of the Google search  and ending with the text being verified. When the test completes, the first two recorded steps are highlighted in green in the Selenium IDE and the last two steps are highlighted in red.  
  
The Log tab at the bottom of the IDE lists the steps that were executed. Steps that failed are preceded by [error] and are listed in red.  
  
The presenter then drags the Playback Speed slider on the toolbar in the Selenium IDE to the slowest setting and then clicks the Play current test case button again. The recorded steps are repeated in the Firefox browser. At the end of the test, the first two steps and the last step are highlighted in green to show that they succeeded, and the third step is highlighted in red to show that it failed.  
  
The presenter double-clicks the third step. The step executes successfully and is highlighted in green. The presenter then double-clicks the first step, which lists the command open, and the Google interface opens in Firefox. The presenter clicks the second step, which lists the command type, and the text selenium ide is automatically entered in the search text box in Firefox. The presenter double-clicks the remaining two steps and the page title and text are verified.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Assert versus Verify

Learning Objective

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

* *choose between Selenium assert and verify commands*

**1. Asserting and verifying test cases**

Within Selenium IDE, you may have seen that there are two sets of commands that are awfully similar, Assert and Verify. Each has many options and indeed mirror each other with one subtle difference, how they are intended to behave when they fail a check within a test. Understanding the difference and when to use each of these is important, so let's take a look at each of these in action. So, we have an interesting site here, we have some validation that we want to test. So, we are going to record a test here and as we go in, we will just hit the save button. And when we do that, we need to validate a bunch of text on this occurs. So, we need to say hey, let's make sure that this main message that occurs, let's Assert that and let's - each of these things, the title of the events required so we better Assert that, that happens. So, let's you know Assert that and a location is required, so we better Assert if that's the case. And you know so we can say, this is really easy to build that, I am just going to go through Assert, Assert, Assert and the description again is required and I can say Assert on any of those. So, there is an easy test, so when I hit the button I should Assert on all those and if I run the test again perfect, it runs great. Now, this is the Assert, what happens on the Assert then is that the test doesn't quite run as it is expected. Let's say that the text I just saved right there wasn't indeed right, I look at the requirements that I can see, well maybe it wasn't – maybe it was supposed to say Please correct the following errors…, maybe this is what the requirements says is supposed to be and our job is to be particular. We need it to say exactly that   
*The Selenium IDE is open and, in a window behind the IDE, a web page with the title Add a New Gathering is open in Firefox.   
  
The presenter clicks the Record button in the Selenium IDE. He then switches to the web page.  
  
The Firefox web page includes a Title of meeting text box, a Date of Event text box with a calendar button, and Starting time and Ending time text boxes with associated drop-down lists for choosing PM or AM. It contains Location and Description text boxes, and Activate Date and Inactive Date text boxes, each with calendar button. The page also contains a Make this Recurring checkbox and Save and Cancel buttons.  
  
On the web page, the presenter clicks Save. An error message below the page title says "Please correct the following errors." The presenter highlights this as the main message that occurs on the page. He also highlights the other error messages on the page: The message "Title of Event is required" appears below the first section, the message "Location is required" appears below the Location text box, and "Description is required" appears below the Description text box.  
  
The presenter selects and then right-clicks the message "Please correct the following errors." A shortcut menu opens and he selects the option assertText css=div.errors Please correct the following errors.  
  
The presenter then selects and right-clicks the text Title of Event is required, and selects assertText css=div.errorMessage Title of Event is required from the shortcut menu.  
  
The presenter selects and right-clicks the text Location is required, and selects assertText css=div.formRow div.errorMessage Location required from the shortcut menu.  
  
Finally the presenter selects and right-clicks the text Description is required, and selects assertText //div[@id='content']/form/div[6]/div Description is required from the shortcut menu.  
  
The presenter switches to the Selenium IDE. On the Table tabbed page, the table contains six rows. Row 1 lists the command open and the target /website/addGathering.vp, Row 2 lists the command clickAndWait and the target name=buttonPressed, Row 3 lists the command assertText, the target css=div.errors, and the value Please correct the following, Row 4 lists the command assertText, the target css=div.errorMessage, and the value title is required, Row 5 lists the command assertText, the target css=div.formRow > div.er, and the value Location is required, and Row 6 lists the command assertText, the target [@id='content']for, and the value Description is required.  
  
The presenter clicks the Play current test case button in the Selenium IDE and the steps execute on the web page. All the rows in the IDE table are highlighted in green to indicate that the test steps were successful.  
  
In the table, the presenter selects the row that lists the command assertText, the target css=div.errors, and the value Please correct the following errors. In the Value text box, which contains the value for the row, the presenter adds an ellipsis after the text Please correct the following errors. He then clicks the Play current test case button.*   
  
so when I run, well I didn't test the rest of this.I stopped at the first one, so let's say that many of these guys were off, so the title of the event is required, maybe it's not supposed to have a period there and the location is required. The location maybe it should be a location of the event is required. So, as I am doing the requirements I shouldn't just trust the page on the text, I should be looking at this inside of here and then Description is required. So, let's just say this should be Event Description is required. So, inside of this, I can go through, run the test and fail and the developer can come by and say oh well look, when I ran the test, this failed. And I can go back and I fix that, I test it locally, I do a build, I send it to the server, it gets deployed. The next day, it comes back and then I would run that one and guess what, the next error would be found. We go through the cycle again, the next day I send it off, fix that, go back and come back, the next one will be found. And that's highly undesirable, I don't want my testing cycle to go over many days for things that are all should be checked right there. Now, when I did my check inside of here, notice I right clicked and it said Assert, because that's the one that pops up right away. The proper choice inside of here should have been go to all the commands here and then pick the Verify inside of here. So, I can do that just as easily by checking it on this side, I'm going over here and change assert to be verifyText.   
*The Log tabbed page in the Selenium IDE shows the following error message: "Actual value 'Please correct the following errors' did not match 'Please correct the following errors...' In the table in the Selenium IDE, the presenter selects the row that lists the command assertText, the target css=div.errorMessage, and the value Title of Event is required. In the Value text box, the presenter removes the period from the end of the text Title of Event is required.  
  
Next the presenter selects the row that lists the command assertText, the target css=div.formRow > div.ErrorMessage, and the value Location is required. In the Value text box, he changes the text to Location of the Event is required.  
  
The presenter selects the final row, which lists the command assertText, the target //div[@id='content']/form/div, and the value Description is required. In the Value text box, he changes the text to Event Description is required.  
  
The presenter clicks the Play current test case button and the first step with the command assertText is highlighted in red to show that an error occurred there.  
  
The presenter switches to the Firefox interface, and selects and right-clicks the message Title of Event is required. In the shortcut menu that opens, he selects Show All Available Commands and, in the submenu, points out the option verifyText css=div.errorMessage title of Event is required.  
  
The presenter returns to the Selenium IDE and selects the first row that contains the command assertText. In the Command text box, he replaces assertText with verifyText.*  
  
So, now by changing this to be Verify, when I run the test again, you can see it fails this one but it continues on and checks the next one. Well again if I do a Verify on this one, it keeps going down. Both of those have failed now and let's just change all of this to Verify, let's change these from Assert to Verify. So, now as I do my verification across all these we can see my test case does indeed fail, but I'm getting all the failures on the test case. Each element is failing because within this, we are not caring about stopping the test case on the first failure. That's when what Assert does when I say Assert, I'm saying that if this isn't true there is no point in continuing testing. The state doesn't make any sense and there is nothing else to check, as opposed to Verify which is just saying hey, our test is running along check this thing and if it is wrong then we will fail the test case but let's keep going. And that's the key behavior between the two of these. The Verify is saying to me I have many things to check and in this case the validation, I want to check all the validation rules. And that gives us more information out of our test, it tells us everything that's wrong within our test and we have to be very careful in choosing which one we are going to use. If indeed the first one failing is going to cause other ones to fail I should do an Assert in the first one. But outside of that using Verify really does help us to get the most out of our test cases and give the most information back as it goes and be able to tell all the test failures to the development team or whoever is going to fix these all at once.   
*The presenter clicks the Play current test case button. The test runs but encounters errors. The row that lists the command verifyText, as well as the first of the rows listing the command assertText, are highlighted in red in the main pane of the Selenium IDE.  
  
The presenter selects the first row with the command assertText. In the Command text box, he changes assertText to verifyText. He then clicks the Play current test case button. The test runs. Both rows with the commands verifyText, as well as the first row with the command assertText, are highlighted in red.  
  
The presenter changes the command assertText to verifyText in the remaining rows. He then runs the test again. All four of the rows are highlighted in red.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Basic Test Cases

Learning Objective

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

* *create basic Selenium test cases*

**1. Creating and running basic test cases**

The Selenium IDE allows us to create test cases by recording our interactions on a web site within the Firefox browser. This tool is a quick and easy way to at least get started if not be able to realize the entire test case by doing the manual testing would be doing anyway and capturing the results. Let's build a test and see this tool in action. So, we have our page that we're going to start with on our testing and we have our IDE up and ready to go. We just want to make sure that our recording button is clicked and as long as this is clicked and says now recording, ready to go. Then I can just start testing as I normally would. So, I click my login because that's what I want to test and I type in my Username and I type in my Password and I hit **Login**. Now, if I come back over here you can see it captured. I started it on the home page. I clicked the login button. I typed in the username. I typed in the password and then I hit the submit button and I've come here. Now, I would need to validate that this is indeed the page expected to be on and the way I can do that is I can right click here and then I go down to one of the assert options inside of here. I can say assert that this text is actually on there. And if I go back to my page you can see I'm asserting inside of this h2 block I have the administrator text that's showing up inside of my test case here and on my web site. So, pretty much I'm ready to go. I can go through and I can Save this Test now if I want to do it don't have to save it right now I could save it later, but this could be myLogin test right here. And as you can see now I have given it a name and every time I save it it's going to be saved as an HTML file inside of my folder there. So, let's go ahead and run the test again and our test broke.   
*The Selenium IDE 2.5.0 is open.The Table tabbed page displays no steps under the Command, Target, and Value headings in the table. The Firefox interface is open in the background. The presenter clicks the Record button in the Selenium IDE taskbar.  
  
He then switches to the Firefox web page. The web page contains a  menu bar with the options  home, news, calendar, contact us, login, and no popups. There is another menu bar with three options: Menu Bar Text, Another Menu Bar Item, and Menu Bar Option. The web page also includes three panes labeled No Active Polls,, Upcoming activities, and News.  
  
The presenter clicks the login option in the top menu bar. The login page appears. It contains Username and Password text boxes, and a Login button. The presenter types admin into the Username text box and types a password into the Password text box. Then he clicks the Login button.  The Administrator Home page opens.   
  
The presenter switches to the Selenium IDE 2.5.0  where he highlights five rows that have been added to the table in the Table tabbed page as a result of his activity on the web page. Row 1 lists the command open and the target /website/home.vpd, Row 2 lists the command clickAndWait and the target link=login, Row 3 lists the command type and the target id=usernameInput, Row 4 lists the command type, the target id=passwordInput, and the value test, Row 5 lists the command clickAndWait and the target name=submit.   
  
The presenter switches to Firefox to validate the web page. He highlights the Administrator Home text, opens the shortcut menu, and then selects assertText css=h2 Administrator Home from the menu. He goes back to the IDE, where a step has been added to the Table tabbed page. The added step has the command assertText, the target css=h2, and the value Administrator Home.  
  
The presenter opens the File menu in the control bar and selects Save Test Case and the Save test case Untitled as… dialog box opens. The box is navigated to a location on the hard drive and contains a File name text box, as well as a Save and a Cancel button. He saves the test case as myLogin. The name myLogin now appears in the Test Case panel of the Selenium IDE. The presenter highlights this folder.   
  
The presenter then clicks the Play current selection button, the test runs and ends with the clickandWait step highlighted red.*   
  
It says down here in our log that the login was not found. So, it went and opened up the page it says but you could see our login page is not here because we still have our site thinking we're logged in. We never logged out. And so, as far as this web site is concerned, it's we're still logged in. And so there is no login button. That's okay. I can fix my test fairly easily. So, after I validate the administrator's home and logged in, I can come down here, click on this next step, go hit **Record** again. And then I can go click on the **logout** button and record that step to say hey logout afterwards.Alright, so now I'm going to save my test case again. **Control+S** save that right there, and then I can run the test case again. And now we run into some timing issues. This happens occasionally. Let's go, let's just slow down our test for a second and we'll let this run over again. And there everything worked exactly as expected. So, all the steps passed, my test case passed again. And if I run it again from scratch, it all seems to be working just fine. So, there we go. Now, my test case is up and running. Now, my test cases isn't necessarily done. I have only tested the happy path test case inside of here. I haven't done anything to test any failures. So, how do I do that? Well I can come back to my admin tool, and I can insert steps anywhere I want to here or I can add them afterwards. So, maybe what I want to do is test the happy path and then hit the record inside of here. And so after I've tested the happy path, after I logged out, I end up on this page so I could just hit the Login button alone.   
*He highlights the Log tab at the bottom of the IDE, which contains the error massage  
  
[error] Element link=login not found  
  
The presenter switches to Firefox and, on the Administrator Home page, and highlights the logout option.   
  
He then goes back to the Selenium IDE and selects a new, empty row in the table and clicks the Record button. He returns to Firefox and clicks the logout option again.   
  
As a result, a new row is added to the IDE table. It lists the command clickAndWait and the target link=logout.  
  
The presenter saves the test case by pressing Control + S. He then runs the test case again by pressing the Play button in the IDE. The test case fails, with a red highlight on the entry in the table and an error in the Log that reads [error] Element id=usernameInput not found.  
  
Because of timing issues, the presenter drags the Playback Speed slider to a position half-way between Fast and Slow and then clicks the Play current test case button again. The test runs successfully. The entries in the Selenium IDE table for the commands, targets, and values, which represent the steps recorded on the web page, are highlighted one by one in green to indicate they are successful. The presenter clicks Play current test case again, and the test executes successfully for a second time.  
  
The presenter highlights the steps in the Selenium IDE as he talks about testing failures and inserting steps.  
  
The presenter clicks the Record button and then switches to Firefox, where the login page is open. The Username and Password text boxes are blank.  
  
The presenter clicks the Login button and a message at the top of the interface states "Username is required" and "Password is required."*   
  
And so, again, you can see I've added since I hit **Record** again and another step to login. And now I want to validate hey the username is required. That should show up here and so I want to assert to that text is there and the password is required. I want to assert to that text is there. So, I have two assert statements I'm adding inside of here. That's the username and password are required at this point. That's why I'm testing and making sure those were happening and I can add on to this test all I want to. I could test them individually. So if I do like this, I can type that in here. You could see I have added the text and then I can validate that it still says password is required. And then I can go through and test the login properly and then validate one more time that this is here. So, we see again this is here right there and then I want to log out again just to make sure I end up back in the same spot. So, I'm going to **Save** my test case again. I can **Run** my test case again and see if it will go through all these steps. So, it logs in just fine and the happy path and it's validating, yeah, but looks like that's working, yeah. It looks like that's working. Alright so now I have a test the in theory fully test my login feature. Now, you might choose to do your test like this everything in one big flow. You could also choose to use many tests. There is nothing to wrong to have a login test suite and have one test that's a happy path or one test its failed test and things like that. That's going to be up to you in test design, but using the IDE you see we can record our steps and very easily capture a full test case and even do a little bit of debugging and make sure everything works within our tool.   
 *The presenter switches to the Selenium IDE to show the latest step that has been added to the table. The added step lists the command clickAndWait and the target name=submit.  
  
The presenter switches back to Firefox. He selects and right-clicks the text "Username is required" text. In the shortcut menu that opens, he selects the option assertText css=li Username is required.  
  
Next the presenter selects and right-clicks the message "Password is required," and selects the option assertText //div[@id='content']/div/ul/li[2] Password from the shortcut menu.  
  
The presenter returns to the Selenium IDE. Two more steps have been added to the test case. The first added step lists the command assertText, the target css=li, and the value Username is required. The second added row lists the command assertText, the target //div[@id='content']/form/div/ul, and the value password is required.  
  
To test the steps individually, the presenter switches to Firefox, types a username in the Username text box, leaves the Password text box blank, and clicks Login. The message "Password is required" appears on the web page.  
  
He switches back to the Selenium IDE to show the two added steps in the table. The first added step list the command type, the target id=usernameInput, and the value admin and the second added  row lists the command clickAndWait and the target name=submit.  
  
On the web page, the presenter selects and  right-clicks the message "Password is required" to validate it. He then selects the option assertTextcss=li Password is required from the shortcut menu. This action then appears in the Selenium IDE in the table as the command assertText, the target css=li, and the value Password is required.  
  
On the web page, he types a password in the Password text box and clicks Login, and the Administrator Home page opens.  
  
In the Selenium IDE, the action of typing the password appears as the command type, the target id=passwordinput, and the value test. The command clickAndWait and the target name=submit are also added to the IDE table.  
  
Going back to Firefox, the presenter selects and right-clicks the title Administrator Home, and selects the option assertText css=h2 Administrator Home from the shortcut menu.  
  
This appears in the Selenium IDE as the command assertText, the target css=h2, and the value Administrator Home.  
  
On the web page, the presenter clicks logout. He then returns to the Selenium IDE, where the command clickAndWait and the target link-logout now appear as the result of him clicking logout on the web page.   
The presenter saves the test case again and clicks the Play current test case button. The test runs successfully, leaving each row in the table highlighted in green. And each step is carried out on the web page simultaneously.  
  
The presenter highlights the list of steps in the Selenium IDE.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Test Suites

Learning Objective

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

* *create Selenium test suites*

**1. Creating and running test suites**

As most of the interesting features and functionalities on a web site require many, many tests to fully validate, Selenium IDE provides us the ability to group and organize test cases as a test suite. The suite allows for us to plan and execute tests as a group and then manage them both as a reusable asset in development, and execution. So, if we start with a brand new Selenium instance, we can set up several tests and group them together, without having to execute everything as one huge long flow. So in this case we will go ahead and start up our recording, and let's record a test case to check out the login. Now the very first test case we want to do is, just simply, some basic validation. So we put in nothing. So, we clicked it and we put nothing in here and then we'll validate. Yes, we have an error name, error for the username. And then we also have an error for the password. Alright, and so there's the checks right inside of there. Now let's put a username inside of here, alright, and we submit it, and we got the next error. We would've validated that the password is required. Okay, we've put that inside of there. And then let's clear out the username and let's put the password only. And put that inside of there and then we can see, in this last case, that we have a basic test case inside of here. So, let's go and **Save** our test case real quick. And so we save this test case, and we're going to name this test case whatever we want to name this test case. And so in this case we can do the inputValidationErrors test case.   
*The Selenium IDE is open and the Test Case pane contains the entry Untitled. In a window behind the IDE, a home page is open in the Firefox browser.  
  
The web page contains a menu bar with home, news, calendar, contact us, login, and no popup options, and a  side menu bar with three options – Menu Bar Text, Another Menu Bar Item, and Menu Bar Option.. It also contains three panes – No Active Poll, Upcoming Activities, and News.  
  
In the Selenium IDE, the presenter clicks the Record button. He switches to the web page and clicks the login button, and a login page opens. It contains Username and Password text boxes, and a Login button. The presenter leaves the text boxes blank and clicks the Login button. The messages Username is required and Password is required display at the top of the web page.  
  
The presenter navigates to the Selenium IDE, where the table on the Table tabbed page lists the three steps he has performed in three rows. Row 1 lists the command open and the target /website/home.vpd, Row 2 lists the command clickAndwait and the target link=login, Row 3 lists the command clickAndWait and the target name=submit.   
  
The presenter returns to the login page. He selects and right-clicks the text Username is required, and then selects the option assertText css=li Username is required from the shortcut menu.  
  
The presenter selects and right-clicks the text Password is required, and selects the option assertText //div[@id='content']/form/div/ul/li[2] Password from the shortcut menu.  
  
In the Selenium IDE, two new rows have been added to the table on the Table tabbed page as a result of the presenter's actions. The first added step lists the command assertText, the target css=li, and the value Username is required, the second added row lists the command assertText, the target //div[@id='content']form/div/ul, and the value Password is required.   
  
The presenter returns to the web page. He types admin in the Username text box and clicks Login. The message Password is required appears. He points to the new row that has been added in the Selenium IDE.The row lists the command clickAndWait and the target name=submit.  
  
On the web page, the presenter selects and right-clicks the text Password is required, and selects assertText css:li Password is required from the shortcut menu.  
  
The presenter clears the Username text box, types a password in the Password text box, and clicks the Login button. The message Username is required appears.  
  
The presenter switches to the Selenium IDE to show that  steps have been captured.   
  
On the web page, the presenter selects and right-clicks the text Username is required, and selects assertText css=li Username from the shortcut menu.  
  
The presenter returns to the Selenium IDE, where six new rows have been added to the table. The first added step lists the command assertText, the target css=li, and the value Password is required, the second added step lists the command click and the target css=div.selectedFormRow > label, the third added step lists the command type and the target id=usernameInput, the fourth added step lists the command type, the target id=PasswordInput, and the value target, the fifth added row lists the command clickAndWait and the target name=submit, the sixth step lists the command assertText, the target css=li, and the value Username is required.  
  
In the Selenium IDE, the presenter selects File - Save Test Case. In the Save as dialog box, he accepts the current folder location, types inputValidationErrors in the File name text box, and presses Enter. In the Selenium IDE, the Test Case pane now lists the test case inputValidationErrors.*   
  
Alright, so we have that test case that's stored here. So, let's go through and create a second test case then. Now notice, it keeps this test case around and we're adding in another one here. So, the first test case we were only looking at some basic validators, just the stuff – is it required or not? So let's check further into a more interesting test case here. So we'll come back and we'll start over from scratch. We'll go back to the homepage because our test design, we want to have each test be isolated on its own. We may or may not run all of them together. And so we could have a test suite that assumes one comes before the next, but let's go ahead and start this one from scratch. We're going to design it so it's a bit more reusable individually. So I'm going to click to record again. So, inside of here, I'm going to go to my login page. Now I'm going to put in admin and this is junkpassword. Okay, so it's definitely not the password to login and you can see I have an Invalid Username/Password inside of here. So I'm going to assert that text is valid. Alright, so now again, we can see you put a non-password inside there and we get the proper error inside of here. So, let's try a different combination, let's do wrongadmin and then we'll say, a password that might be viable inside of there, and again we get the exact same error message inside of here. Which, according to our design, we shouldn't be able to figure out that it's not a valid username based off the messages coming inside of here. And so now we have another test case, and again we can **Save** this other test case. Now we're really not…   
*The presenter clicks New Test Case in the File menu. The new Untitled test case opens in the Table tab under the existing test case called inputValidation errors. To start the new test case, the presenter navigates to Firefox clicks the home option in the top menu bar and then goes back to the Selenium IDE where he clicks the  Record button. He then returns to Firefox.   
  
In Firefox, the presenter clicks the login option in the top menu bar. On the login page, the presenter types login details and clicks the Login button. The text Invalid Username/Password, Try Again appears above the Username when the login fails. The presenter selects the text, right-clicks, and selects the assertText css=li Invalid Username/Password, Try again option in the shortcut menu and then navigates back to the Selenium IDE where the six executed steps have been captured by Selenium IDE.   
  
The presenter highlights the junkpassword entry along with the error entry on the Table tabbed page of the Selenium IDE. The six new entries shown are as follows: Row 1 lists the command open and the target /website/home.vpd, Row 2 lists the command clickAndWait and the target link=login, Row 3 lists the command type, the target id=usernameinput, and the value input, Row 3 lists the command type, the target id=usernameinput, and the target admin, Row 4 lists the command type, the target id=passwordInput, and the value junkpassword, Row 5 lists the command clickAndWait and the target name=submit, Row 6 lists the command assertText, the target css=li, and the value Invalid Username.  
  
The presenter navigates back to Firefox where he removes the text in the Password text box, and then types "wrongadmin" into the Username text box, and a password in the Password text box before clicking the Login button. The presenter selects  the error message above the Username text box and asserts it before navigating back to the Selenium IDE. A step has been added which lists the command assertText, the target css=li, and the value Invalid Username.  
  
The presenter clicks Save Test Case from the File menu in the IDE and then saves the test case as validateWrongInputs.   
The presenter clicks File in the control bar, and clicks Save Test Suite in the File menu. The Select Test Suite dialog box opens and is navigated to a location on the hard drive. The Select Test Suite dialog box includes a File name text box, as well as the Save and Cancel buttons. The presenter types "TotalLoginValidationSuite" into the File name text box before pushing Enter on the keyboard to save it.*   
  
Okay, so we're really not at this point saving anything as a test suite. We have them set up as a test suite, and we can actually execute them as a test suite. But we want to make sure, if we want to save this as a test suite, we need to choose to do this separately. Now when I save it as a test suite, this could be my TotalLoginValidationSuite or whatever I want to call it inside of there. And so that allows me to set this up and then load these individually or as a group. So again, I can run everything that will test all of the errors inside of here, and I'd test all the test cases inside of here. It looks like something might've come up wrong on that one, but again we can go through and see what came up here. If there's an error inside of here in the test board, we're not really here to design the test… at this point, well we can debug it later on. The point is we have this the setup. Now let me go ahead and say **New Test Suite**, because that'll clear it all out. The best part about this though, is I can go through and I can go and **Restore Test Suite** by doing the **Open** here. And so I have my TotalValidationSuite, it opens all of these. And so by using the test suite mechanism, the mechanics inside of here, I can open up individual tests or I can open up groups of them and have them all execute, and run, and tweak them together.   
*Back in the Selenium IDE, the presenter clicks the Play entire test suite button in the toolbar and Selenium IDE runs through the recorded steps, testing the test cases inside the IDE. The presenter highlights an error message in the Log pane of the IDE but lets the test case run anyway.  
  
Once the test has run, the presenter clicks the New Test Suite option in the File menu. The old test suite with two components is cleared and replaced with the new Untitled test case in the Test Case field of the Selenium IDE.   
  
To restore the old test suite, the presenter clicks the Open Test Suite option in the File menu and the Select Test Suite dialog box opens. The dialog box is navigated to a folder on the hard drive which contains previously saved Test Suites and includes a File name text box as well as the Open and Cancel buttons. The presenter double-clicks the TotalLoginValidationSuite Test suite and it opens the two test cases in the suite - inputValidationErrors and valdidateWrongInputs - in the Selenium IDE.*  
  
Again, the best tests run individually but I can also move these tests around if I need them to happen in this particular order. So, I could have a test here that does certain validations and then a test that successfully logs in. I could even have a test that logs in and then go through and add in additional tests that do something else later on. And again, I can also add in individual, yes let's **Save** that, I can add in individual test inside of here. So, like a test I had done before, created my login… sorry I didn't mean… let me open up my test suite again. And I can add in a test case inside of here, I used the wrong feature, not open, I want to right-click and say **Add Test Case** here. And I can add my login and this is the actual test case that will test the successful login. So now within my test suite, I can test all of these different combinations: bad inputs, different types of bad inputs, and then a successful login by walking through my test cases here – let me actually speed this up here. And we can bounce through this and it will in theory go through and run all of my tests, and they'll all run as a suite. And then you can either pass through the success as they go and then I could debug them as a group, and make sure everything is running as a group as well. So use test suites to be able to organize and run your Selenium tests within your ID, as a group instead of all as individuals.   
*In the Test Case field, the presenter clicks and drags the two tests to reorder them. He then clicks the Open option in the File menu and the Save? dialog box opens and asks whether or not to save the changes made to the Test Suite. The presenter clicks the Save button and the Select a File dialog box opens. He double-clicks the myLogin test and it opens in the Selenium IDE to replace the previously opened test cases in the Test Case panel.   
  
The presenter reopens the Select a File dialog box and selects the TotalLoginValidationSuite and then right-clicks in the Test Case field. The shortcut menu in the field contains five options, of which the presenter clicks Add Test Case, which opens the Select one or more test cases to add dialog box. The dialog box is navigated to a location on the hard drive, shows a number of test cases, and includes the Open and Cancel buttons. The presenter double-clicks the myLogin test case and the test case appears in the Test Case field of the Selenium IDE under the validateWrongInputs and inputValidationErrors cases.   
  
He clicks the Play entire test suite button and the three different test cases in the Test Case field Play current selection in Firefox. The steps in the IDE are highlighted in green as they are tested.  
  
To speed up the process, the presenter drags the Playback Speed slider to the extreme left where Fast is situated. The last two test cases have errors because they are highlighted red in the Test Case field.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Selenium IDE Controls

Learning Objective

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

* *work with Selenium IDE controls*

**1. Working with Selenium IDE controls**

The Selenium IDE is a Firefox extension that gives us the ability to record and formulate test cases on a web site. The tool is very basic to get started with but has advanced options and controls that really allow us to refine our test cases and manage them. Let's take a look at the IDE and the controls it gives us for testing within Selenium. So, we have a test case that we formulated here. And it's really simple to go off and save that. I can go through, I can just say **Save Test Case**, **Save Test Case As**. In this case, it was saved here and so it already has a name. I can do **Save Test Case As** and then rename it as I want to. I pull this up just to show you these are saved as an HTML document. It's something we can open and edit by hand, we don't have to use it within the IDE. But the IDE gives us this really simple interface here. For instance, let's say we don't want to type the username before the password, I can move individual items around. I can move them around one above the other. Unfortunately, it only allows to let me move these one at a time. I can't move them as a group. But I can insert statements. So again, as I'm moving these around, let's say I wanted to put an additional statement inside of there. I can insert a command and then I can choose, by hand, to pick a command or I can record a new command either way. So, I can come in, I could say I want to add something, assert something, validate something, whatever the command might be. Or I can hit the **Record** button and then it will record things at this location. So, the IDE gives us a lot of flexibility on editing our test case inside of here even within this tool. Now, past that we can, when we go through our playback, we can control the speed of this. So, right now I have the playback set very, very slow and so it really allows me to see step-by-step. And I can even **Pause** the playback – I'm sorry, let me take out this step.   
*The Selenium IDE is open and the Test Case pane lists the myLogin test case. Steps have already been recorded and are listed as rows on the Table tabbed page. Row 1 lists the command open and the target /website/home.vpd, Row 2 lists the command clickAndWait and the target link=login, Row 3 lists the command type, the target id=usernameinput, and the value admin, Row 4 lists the command type, the target id=passwordInput, and the target test, Row 5 lists the command clickAndWait and the target submit, Row 6 lists the command clickAndWait and the target link=logout, Row 7 lists the command assertText, the target css=h2, and the value Administrator Home, Row 8 lists the command clickAndWait and the target name=submit, Row 9 lists the command assertText, the target css=li, and the value Username is required, Row 10 lists the command assertText, the target //div[@id='content']form/div/ul, and the value Password is required, Row 11 lists the command type, the target id=usernameInput, and the value admin, Row 12 lists the command clickAndWait and the target name=submit.   
  
The presenter points to the test case myLogin in the Test Case pane. He then expands the File menu and points out the options Save Test Case and Save Test Case As. He points again to the myLogin test case to indicate that it was saved using the Save options. He selects Save Test Case As to open the Save as dialog box, pointing out test cases that have already been saved in the current folder location, including the myLogin test case along with a folder called "suite," and documents called "allTests," "assertExel," "dom," "login," "patterns," and "storyLifeTest." The Name of each case is shown along with information on the Date modified and the Type of file - most are listed as HTML documents but the folder called "suite" is listed as a File folder. The size of each HTML document is also given.   
  
The  Save test case as dialog box includes a File name text box. The presenter then clicks the Cancel button to close the dialog box.  
  
He returns to the Table tabbed page in the main  pane of the Selenium IDE, where he points to the rows in the table. There is row that lists the command type and the target id=passwordInput is currently below a row that lists the command type and the target id=usernameInput. The presenter drags the row with the target id=passwordinput to above the row with the target id=usernameInput.  
  
Next the presenter right-clicks the step with the target id=passwordInput. This opens a shortcut menu with options such as Insert New Command, Insert New Comment, and Clear All. The presenter selects Insert New Command and a new, blank row appears below the row that was selected.  
  
With the new row still selected, the presenter expands the Command drop-down list box, which lists available commands in alphabetical order. Examples are addLocationStrategy, addLocationStrategyAndWait, and addScript. The presenter scrolls through the available options. He clicks the Record button and points to the Table tabbed page.  
  
Next the presenter points out the Playback Speed slider, which is set to the slowest setting. He then clicks the Play current test case button and the test case steps run in Firefox.   
  
The Pause/Resume button is grayed out. To fix that, the presenter presses Delete to delete the new, empty row added to the table to show how to insert new rows. When he does this, the Pause/Resume button becomes active.*  
  
I can even **Pause** the playback at certain steps if I want to go and see what exactly is going on. So, you can see I can **Pause** it right here and say "Hey hold on, what's going on here, what step am I on? I just asserted something and then I did it – okay let me just continue and resume playback of my test." Or I can even go through and I can s, I can run the test, I can Pause it and I can go step-by-step myself. And so that gives me a lot of debug options inside of here to be able to sort out things as they go. Now, apparently I've broken my test in moving things around. Oh no, it's because I started running a test and I got it out of sequence, that's what happened here. I was logged in right there. But that's what happened there. And so now, again, I can crank this up to full speed, run the test from there. Sorry let me bring it down, there we go. And I can crank up the speed and let it run from there, and I can crank it up to a higher speed along the way as well. And again, I think I – oh yes I moved my items around here, so I broke my test there. So, let me try that one more time, you have to be careful in editing and speaking at the same time, otherwise you will end up with that test case. Alright, so that's a little bit of the editing features we can do inside of there. And we aren't here to actually make this test case work, we're here just to talk about the editing features. Again, I can insert, I can delete, I can move steps around.   
*The presenter pauses the playback at a row that contains the command assertText. He checks the web page. He then clicks the Pause/Resume button again in the IDE and the test case continues to run.  
  
The presenter clicks the Play current test case button, Pause/Resume, and Step buttons one after the other. He uses the Step button to run each step in the test case. He stops at a row that lists the command assertText because that step fails and is highlighted in red.  
  
The presenter then drags the Playback Speed slider to its fastest setting and clicks the Play entire test suite button. The test executes quickly, stopping at the step with the command type, the target id=usernameInput, and the value admin. The presenter clicks the Play entire test suite button and the test fails at the same step as before. He then moves the Playback Speed slider to roughly half-way between the Fast and Slow settings, clicks the Play current test case button, and then moves the Playback Speed slider to its fastest setting while the test is running. The test fails at a row that lists the command assertText, the target css=h2, and the value administrator Home.  
  
To fix the broken test, the presenter drags a row that lists the command clickAndWait and the target link=logout to the position below the step where the test case failed. The presenter clicks the Play current test case button, and the test fails at the step with the command type and the target id=usernameInput.  
  
The presenter moves the Playback Speed slider to a middle setting and clicks the Play entire test suite button. The test case runs successfully. He points to rows in the Table tabbed page.*   
  
Past that, there are a few other options we can talk about. We might have noticed up here this **Base URL**. The **Base URL** right now I'm testing on my local machine. These tests were all based, are based assuming my local machine. Let's say though, I wanted to move off to some dev machine. Like I knew that I have some machine in my organization called dev.myorg.com or something like that. And this is our dev machine. I can go and run these tests against a different URL and it would just basically swap out that local host. Everything else would be the same it would just swap out that local host. Or as I'm moving my test up, maybe I have a system integration test location, so that's what I call sit here, or a user acceptance test location, or I could eventually get to my real site out there. You know, who knows what this is going to be, it's going to be whatever is in the organization. But I can change around this base URL to allow my tests to run in different locations even though they're the same tests. I can edit that quite simply and run these tests in many locations instead of being just stuck here on my local machine. Even further here, I can do some additional options. There's a few other things we can look at here in underneath our **Options** menu. So, if bring up our **Options** menu, you can see I have a few other features. Now one of them that's important here is this Default timeout.   
*The presenter highlights the Base URL text box, which currently contains the URL http://localhost:8080/. He changes the address to http://dev.myorg.com/. He also then shows that the dev in the address can be replaced with sit, uat, or www. The presenter then changes the URL back to http://localhost:8080/.  
  
Next, the presenter selects Options - Options to open the Selenium IDE Options dialog box. The dialog box is open on the General tabbed page. It also includes Formats, Plugins, Locator Builders, and Drivers tabbed pages.*   
  
This is saying, "How long will the test wait for their web site before it gives up on it?" The default is 30 seconds. I changed mine to be 10 seconds just as not to be sitting around for too long. You can make it 5 seconds and you can make it 2 minutes if you have a slow action that's happening inside of there. But this is the default time by which it's going to sit around and wait. The other thing we can point to real quick is we have formatters and builders and **Plugins** and other things we can talk about this that you can use as an extension inside of here. I want to talk about the formatters for a second. The formatters inside of here is what sort of default format you want to have if you're going to extract the test cases. So, if I'm going to copy and paste these steps in the test cases, as we'll see in a second, you can actually copy them out as Ruby or Python or Java or C# test cases. So, we see here, if I select all these and I right-click and I say **Copy**. And let's say I'm going to come over and just going to paste them in some other place. I'll just put them in a Word doc for now. And I paste them, notice here it's actually spitting out Java code. This is a way I can build test cases and move them into another source. So, our Selenium IDE, as you can see, has a ton of different features that allow us to go beyond just basic formulating of test cases. Of course, we can move them, we can edit them and record them. But we can actually create other versions of test cases – our code-based ones, our Web driver-based test cases – as well as managing the execution and playback of the test within this tool as well. So, understand this tool and use it to its great effect to maximize your testing power within Selenium.   
*In the Selenium IDE Options dialog box, the General tabbed page contains various options, including a Default timeout value of recorded command in milliseconds text box, which is set to 10000 by default. The presenter highlights the Default time.  
  
The presenter briefly opens each of the other tabbed pages - Formats, Plugins, Locator Builders, and Webdriver.  He then opens the Formats tabbed page, which lets you choose a language such as HTML, Python, or Ruby and provides formatting options for that language. The presenter closes the Options dialog box.  
  
The presenter then selects a row in the table in the Selenium IDE and clicks Ctrl+A on the keyboard to select all of the rows. He then copies them into a Microsoft Word document. He highlights one of the lines in the Word document - namely, driver.get(baseUrl +"/website/home.vpd"); - to show that the original steps have been changed to Java code.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

XPath Locators

Learning Objective

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

* *use XPath as a locator strategy with Selenium commands*

**1. Finding elements with Xpath locators**

As HTML is a close relative of XML, we can take advantage of the XML searching tool Xpath to find elements inside of our web pages, to aid us in our Selenium IDE tests. Xpath provides a powerful language for searching HTML structures and can be used in all forms of Selenium testing. Let's take a look at some Xpath examples, and see how you can make your tests easier and more accurate. So for this example, we have five input boxes and we are going to be searching between them. And we can see the source here, that this is going to be what we're searching within. We'll use that here in a second. We're going to use the command for the IDE, storeXpathCount just to show all the items we find. We don't always individually just find one, we can find many. But again, this can be used whether we are doing Java, or Python, or C# or whatever language we're using. We can use this across all of our tests. So, in this, the first test, we're using //input. And this is an Xpath to say, "Find me all tags that are input tags." And so if I run this test right here, you can see it counts 5 tags inside of there. If we go to this, this is the click, we can see we can say, "Our target's going to be an Xpath search." We can put any of the options inside of here, this we have chosen xpath and //input. This finds the first one that's out there. Now, we're not limited to just something basic like that. Let's say we want to find all of these items that have a particular attribute. In this case, we'll say, "All the ones that have the class that points to the text Type." Those are the – that's the red one over there, we can put this in both of these now. Now, if we run this test, we can see we now have reduced our count to two.   
*The Selenium IDE is open and the Test Case pane lists an Untitled test case. The table on the Table tabbed page already contains four rows that list the commands, targets, and values for each of four recorded steps. Row 1 lists the command open and the target http://localhost:8080/website, Row 2 lists the command storeXpathCount, the target //input, and the value countValue, Row 3 lists the command echo and the target We counted ${countValue}, and Row 4 lists the command click and the target xpath=//input.  
  
In a separate window, a web page with the title Selectors is open in Firefox. It contains five input boxes, which are formatted as a table with Page and Source columns, and five rows. The Page column lists Text A through to Text E. The Source column lists HTML input type tags for each of the entries in the Page column. For Text A, it contains the code  
  
<input type="text" id="textA" value="Text A"/><br/>  
<div>  
  
For Text B, the code is  
  
<input type="text" class="textType" value="Tex tB"/><br/>  
<div>  
  
For Text C, the code is  
  
<input type="text" id="textC" value="Text C"/><br/>  
  
For Text D, the code is  
  
<input type="text" value="Text D" disabled="disabled"/><br/>  
  
For Text E, the code is  
  
<input type="text" class=textType" value="Text E"/><br/>  
  
The presenter highlights the source code in the input boxes and then moves to the IDE, where he highlights the commands in the Table tabbed page in the central pane.  
  
The presenter clicks the Play current selection button and the test runs successfully. He highlights the tags on the Log page in the IDE.  He then selects the final row in the table, which lists the command click and the target xpath=//input. He highlights xpath in the Target text box and then clicks the Find button. Text A in the input boxes on the web page is highlighted as a result.  
  
The presenter highlights the entries in the Table tabbed page, and then changes the entry in the Target text box to xpath=//input[@class=textType']. He refers to Text B and Text E entries in the input boxes on the web page as the red ones that are affected by the change in target.  
  
The presenter selects the second row in the table, which lists the command storeXpathCount and the target //input. As a result, in the Target text box,  the target changes to //input[@class=textType'].  
  
The presenter then clicks the Play current test case button. He then highlights the relevant entries in the Log page which indicate the count has been reduced to 2.*   
  
These two red text items. The one with the class textType inside of here. Now, this is searching basically on the attribute. It doesn't know anything about the class as a specific HTML feature. It's just saying, "Which one of these has the class attribute that has a value of textType?" I can do a simpler search just to say, "Show me all the items that have any class at all." And I could just take out the textType inside of here. You can see I can just come by here, and just take this out. And then when I run it this way, you can see I still have two. There's only two of them that have any class at all, but it's still a valid search. If it's not a valid search, it will tell us so. I mean it'll just say this search doesn't make any sense. So, it's going to help us a little bit that way as well. So, again, if you wanted to search for something with a specific ID, an ID is just an attribute inside of there. I can search for an item with this specific ID of textA and it would go off and find that. It'd find one of them inside of there. And this could be updated in either location. I'm just showing you the Xpath here. But to just show you, inside of many commands, I can put the **Target** outside of there. And if I do a **Find**, it shows me the specific one out there. Again, I could do **Target** this to be D if I wanted to, and I can find whichever one I want to. See it didn't find D out there. Let's – oh that's because D doesn't have an ID on it, C is the one with the ID and that's my apologies. Find this one and then it will find the one with C. It's pretty smart search.   
*On the web page, he points to Text B and Text E, highlighting the class="textType" value section of the code.   
  
In the Selenium IDE, the second row in the table is selected - the command is storeXpathCount, the target is //input@class='textType'] and the value is countValue.  
  
To do a simpler search, the presenter changes the entry to //input[@class] in the Target text box. He then clicks the Play entire test suite button. He goes back to the web page to point to Text B and Text  E.  
  
The test runs successfully and the Log tabbed page includes the entry [info] echo: We counted 2.  
  
The presenter changes the text inside the Target text box for the second row to //input[@id='textA' and clicks the Play entire test suite button again.  
  
The test runs and the Log tabbed page includes the entry [info] echo: We counted 1.  
  
The presenter selects the last row in the table, which lists the command click and the target xpath=//input[@class=textType']. In the Target text box, the presenter changes the target to xpath=//input[@id='textA']. He then clicks the Find button.   
  
Text A on the web page is then highlighted.   
  
In the Target text box, he then changes the entry to xpath=//input[@id='textD'] and clicks the Find button again. The Log tabbed page lists a series of entries that read  
  
[error] locator not found: xpath=//input[@id='textD']  
  
The presenter then changes the entry in the Target text box to xpath=//input[@id='textC'] and clicks the Find button. Text C is highlighted on the web page.*   
  
Now, I can get more details inside of Xpath. So this is // here it says, "Search for all tds, and inside of that look for anything that has an input as a child of the td." So, when I run this guy, you can see I got four of them. Now, why 4? It's because the second one inside of here is not inside of a cell as the rest of these are, it's inside of the div. So it's a div inside of a cell. And so if I wanted to see this one specifically, knowing it's inside of a div, I could say, div inside of here. And when I run this, it finds just one. It finds this one right here, the B item inside of there. So, there's a lot of different options. I could even do a check such as let's find anything that, for instance, contains an ID that has the word text inside of there. So, it has to have an ID, and that ID is going to contain the value text in some form or another. Now, again, this is going to find two of them, because they have both have the ID of text something. But I don't have to be limited to exact text. I can do the contains option to find something basically as a wildcard search. So, there's so many different options inside of here. For instance, as I said, let me go to the click just to show you some of these. I could look for an item that has the attribute disabled. Again, it's nothing smart about HTML, this is an XML search. It's just saying, "Does this have a property inside of here?" Right here, you can see disabled. If it has that property inside of here, we run it. I'm sorry let's go back to this, I can do a Find. You can see this box right there is the disabled, the one that's flashing.   
*The presenter selects the row that lists the command storeXpathCount, the target //td/input, and the value countValue. He then highlights the target in the Target text box - //td/input.   
  
The presenter runs the step and the Log tabbed page includes the entry [info] echo: We counted 4.   
  
The presenter switches to the open web page to point out the four instances that were found. He also highlights <input type="text" class="textType" value="Text B"/>, in the <div> class.  
  
He then returns to the Selenium IDE.  
  
In the Target text box, the presenter changes the entry from //td/input to //div/input. He then clicks the Play entire test suite button. The Log tabbed page includes the entry [info] echo: We counted 1.  
  
He also highlights Text B on the web page.  
  
Next the presenter changes the target text to read//input[contains(@id,'text')] and then clicks the Play current test case button. The Log tab contains the entry [info] echo: We counted 2. He points to the two input boxes on the web page that are highlighted - Text B and Text E.  
He then highlights the word "contains" in the Target text box in the IDE.   
  
The presenter selects the last row in the test case, which lists the command click and the target xpath=//input[@id='textC']. In the Target text box, the presenter changes the entry to xpath=//input[@disabled]. He then switches to the web page to highlight the word "disabled" in the code in the input boxes.  
  
He then clicks the Find button and Text D on the web page is highlighted.*   
  
I can do something like show me the last input in the sequence of inputs you have inside of there. So, I can say, of all the input boxes, show me the last one. And so that's actually B, because we told the other 4 first, then we go to the one nested inside of the div tag. Or I could say, "Hey, show me the one that is in a specific position." I can do that this way by using the position operation here. And the position will then return the position of the item and say, "Which one is equal to 3?" So, again D = 3. So I could select any of these off of here by locating the position, if I know the order these guys are located within. So, Xpath is really powerful for searching when I don't know the structure of my, arrangement of my — structure and arrangement of my HTML. And particularly if I'm not using CSS, it allows me to search this using raw XML searches. And so I don't have to be stuck whether or not something has IDs, or whether it has names or those things inside of the HTML, which makes it easier. I can use them however it lies and use Xpath to go and find the items. Again, whether I'm using IDE or any of the coding options, Xpath is a great option for being able to search and find the items as part of my tests.   
*The presenter changes the entry in the Target text box to xpath=//input[last()] and then clicks the Find button. Text B on the web page is highlighted.  
  
The presenter changes the target to xpath=//input[position()=3 and clicks the Find button. Text D is highlighted on the web page.  
  
He points to the entries in the input boxes on the web page to indicate any of them could be selected by their order.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

DOM Locators

Learning Objective

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

* *use the DOM as a locator strategy with Selenium commands*

**1. Searching with DOM locators**

These days the tie between HTML and JavaScript is tight and here to stay. If your preferred method for looking through HTML elements is JavaScript, Selenium will allow you to use the Document Object Model or, 'DOM', to find page elements you need within your test. Let's take a look at using the DOM as part of a locator within our Selenium testing. So, for our testing, we are going to search using the click element. It's just representative of something with a target, anything with a target inside of here I choose what my target is going to look like and in this case I am choosing DOM. I can put any elements of DOM inside of here, and so I can select, as we can say in this case from the form element. So, I know my document has form elements, I can go to the first form that comes inside of here, there is a form that wraps around this inside of here and look at the elements and find the element with the ID of 'textC' inside of here. So, again I hit the **Find** button and it finds it just fine. Now, I'm not limited to the IDE, I could search within the IDE or I can search within my code as well. Now, if I'm using Java or Python or C# or whatever code language I'm using within Selenium, I might not find the command to say find by DOM, but I can execute any script that's going to allow me to even be more, you know, detailed in one line inside of here, and be able to find whatever I want to. So, DOM is always an option for you and so if you know JavaScript and DOM best, use it within your Selenium tests. So, again another example here of using the form. I can go to the first form inside of my page or if my form's named, I can change this ID into a form. And if I don't know the name of an item, I can just go get the fourth item inside of here. So, again the fourth item is 'E', and 'E' has no ID associated with it, it has no particular name associated with it.   
*The Selenium IDE is open and the Test Case pane lists an Untitled test case. The table in the IDE currently contains two rows. Row 1 lists the open command and the target http://localhost:8080/web site/pages/selector.html. Row 2, which is currently selected, lists the command click and the target dom=document.forms[0].elements['textC'].  
  
In a separate window, a web page titled Selectors is open in Firefox. It contains a table with Page and Source columns, and five rows. The Page column lists Text A through to Text E. The Source column lists HTML input type tags for each of the entries in the Page column. For Text A, it contains the code  
  
<input type="text" id="textA" value="Text A"/><br/>  
<div>  
  
For Text B, the code is  
  
<input type="text" class="textType" value="Tex tB"/><br/>  
<div>  
  
For Text C, the code is  
  
<input type="text" id="textC" value="Text C"/><br/>  
  
For Text D, the code is  
  
<input type="text" value="Text D" disabled="disabled"/><br/>  
  
For Text E, the code is  
  
<input type="text" class=textType" value="Text E"/><br/>  
  
Currently the entries Text B and Text E are displaying in red. In the Selenium IDE, the Log tabbed page lists a series of errors specifying that the locator wasn't found.  
  
In the IDE, the presenter highlights the step with the command click and the target dom=dcocument.forms[0].ele...in the Table tabbed page.  
  
He then highlights parts of the text in the Target text box, which contains the following text:   dom=document.forms[0].elements['textC'] . He briefly goes to the web page to indicate how the form wraps around in the input boxes on the page.  
  
He clicks the Find button back in the IDE, and Text C is highlighted on the web page.  
  
He highlights the entry in the Target text box and changes the target to dom=document.forms[0].elements[4] and clicks the Find button. Text E is highlighted on the web page.*  
  
And so I can go and get it, by the fourth element. Again, it's zero listed, so A is 0, B is 1, C is 2, D is 3 and then E is 4. I can also if there is a name use the shorter hand I can go get item A by just saying hey, give me forms, give me the items with an ID of Text A. So, forms is the most basic way to be able to find elements and input elements in particular on my page, but it's not the only way. I have the ability to use the more detailed DOM searches directly as well. So, what's more typical inside of a DOM search is I use a method such as getElementById. So, the element A, I could just as easily **Find** by jumping straight to it. In this way, it doesn't even need to be inside of a form tag or inside of some other structure inside of there. It can be anywhere in my page, I am searching by ID, so particularly when my page is setup with IDs and I want to use JavaScript, this is a great way to go off and find that. I'm not limited to just the search by ID though, there are other options, and so for instance inside of my JavaScript and my DOM, there is a method out there to search by tag name. So, I can search for all of the tags that are input tags inside of there, now let me take this last little bit out for a second and show you what happens. So, when I say search by tag name and if I clear this out and I say **Find** inside of here, we get the error message, the one that I just cleared out saying the locator is not found. I'm finding elements by tag name, but there is a bunch of them coming back, this target has to point to a single element to make sense. And so when I search on this I'm going to get five input elements and in this case that could be more or less depending on what's going on out there. And so I'm getting an array of them and so I can go off and say, give me the number 2 item, this is actually the third item, 0, 1, 2 as we talked about inside of here.   
*The presenter highlights Text A through Text E on the web page to indicate the numbers they are associated with.   
  
Back in the IDE, he changes the Target text box to dom=document.forms[0].textA and clicks Find. Text A is highlighted on the web page.  
  
Next, the presenter changes the Target text box entry to dom=document.getElementByID('textA') and clicks the Find button. Text A is highlighted on the web page.  
  
He then changes the target to dom=document.getElementsByTagName('input')[2].  
He removes the [2] at the end of the entry.  
  
And he clicks the Clear button to clear the Log tabbed page.  
  
The presenter clicks Find and the following error message appears on the Log tabbed page:  
  
[error] locator not found: dom=document.getElementsByTagName('input'), error = TypeError: e.scrollIntoView is not a function  
  
The presenter changes the target for the selected row to  
dom=document.getElementsByTagName('input')[2]  
  
He then clicks the Find button and Text C is highlighted on the web page.*  
  
And so if I don't have IDs, but I know that there is input boxes, and I know the order of them, I can go ask them in a particular order. Now, I can do 0, 1, 2, 3, 4, if there is a static number on the page. If it's a dynamic number, I can actually go and search for the array and do something like the array length. So, I could go through inside of here and just do a nested search and put something inside of here, .length, I cannot type, length-1. And so I could do a search like this that will return me the last item or length-2, the next to last item. So, again I'm limited to one line, but I can use that very cleverly inside of there. So, the last one I will show you here before we wrap up is I can do by class name as well. So, I can do by ID, I can put my tag name or do by a class name and this will show me all the elements that have this class text type. Now, there is two of them outside of here, you can see a class ID, it's textType class ID, it's textType. Now, within our test then I have to again point to either the 0 or the first element. This is the first element always that comes back and again you can see the first one that comes inside of here or I can point to this second one that comes inside of here. So, the DOM is a powerful way to navigate our HTML page within JavaScript and Selenium testing is going to allow you to use that and use that information you might be developing on anyway to be able to find items in your tests. So, whether you're doing Selenium inside of the IDE or whether you're doing it in code, use the DOM to great effect to be able to grab the items and complete your tests and complete by finding the items you need very specifically knowing how you would find them in DOM just as if you are writing a JavaScript solution for your page.   
*The presenter then changes the target to  
dom=document.getElementsByTagName('input').length-1  
  
The presenter clicks the Find button. Text E is highlighted on the web page.  
  
Next the presenter changes the target to  
dom=document.getElementsByTagName('input').length-2  
  
He clicks the Find button. Text D is highlighted on the web page.  
  
To search by class name, the presenter changes the target to  
dom=document.getElementByClassName('textType')[0]  
  
On the web page in Firefox, the presenter highlights the two instances of class="textType" that appear in the Source section of the table.  
  
In the Selenium IDE, the presenter changes the target for the selected row to  
dom=document.getElementByClassName('textType'  
)[0]  
  
The presenter clicks the Find button and Text B is highlighted on the web page..  
  
He then changes the Target text to:  
  
dom=document.getElementByClassName('textType'  
)[1], and clicks Find. Text E is highlighted on the web page.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

CSS Selectors

Learning Objective

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

* *use CSS selectors as a locator strategy with Selenium commands*

**1. Searching with CSS selectors**

The biggest challenge in validating content on a page is to find what we're looking for amongst all of the HTML and CSS code that renders the page. Many sites use Cascading Style Sheets to decorate and arrange the page and Selenium allows us to piggyback on this, to help us locate the items on the page. Let's take a look at how we can use CSS selectors inside of our Selenium tests. So, we're going to use the IDE here, but these CSS selectors can be used in code versions as well, so: Java, C#, Python, whatever other code versions we might choose to use. Now, we're going to use the CSS Count here, the storeCSSCount to show when we can find a bunch of items, but I also want you to see many commands allow us to focus on the target using the CSS as well.We use the click, in this case. So, if I do a **Find** here, you can see this pound Text A is looking for something with the ID of textA, right there. And so here, I can go through and I can run this whole test and see we even get a count of 1 on Text A. Now, there's many different types of CSS selectors. This is just the first one, here by ID. I could change this really quickly to say let's look at everything with a .textType. Okay, so the .textType inside of here, is going to count all the items that are red. Now, do a **Find** on this guy, I'm sorry, the Find will work on this guy, if I run this guy. You can see there's two of them out there. There's two items that are decorated in red. That's the class "textType" in here, so the .textType and the class. I could be very simple and just say input. Show me all the input items out here and you can see there's just five input items out here. There's five text boxes. It's not looking at this, because this is just text. We're just actually escaping out to see the text inside of here.   
*The Selenium IDE is open and the Test Case pane lists an Untitled test case. The table in the IDE Table tabbed page contains four rows. Row  1 lists the command open and the target http://localhost:8080/website/, Row 2 lists the command storeCssCount,  the target #textA, and the value countValue, Row 3 lists the command echo and the target We counted ${countValue}, and Row 4 lists the command click and the target css=#textA.  
  
The row that lists the command click and the target css=#textA is currently selected.  
  
In a separate window, a web page with the title Selectors is open in Firefox.It contains a table with Page and Source columns, and five rows. The Page column lists Text A through to Text E. The Source column lists HTML input type tags for each of the entries in the Page column. For Text A, it contains the code  
  
<input type="text" id="textA" value="Text A"/><br/>  
<div>  
  
For Text B, the code is  
  
<input type="text" class="textType" value="Tex tB"/><br/>  
<div>  
  
For Text C, the code is  
  
<input type="text" id="textC" value="Text C"/><br/>  
  
For Text D, the code is  
  
<input type="text" value="Text D" disabled="disabled"/><br/>  
  
For Text E, the code is  
  
<input type="text" class=textType" value="Text E"/><br/>  
  
In the Selenium IDE, the presenter selects the second row, which lists the command storestoreCssCount and the target #textA.  
  
He then selects the step with the command click and the target css=#text A and clicks the Find button. Text A is highlighted in the Page column of the table on the web page. The presenter highlights id="textA" in the Source column of the table.   
  
He runs the test in the IDE by clicking the Play button. He highlights the entry in the Log page that says [info]echo: We counted 1.   
  
The presenter then changes the entry in the Target text box to .textType. He clicks the Find button and then clicks the Play entire test suite button. The Log tabbed page shows that two items were found in the search.  
  
He points to the two red items in the table on the web page, namely Text B and Text E. He then highlights class="textType" in the Source column for Text B.  
  
The presenter changes the target in the Target text box from textType to input and then clicks the Play entire test suite button. The Log shows that five items were found in the search.  
  
The presenter switches to the web page in Firefox and highlights all of the code text in the Source section of the Locators table. He then returns to the Selenium IDE.*  
  
Now, we can be more complicated than that. We can be more interesting than that. I can look for an input item inside of a td box. So, this is all of the input items that are inside of a column and row, and basically inside of a cell inside of a table. There's another selector I can choose. if run this, you can see there's four of them. There's four of them because this one's inside of a div tag. There's one of these we put inside of a div. I could do a div right here, just as easily, and when I run this one, you can see there's only 1 of them that's showing up. And so, we can be very specific within here. Now, there's a lot of different options. Let's look at a couple of other advanced ones here. So, let's say we want to find something with an exact text value that's inside of this, something I put in exactly as text. I am going to go take a look at the click option down here instead, just to show you some variety. This would work on either one. So, you can see, I can do css=input inside of here, where the value of the box is equal to Text. I can do this on any attribute of the box. I chose a value for this one. You can see I **Find** here, Text E. And the **Find** will, you know… see that little yellow highlight over here, it shows up as Text E when I hit the **Find** button. There's many, many options inside of here. I can do the options you saw up above. Let's just show some of the other ones. Let's say I want to find Text D, which is disabled. So I can say, I have to do css= inside of here, input is disabled. So, this is; does it have a disabled field inside of it? And there, it's been disabled. I can Find that one inside of there.   
*The presenter changes the target from input to td>input and then clicks the Play entire test suite button. The Log shows that four items were found in the search.  
  
The presenter highlights the code on the web page, in particular <div> tag.  
  
He then changes the target to div>input and then clicks the Play entire test suite button. The presenter highlights the entry within the Log page that indicates that one item was found.  
  
The presenter selects the step with the command click and the target css=#textA. He changes the text inside the Target text box to css=input[value='Text E'] and then clicks the Find button. Text E is then highlighted on the web page.  
  
The presenter changes the target inside the Target text box to css=input:disabled and clicks the Find button. Text D is highlighted on the web page.*  
  
A couple more options just to show you the breadth of CSS. Again, I'm not here to teach you CSS selectors, I just want to show you how we can use them inside of our Selenium here. So, there's much more than you can see here; there are some definitely worth researching. But we can, I am sorry, we have to do a css= inside of here. Inside of the click command, we have to tell it what type of selector we're doing. So, css=. What's the last input item outside of there? Now here, you can see there's a little trick. It shows Text B. That's because Text B is inside of the div. It goes to all of these items first: 1, 2, 3, 4 and then it goes to Text B, 5, because it's one layer nested inside of there. Now, if I know that I want to get to a specific item though, I don't have to just say last item and first item and take a guess, I actually can go to what's called the nth item inside of there. And so I can say go to the third one inside of here, so that's Text D. I can go to the fourth one inside of here, that's Text E. You know, I can choose off of this, using my CSS selectors. Now, when we're working on this and I do recording, Selenium is going to offer me some choices. If I record a click or type, it's going to offer me some choices but I just wanted to point out that if you know CSS selectors, you can choose and customize this in many different ways. It's a good option for being able to focus in and target your tests and not just be at the mercy of what one page happens to render. If you know how the page is designed, you can predict it and use the CSS selectors to improve your testing. So definitely, if you don't know much about CSS selectors, take some time, do some searching on them and find some resources to help you. Use this tool in any type of Selenium testing you're going to do, to really be able to map directly to items, when you're using HTML in CSS.   
*The presenter changes the target to input:last-of-type and clicks the Find button. This time the Log tabbed page shows an error, which specifies that the locator was not found.  
  
The presenter changes the target to read css=input:last-of-type and clicks the Find button. Text B is highlighted on the web page as a result.   
  
The presenter highlights instances of the type="text" in the Source section of the Selectors table on the web page.  
  
In the Selenium IDE, the presenter changes the target inside the Target text box  to css=input:nth-of-type(3) and clicks the Find button. Text D is then highlighted on the web page.  
  
Next the presenter changes the target to css=input:nth-of-type(4) and clicks the Find button. Text E is highlighted on the web page.  
  
  
  
Finally the presenter changes the target to css=input:nth-of-type(2) and clicks the Find button. Text C is then highlighted on the web page.  
  
He highlights the target text within the Target text box.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Locators in the IDE

Learning Objective

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

* *use the Selenium IDE to locate elements*

**1. Using advanced locators**

While the Selenium IDE does an amazing job of recording behavior, we sometimes need to tweak a recording or add elements that are not recordable. When doing this, we need to be able to identify elements in our HTML pages to provide to the test steps. The IDE still provides some helpful tools to let us target the elements as we need. So, here, we have a page that we're going to do some general testing on. So, let's just start with the recording. We'll put a few elements inside of here. So, we say hey the **Poll** **Name** is Animals, and the question is Which is your favorite? And then we can put in some elements here, like dog, cat, fish. Let's just put a few elements inside of there. Now you can see it did an amazing job of recording all this stuff. It picks them out, it targets them, it does all this great stuff. As we're doing validation then, we can come along and pick something along the lines to say "Okay let's make sure that the fish was out here." So I can say "Assert that the fish was out here." And it even targets that really well. And so if I walk through each of these steps, I can actually just run it and you can see it's going to go through and do all the steps and insert this out here. At this last element then, it has a very complex target out here. We need to know is this really what I was going off there to target? So let me shrink this down so we can see the page at the same time. Let me show you a few of the elements we can do and choose off of here. So, we know we are going to do an assertText and it will pick it for us when we right-click it outside of there. But let's say we don't want to do an assertText. Let's say we want to do, instead of saying that this guy must be out there, maybe we want to do some validation out there. We need to do some stuff by hand. And so let's actually take this step out. So, let's come down here   
*In Firefox, a web page with the title Add a Poll is open. It contains a Poll Name text box, Active Date and Inactive Date text boxes with calendar buttons, a Question text box, a Possible Answer text box with an Add button, a Language drop-down list box with a Translate button, and Save Poll and Cancel buttons.  
  
The presenter opens the Selenium IDE. The Test Case pane lists an Untitled test case. No steps have been recorded yet. The presenter clicks the Record button and returns to the open web page.  
  
The presenter types Animals in the Poll Name text box. He then types Which is your favorite? in the Question text box. He types dog in the Possible Answer text box and clicks Add, types cat in the Possible Answer text box and clicks Add, and then types fish in the Possible Answer text box and clicks Add. Three radio buttons – cat, dog, and fish – now display below the text box. Edit, Delete, Up, and Down buttons are now also available on the page.  
  
The presenter switches to the Selenium IDE and scrolls through the recorded steps. The steps are in a table in the Table tabbed page and 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 Animals, Row 3 lists the command type, the target name=question, and the value Which is your favorite, Row 4 lists the command type, the target name=answer, and the value dog. The second-last row in the table lists the command type, the target name=answer, and the value fish. The last row lists the command clickAndWait and the target name=buttonPressed.  
  
On the web page, the presenter selects and right-clicks the fish radio button, and selects assertText //div[@id='content']/form/table/tbody/tr[6]/td[2]/p[3] fish from the shortcut menu. He switches to the Selenium IDE where a step has been captured and lists the command assertText, the target //div[@id='content']/form/table/tbodytr[6]/td[2]/p, and the value fish. The presenter clicks the Play current selection button. The test case runs successfully, leaving all of the recorded steps highlighted in green.  
  
The presenter selects the final row in the table, which lists the command assertText and the complex target //div[@id='content']/form/table/tbody/tr[6]/td[2]/p[3].  
  
He makes the IDE interface smaller on the screen.. He highlights AssertText in the Command text box. He then points to the text on the web page.*  
  
and we're going to delete this step entirely. And instead we want to do a verifyText inside of there. Because the verify, if you remember, won't fail the test and stop testing. It'll just say "Hey, this text is supposed to be out of here." And so wait, I need to verify what, I need to verify an element out here. How do I pick this? I know I need a dog out here. How do I pick this? Well the first quick way to do this is like hit the **Select** button and go and click on it. And it says, "css=p and verify that's dog." So let's go ahead and oh yeah verify and click it, yeah there is definitely dog out here. So, if you double-click this it will execute that and then yeah, that works well. So, let's add another step here and go to the next one. So verifyText alright, and we would need to make sure that the cat is out there. Sorry and then we do a **Select** and we go select the next one. And oh now it came as a very complicated one. So, if you look at the two of these side by side and make it up, dog was css=p and then cat is very complex inside of there. But you know hey we can… just click on that and see it still works, all very nice. Now, one of the tricks inside of this though is these selectors are relative to the page as it's static at the moment. So, let's say I want to go there and maybe catch a cat before dog. Say I don't want to bias dog owners to click the first one, so I want to move cats above dogs inside of there. And if I went by and looked at this again and I said verifyText well css=p. It's not, it's failing, it's what's going on inside of there.   
*The presenter scrolls through the steps in the IDE Table tabbed page and highlights the assertText step that he just added.  He then types verifyText in the Command text box and dog in the Value text box and then clicks the Select button before switching to Firefox and clicking the dog option. The text css=p appears in Target text box when the presenter selects the dog option in Firefox. He clicks the step in the Table tabbed page and it verifies.   
  
Next, the presenter clicks a new, blank row below the last row. He types verifyText in the Command text box and cat in the Value text box. He then selects the cat radio button on the web page and the Target text box is automatically set to  //div[@id='content']/form/table/tbody/tr[6]/td[2]/p[2].  
  
He highlights the step for verifying dog and for verifying cat, drawing attention to the more complex target for cat.  
  
The presenter clicks the step with the command verifyText and the value cat. The step runs successfully.  
  
He highlights the Command, Target, and Value text boxes (selectors) in the IDE.  
  
On the web page, the presenter clicks the cat radio button and then clicks the Up button to move it to the first position in the set of radio buttons.   
  
In the Selenium IDE, the presenter selects and plays the step with the value of dog. The step fails and is highlighted in red.*   
  
Well remember that this is not the text, this is the location. So how do I know which location it's working on here? Well that's what the **Find** button is for. If I hit **Find**, you see now it's pointing to the cat text. And so this is also a helpful debug tool, allowing me point to what's going on out there. I come down to this guy, this one also fail, we do a Find it's going to point to dog. And so this will allow me to go through and tweak these. Now we can get even fancier inside of our -- inside of our solution there. We can go through and say, "You know, I just need to know that this guy is out here. I need to verify this guy out here." And so as part of that, I can do a very specific x path inside of there. So let me paste this one I've already created here. So, what we're saying here is, "I'm going to go to a p element. And I want a p element inside of here that contains the text, you can see that the text is contained inside fish." So I'm a little bit cheating. I'm just saying that one of these elements should have fish inside of it. And if I run this test right here, it's going to fail because it says cat, but I can change it to be fish down here. You know, so Find, this guy says fish, execute this, yes it now indeed it does says fish. But that's where the **Find** comes up to make it helpful. So I go back and I want to fix this one up above, I can go to – sorry, let me make this bigger again. Paste this inside of here, so we want the one that contains d.   
*The presenter clicks the Find button and the cat option on the web page flashes yellow. The presenter runs the step with the verifyText command and the cat value. The step fails, leaving it highlighted in red. When the presenter clicks the Find button, the dog option on the web page flashes yellow.   
  
He points radio buttons as he explains about verifying what's out on the web page.  
  
To create a new specific xpath, the presenter creates a new step with the command verifyText, the target //p[text()[contains(.,'fish')]], and the value cat.   
  
He highlights the word contains in the target text within the Target text box.  
  
He clicks the Find button and the fish option flashes yellow on the web page. He runs the step with the command verifyText, the target  //p[text()[contains(.,'fish')]], and the value of cat. The step fails.  
  
The presenter changes the value of the step from cat to fish and then runs the step successfully. He enlarges the Selenium IDE. He then selects the step with the command verifyText and the value of dog and changes its target from css=p to //p[text()[contains(.,'d')]. He clicks the Find button and the dog option flasheson the web page.*   
  
Okay, so yeah that works fine. Now, it's finding it right there. So we do there, except for if I do a dog fish. I add that inside of there and say, "I don't want my fishes next to each other." Let's move this up – I hit the wrong thing there. I will move that up inside of here. Now if I say, "It contains d", I need to see it finds the first one that contains d. So I have to be very accurate inside of here. But the find is going to help me do that. So **Select** will help you but the **Find** helps me refine it even further. So, use these two and these capability in the IDE to really allows to target and get exactly what we need as we're creating our tests by making sure the target of each of these commands that we put in there is exactly what I intended it to be.   
*The presenter switches to Firefox, types dogfish in the Possible Answer text box, and clicks Add. A new dogfish radio button is added below the other radio buttons. The presenter selects the dogfish radio button and clicks the Up button twice to move it above the dog radio button.  
  
He then switches to the Selenium IDE and clicks Find for the command verifyText, the target //p[text()[contains(.,'d')] and the value dog . The dogfish option flashes yellow on the web page.  
  
He highlights the Find and Select buttons on the IDE.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Pattern Matching

Learning Objective

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

* *use patterns as parameters in Selenium commands*

**1. Pattern matching for content searches**

Selenium Verify and Asserts within the IDE can easily be added for static text. But what if the text is not exactly the same every time? In many commands, I want or need the flexibility to match patterns and to validate fields without having to know their exact value. We are given this ability in Selenium in several different ways, so let's take a look. So, I'm going to start recording a test here and I'm going to log in. I'm going to put the admin password here and then I login. And then I want to verify some text on the page. So, if I go through and I just come inside of here and I say, "Okay, I want to verify that" – let's say "that this text is on the page right there." I can come back and look onto my test and there it is. So here, I click on here, I sign and I'm asserting the text. Now, depending on the roles somebody logs into, maybe they're an administrator, maybe they're a user, maybe they're a super administrator, maybe that name varies inside of here. So, the issue is not necessarily what type of name comes up here. I mean this name could vary. The fact that it just came to some homepage. So maybe I want to just verify that they are at this homepage. So, how do I do that? Well the easy way around that is for this text inside of here, Selenium gives us the option to switch that using the glob command. And so putting a glob on the front like this says, "Hey, I can put in a wildcard."   
*The Selenium IDE is open. The Test Case pane lists the entry Untitled and no test steps have yet been recorded.The presenter clicks the Record button.  
  
In a separate window, a login page is open in Firefox. It contains Username and Password text boxes, and a Login button. The presenter types admin in the Username text box, types a password in the Password text box, and clicks the Login button. This opens a page titled Administrator Home. It includes a menu with the options home, news, calendar, contact us, admin home, logout, and no popups. It also contains a site menu and a New and Uncompleted Notices section, which currently displays the message No Notices to Handle.  
  
The presenter selects and right-clicks the title text Administrator Home and selects assertText css=h2 Administrator Home from the shortcut menu that opens. He then switches to the Selenium IDE.  
  
The table in the IDE now contains five rows. Row 1 lists the command open and the target /website/signon.vpd, Row 2 lists the command type, the target id=usernameInput, and new 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, and Row 5 lists the command assertText, the target css=h2, and the value Administrator Home.  
  
The presenter selects the final row, and the command assertText appears in the Command text box, the target css-h2 appears in the Target text box, and the value Administrator Home appears in the Value text box.   
  
He changes the Value text box to read glob: \*Home. He then adds a \* after Home. He  double-clicks the step in the Table tabbed page to run it.*   
  
And so when I check the text here, I can just say anything that says Home. I can even say anything Home\* or anything out there. And it'll run and check this even if the page isn't exactly the same. So, again, it could say user home, it could say admin home, super admin home, home for the admin, you know, any of those texts. As long as it says home inside of there, then I can kind of assume that something is going to show up. I can play with that quite easily and move it around and be very flexible within the pattern matching I have going on using \*. And I can actually use brackets to show individual characters in there as well. Now, just to show you here, we can use that command in many different places. We can do a just **verifyTextPresent** inside of here. And that isn't just – that's not like any specific item. I'm just using the target inside of here. I could say inside of here, glob for let's just say \*Notice\*. I just want to say, "Hey there's some notice inside of here that every user should have." And so I could put that inside of here as well and it works just fine. It's going to verify the text. If I said, "Is the text NotHere present on this text?" You can see it fails. The text \*NotHere\* is certainly not on here. It's not matching anything. But if I did Notice inside of here, then yes I can say, "Not only should the h2, see some side of a home inside of here, but somewhere on the page, this text should exist." This could be a type of text, test we do inside of here. Again, the details aren't terribly important.   
*The step runs successfully and is highlighted in green. The presenter changes the entry in the Value text box to glob:\*Home\* and then double-clicks the step to run it again. The step runs successfully.  
  
The presenter clicks an empty row below the last step. He types verifyTextPresent in the Command text box and glob:\*Notice\* in the Target text box, and then double-clicks the step. It runs successfully and is highlighted in green.  
  
The presenter changes the entry in the Target text box for the row to glob:\*NotHere\*and then double-clicks the row to run the step. It fails and is highlighted in red.  
  
The presenter changes the target to glob:\*Notice\*and then runs the step again. It executes successfully.  
  
Next the presenter selects the step with a target of css=h2 in the Table tabbed page.*   
  
Now, that being said, let's say we went over here and we went to the contact page. Alright and on the contact page, we would verify that this text is present. Well, I can actually go through and use a different term inside of here. I can say exact:. So I want to be \*\*\*.com. This allows me to get past some of the special characters and say, this text must match exactly inside of here. And you can see when we match it on here, it does match exactly. Put something else inside of here, it does not match d\*\*\*\*, it just has to have \*\*\* exactly. And that gives me, it gets me around the special character restriction inside of there. So, there's glob to be able to check your search and something inside of there. And then there's exact that says it needs to match exactly. We're here on the \* in this case with special characters. And there's other users of that inside of there as well. Now, the last thing we can check inside of here, the last two we'll talk about, is the regular expressions- the use of regular expressions. Very powerful, it gets very much more detailed inside of here. So, for instance, I want to check maybe every page has a copyright at the bottom. And so I can do that here by saying, "I want to verify this text that's out there somewhere." So, I can use this verifyTextPresent or I can use in a more exact search. And that's what I'm going to do in this case. I'm going to go ahead and switch this to be **verifyText** inside of here. And the target down here is going to be the footer.   
*The presenter switches to the Administrator Home web page and clicks the contact us option in the menu bar. The contact us web page opens. It includes the e-mail address staff@\*\*\*.com. The presenter highlights the address as the text he wants to verify.  
  
In the Selenium IDE, the presenter selects the final step with command verifyTextPresent and the target glob\*Notice\*. He changes the entry in the Target text box for the step to exact:\*\*\*.com and then double-clicks the step. The step runs successfully.  
  
The presenter changes the target for the step to exact:d\*\*\*.com inside the Target text box. He double-clicks the step and fails because it doesn't match the Target text.  
  
The presenter points to the steps in the Table tabbed page inside the IDE.   
  
He switches to the web page and highlights the following text at the bottom of the page: ©2005-20014  
  
Returning to the IDE, the presenter changes the entry in the Command text box to verifyText and the entry in the Target text box to id=footer. He clicks the Find button and the step runs successfully.*  
  
And I can just check that out, do **Find**, yeah see there's the footer. And then I can put in a regular expression. I've actually premade my regular expression here, so we won't type that out. But the regular expression is saying, I'm going to have a copyright, a space and then I can have the characters zero through nine and four of them, a dash and then zero through nine and four of them. And that's going to be my expression here. And when I check this out, you can see, yes indeed that is what's going on. So, regexp, regular expression, allows us to use whatever regular expressions. Very powerful and gets very much more detailed inside of here. And there is lots of rules around that behind the scope of Selenium. But to check that the copyright is formatted properly there's 4 digits out there. Again, if I switch it to 3 digits inside of here, that would be a failure. The regular expression would fail and in that case. It has to be four digits out there. And so that's the space I want to check, the space I want to do. So, whether I'm using some simple pattern matching using the glob, super fancy pattern matching within regular expressions, or just simply an exact text item, these are options I can do past just the simple rules of text matching that's out there for Selenium.   
*The footer ©2005-2014 is highlighted on the web page.  
  
The presenter adds the text regexp:© [0-9]{4}-[0-9]{4} to the Value text box in the IDE and then double-clicks the step. It runs successfully and is highlighted in green.  
  
On the contact us web page, he points to the footer text ©2005-2014.  
  
In the Selenium IDE, the presenter changes the entry in the Value text box to three digits to read regexp:© [0-9]{3}-[0-9]{4}. He double-clicks the step and it fails to run.  
  
The presenter changes the value back to regexp:© [0-9]{4}-[0-9]{4} and double-clicks the step. It runs successfully.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Using Variables

Learning Objective

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

* *use variables in Selenium scripts*

**1. Saving values as variables**

Not all tests we create do we know, or even care, about the values that are going to show up on the site. We sometimes simply need to validate values that are used consistently on the site and this would be nice to be able to create a test case that makes use of dynamic values. Selenium IDE, in fact, does give us the ability to save values as variables and then use them within other commands we're going to do as we go through our testing. So, in this page we can see here, we have a randomly generated number, we use JavaScript to randomly generate a number, and when we click a button, this randomly generated number is updated from here in the value over to here. So, just to show you. If I type something here, whatever is inside of this box, it updates inside of here. But when I open up the page and it gives me an initial value, just randomly generated inside of there. So, for our test, we could do something along these lines. We record the test and we say, you know, let's verify here that there is a value to start with. So, we need to see that this value to start with inside of there and then we're going to press the button. Okay, and you see that we click the button inside of there and then we are going to validate the text inside of here. So, say okay, let's assert that the text is stored inside of here. So, we start with a value here and then we have the same value here. Very simple. I run my test and nothing works. Why? Because when I open up the page, as we said, we get a different value every single time. So, every time I run this test, we'll see the actual value's not going to match the real value and the odds are very low that it ever would, given the random generation that's going on inside of here. So, what we'd like to be able to do instead is to, instead of verifying an exact value, what if we go through and let's do this.   
*The Selenium IDE is open. The Test Case pane lists the entry Untitled and no test steps have yet been recorded.  
  
In a separate window, a web page titled Variables is open in Firefox. The page includes a text box that contains a randomly generated number – currently 0.5447569374957916. The message "To be Replaced" displays below the text box, and there's an Update button. The presenter clicks the Update button and the number 0.5447569374957916 displays in the text box. It also displays in place of the message "To be Replaced." The number in the text box is updated and appears in the space below the text box.  
The presenter deletes the number from the text box and types a random string of characters, in this case fdsafa. He then clicks the Update button and the random text displays below the text box.  
  
The presenter selects the URL for the page in the Firefox address bar and presses Enter to reload the page. When the page reloads, a new random number, in this case 0.2887415087292311, appears in the text box.  
  
In the Selenium IDE, the presenter clicks the Record button. He switches back to the web page, selects and right-clicks the number in the text box, and selects verifyValue id=inputValue 0.2887415087292311 from the shortcut menu that opens.  
  
Inside the IDE, on the Table tabbed page, the table contains two rows. Row 1 lists the command open and the target /website/content/variable and Row 2 lists the command verifyValue, the target id=inputValue, and the value 0.28875\4515807292311.  
  
The presenter switches to Firefox and clicks the Update button on the page, and the random number displays below the text box. The presenter selects and right-clicks the number, and selects the option assertText id=outputValue 0.2887415087292311 from the shortcut menu.   
  
The presenter highlights the two rows that have been added inside the Selenium IDE Table tapped page. The first added step lists the command click, the target id=update, and the value 0.2887415087292311 and the second added step lists the command asserText, the target id=outputValue, and the value 0.2887415087292311.  
  
The presenter clicks the Play current test case button and the test runs, with errors in the second and final steps. Both of these steps are highlighted red.   
  
On the web page the value has changed, to 0.171446683927459 as the presenter opens up the page again.  
  
The Log tabbed page at the bottom of the page displays an error message that reads "Actual value 0.171446683927459 did not match 0.2887415087292311."*   
  
Let's redo our test. We can actually type it in by hand or we can actually do this through the recording. So, I am going to, and let me just delete everything here, we're going to start over here. So now, instead of verifying the value, there's another option here called storeValue. And when I click storeValue, it's going to give me this pop-up to say, what do you want to store the value as? And a variable name. So I'm going to say valueFromSite. This might be some other Value: the Username, the User ID number, the number of items sold, randomly generated, whatever. Whatever variable name I want to put inside of there. But now you can see I do a storeValue inside of here. And then I can go through, I can click my **Update** button. Alright, and then the last step I want to do is then assert that text. Okay, so assert the Text is here, but now again, I don't want to assert this value. If I run the test from here, it'll store the value, but it'll still fail, because the value is going to be generated randomly inside of there. So, what I want to do instead is, instead of changing up against this predefined, this hardcoded value, let's change to be the variable. The way I do that is I do a $ and inside of these curly brackets, I put the variable name, valueFromSite. And so what Selenium will do is, it will take the variable and translate it, and grab the value and compare those values. So, now when I run it, you can see it does indeed pass. It's going to check these two values. If I, you know, just type in something randomly inside of there, this works just fine. It doesn't make any changes here, because it generates it, as well. But it again, ignores the specific values. It's just going to use those values around.   
*The presenter deletes the steps on the Table tabbed page and clicks the Record button.  
  
He switches to web page, highlights the random number in the text box and right clicks to get the shortcut menu. He selects storeValue id=inputValue0.171446683927459 for the menu. A [JavaScript Application] dialog box opens with the text Enter the name of the variable, an empty text box, and OK and Cancel buttons. The presenter types valueFromSite in the text box and clicks OK.  
  
He highlights the random number on the web page before returning to the Selenium IDE. Two steps have been added inside the Table tabbed page. Row 1 contains the command open and the target /website/content/variable... Row two contains the command storeValue, the target id=inputValue, and the value valueFromSite. He highlights the row two step inside the IDE and then moves back to the web page where he highlights the random number not inside the text box and clicks the Update button.  
  
The IDE briefly shows the step inside the Table tabbed page. The command for the step is click and the target is id=update.   
  
Back on the web page, the presenter right-clicks the random number and selects assertText id=outputValue 0.171446683927459.  
  
In the IDE, the assertText step has been added. The command for the step is assertText, the target is id=outputVale, and the target is 0.171446683927459.   
  
He highlights this step and clicks the Play current test case button. The test fails.   
  
The presenter highlights the text in the Value text box - which is the random number 0.171446683927459. He replaces this number with $(valueFromSite). He highlights the text in the Command and Target text boxes (as he says "compare those values").   
  
He runs the test and it is successful.  
  
The presenter highlights the random variables on the web page and then changes the variable in the text box to dsafdas and clicks the Update button.  
  
Returning to the IDE, he clicks the Play current test case button and it runs successfully.*  
  
And this gives me a lot of the flexibility in my testing. I can look off into my site and then be able to pull it back. Now, it does make things a little bit difficult as far as de-bugging goes because if it's not working, I'm not necessarily sure what's going on here, but we aren't without recourse. If I insert a command here, Selenium has the echo command and the echo allows me to put anything inside of here. I can just say Hello inside of here. If I run my test here, you can see, I add a line to the Log that says echo: Hello. But even better, I can put Value is and the $ and curly brackets inside of there, valueFromSite, and it will print out for me to, at least, see what that value is. It gives me that de-bug option to be able to look inside of the values and be able to see exactly what's going on inside of there. Now, just a quick note. There's a lot of store options inside of here. I can say store. I can store an Alert, I can store if the Alert was Present. I can store Buttons and Links and Fields, attributes and whether something is checked or not, a lot of different things inside of here. So, just as an example, I can say storeText and then I can then store the text from whichever element I want to in here. So, I can say select this element right here. So it's going to store the text of outputValue and then if I need it in some context, in this case, I'll just echo it out. I could say echo, I forgot to give it a variable name. Let me do that real quick. This is going to be textvalue. I can say **echo** here, ${textValue}. And this will allow me to create however many variables I need and use these variables in whichever context I need to pass them along. I'm showing you the echo here, but again, you can use them as part of a search, a part of validates. Wherever you can use a hardcoded value, you can replace it with a variable and have dynamic testing from there.   
*The presenter highlights the steps inside the IDE.  
  
He then selects the step with the click command, right-clicks to open a shortcut menu, and selects Insert New Command. A new row is added to the table, above the step with the command click and the target id=update.  
  
For the new row, the presenter types echo in the Command text box and Hello into the Target text box. He then clicks the Play current test case button. A line with the text echo:Hello is added to the Log tabbed page, which the presenter highlights.   
  
He changes the text in the Target text box to Value is ${valueFromSite} and then clicks the Play current test case button. A line with the text echo: Value is 0.836338816088316 is added to the Log tabbed page.  
  
He highlights the steps inside the Table tabbed page.  
  
The presenter then clicks the space below the last step to create a new step. He types store in the Command text box.The various options for completing the command appear in a drop-down list. The presenter scrolls through them, highlighting a few such as storeAlertPresent, storeAllButtons, storeAllLinks, and storeAllFields.  
  
The presenter completes the entry, typing storeText inside the Command text box. The presenter then clicks the Select button alongside the Target text box, switches to the web page, and selects the random number of 0.8363388160688316. This automatically enters id=outputValue in the Target text box.  
  
The presenter clicks the space below the final step to create a new step. He types echo in the Command text box.   
  
To add a variable to the storeText step (which he forgot to add), the presenter selects the step and types textvalue in its Value text box.   
  
The presenter selects the last step again and types ${textvalue} in the Target text box. He clicks the Play entire test suite button and all the steps run successfully.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Debugging

Learning Objective

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

* *debug Selenium tests*

**1. Applying basic intelligent testing**

The easiest way to create a Selenium test is to record our actions on the screen within the IDE. Sometimes though our recording only gives us part of a final automated test. We have to plan carefully and then adjust our tests to figure out not only to have them test correctly, but to do so on their own, when they are grouped as part of a regression testing bundle. Knowing how to checkout and debug our test is going to help us. So, let's take a look at this test right here and we are going to walk through and let this test execute itself. I have recorded it already, it's a pretty straightforward test, you can see it's going to go through here, it's going to put some information to a story. It's going to go through and it's going to edit the story, okay so it's edited there. It's going to go through and it's going to delete the story. So, I just created test to create, I just created the update, I just created test to delete all at once. And in the meantime, I'm actually viewing the story kind of sort of along the way too. So, I have created and tested all those main functions, create, read, update and delete. Now, this test works perfectly, I could – yeah I can put on fast, I can run it again, boom, perfect, it ran perfectly. But let me change the state of our system just a little bit. Let's add a new story, Bad interruptor. All right, this is going to go off and add this here in Some page and then I can hit **Done**. Okay so I have this test out here already, there is no guarantee that the system is going to be empty when I go through and run my test. So, now if go through and run my tests, it fails immediately. So, you can see I have come down to the step right inside of here and I can actually, you know let me actually go through and let me **Delete** this story, just to clean it up,   
*In the Selenium IDE, a test case called storyLifeTest is open and the table on the Table tabbed page lists several steps, some of which are not immediately visible. Row 1 lists the command open and the target /website/addNewStory, Row 2 lists the command type, the target id=titleInput, and the value Test Story, Row 3 lists the command type, the target id=pageInput, and the value page info, Row 4 lists the command clickAndWait and the target document.addStoryForm.buttonPressed[3], Row 5 lists the command assertText, the target //div[@id='content']/form/table/tbody/tr[2, and the value Test Story, Row 7 lists the command clicks and the target name=id, Row 8 lists the command clickAndWait and the target name=buttonPressed, Row 9 lists the command type, the target id=pageInput, and the value page Info 2, Row 10 lists the command type, the target id=titleInput, and the value Test Story 2, Row 10 lists the command clickAndWait and the target document.addStoryForm.buttonPressed[3], Row 11 lists the command assertText, the target //div[@id='content']/form/table/tbody, and the value Test Story 2, Row 12 lists the command click and the target name=id, Row 13 lists the command clickAndWait and the target document.viewStoriesForm.buttonPressed[1], Row 14 lists the command assertText, the target id=storyTitle, and the value Test Story 2, Row 15 lists the command clickAndWait and the target name=buttonPressed, Row 16 lists the command assertText, the target css=td, and the value No Stories currently defined.  
  
In a separate window, a web page titled Add a New Story is open in Firefox. It contains a Title of story text box, Date of story, Active Date, and Inactive Date text boxes with calendar buttons, a Page Text pane text box, and Preview Story, Done, and Cancel buttons.  
  
The presenter points to the list of steps inside the Table tabbed page of the IDE.  
  
He then clicks the Play current test case button and as the test runs, the Title of Story section on the web page is highlighted, then the Page Text section gets highlighted, The View News Stories page opens which includes a Test Story radio button. The radio button gets selected, and then the Delete Story page opens which includes Test Story 2 in a box with other stories, a Delete button and a cancel button. Test Story 2 is selected by default. The View News Stories page opens and this time it cointains the message No stories currently defined where there had been a Test Story radio button before.  
  
Back inside the IDE, the presenter drags the Playback Speed slider to its fastest setting and then clicks the Play current test case button again. The test case runs and the steps are performed again on the web page, ending with the View News Stories page. It contains Delete News Item, Add New Story, and Add Slideshow buttons.  
The presenter clicks the Add New Story button.  This opens the Add a New Story page. The presenter types Bad interruptor in the Title of Story text box, types Some page in the Page Text text box, and clicks the Done button. The View News Stories page opens. It now contains a Bad interruptor radio button.  
  
The presenter switches to the Selenium IDE and clicks the Play current test case button.   
  
As a result, the Add a New story page opens in the browser, and then the View News Stories page opens with the Bad interruptor and Test Story radio buttons available.   
  
Inside the IDE, the test fails at the step with the command assertText and the value Test Story. The presenter highlights the step inside the Table tabbed page of the IDE environment.*   
  
the one that's out there and let me run this really slow now, so we can see exactly what happens. So, I go in, I add my story inside of there, I click over here and I go trying to assert the text. Hey, is my story on here? It's there, it's just not in the location I thought. So, I need to have some ways by which I can go off and debug this. So, if I go and look at this, I look at this command right here, right here, I have – if I do the **Find**, you can see it's searching on the very first one that's out there. It's assuming that the story added is the first story out there. So, one of the things I could do is I could just go out and say hey, does this page contain anything, is there anywhere there on this page that has this text. So, I could actually do something along the lines of verifyTextPresent, all right and just say is this text right here and I am going to delete this out here and put this out here. Is the Test Story anywhere on the page, now if I use a very unique name and I only use this Test Name right here, then I can probably get away with this, let's see. Yeah, so that will work just fine so again and now I can go back, I can **Delete** this test, this test data and so there I can try and run my test again. So, I need to work through my testing this way, so again it goes through and it is going on, going on it is working okay still. I can edit it, that's looking good, it's looking good, we have fixed the first problem.   
*The presenter switches to the web page. He clicks the Delete News Item button, and then clicks the Delete button to clear the story.  
  
In the Selenium IDE, the presenter drags the Playback Speed slider to its slowest setting and then clicks the Play current test case button and the steps play out on the web page. The Title of Story section gets highlighted then the Page Text section gets highlighted, The View News Stories page opens which includes a Test Story radio button.   
  
The test fails at the same step as before, with the assertText command.  
  
The presenter selects the failed step inside the Table tabbed page of the IDE, and clicks the Find button alongside the Target text box. The Bad interruptor story flashes on the web page. With the step selected inside the IDE, the presenter changes the entry in the Command text box to verifyTextPresent. He changes the entry in the Target text box to Test Story and removes the entry from the Value text box. He points to the name Test Story in the IDE. He double-clicks the step to execute it and it runs successfully.  
  
On the web page, the presenter clicks the Test Story radio button and then the Delete News Item button. The Delete Story page opens up with the question Are you sure you want to delete this story? The presenter selects Test Story from the drop down menus and clicks the Delete button. The View News Stories page opens and the Test Story radio button is no longer under Title of Story.   
  
The presenter clicks the Play current test case button again. On the web page the Title of Story section gets highlighted, then the Page Text section gets highlighted , The View News Stories page opens which includes a Bad interruptor radio button and a Test Story radio button. The Bad interruptor button gets selected, and then the Edit Story page opens with Bad interruptor in the Title of Story text box. This is then edited to Test Story 2. The View News Stories page opens with two stories listed Test Story 2 and Test Story, along with radio buttons for each.  Then the Delete Story page opens with Test Story 2 highlighted. The Delete button is clicked and the View News Stories page opens again with just Test Story listed.*   
  
Oh wait I didn't fix the first problem because well what happened right there is if you look at that I just edited that extra story I put out there. My Test Story is still here I never did edit it, I edited some other story out there, well that's a problem and I should probably be doing Assert when I hit the Edit button as part of the test step inside of there. And when I get to that stuff, I should probably assert the fact that it is indeed the page I am looking at, as part of my test design I need to validate the things around here. Now, there is a bunch of other little things we can do as far as our testing goes. One of the items inside of here is when we go back and we saw that original test, so if I actually go and open that original test here, I'm not going to save it, and go to the original one and I get to this step right here. And let me – and I can leave that actually I will use that inside of there, go slow, we run it and let it fail inside of here. I'm sorry, it actually worked, but when I get this step inside of here what I could do is I could find a different value for this selector. You can see if I actually I apologize, let me go back to this, this, this, this and when I do the Assert here, I can use a different selector inside of here. I can select and say this guy right here and I can change from several different ones. There is X path, there is relative positions instead of here or even better when I go and click this guy, I'm going to go and click this guy right here actually.   
*The test fails at the step with the command assertText, the target css=id, and the value No Stories currently defined.   
  
The presenter highlights the Test Story radio button on the web page. He then highlights the Edit Story button on the View News Stories page.  
  
Returning to the IDE, the presenter selects File - Open. A dialog box with Save, Don't Save, and Cancel buttons asks if the presenter wants to save the changed test case, and the presenter clicks Don't Save.   
  
The Select a File dialog box opens and the  presenter opens the storyLifeTest text case. In the IDE, the presenter selects the step with the command assertText and the value Test Story on the Table tabbed page.  
  
He then clicks the Play current test case button inside the IDE.  In Firefox, the Title of Story section gets highlighted, then the Page Text section gets highlighted , The View News Stories page opens which includes a Test Story radio button. The presenter  pauses the test case after the first step with an assertText command and a Test Story value. He highlights this step in the IDE Table tabbed page.   
  
The presenter then double-clicks each step in turn, including the first step with a command of assertText, to run the steps one by one. He clicks the Select button on the IDE table tabbed page. and then switches to the web page, where he clicks the second of three Test Story radio buttons. He then highlights the Target text box, which reads //div[@id='content']/form/table/tbody/tr[2]/td[2]. There's also a drop-down menu with other options for the target text, which the presenter points to.  
  
He then selects the first Test Story radio button on the web page.  
  
The Target text of the first assertText step is now name=id on the Table tabbed page.*   
  
If I look at this, I have a name associated with this and there is several of these outside of here that I have to choose. This is the first one, the second one and the third one, but if I go and inspect these elements inside of here, I can ask my developers to put an ID or something on this field, see if there is some way I can make it easier to test by really allowing me to identify it. Okay, I want this one to match some target number, some values, some ID that I know when I test I can pull that out. And if you are the developer also writing the test, you can do this for yourself. So, there is several options I have to be aware of as I am doing the tests. One of them is planning the test appropriately, may be I have to ensure, may be I have to do a test or maybe a database swipe to ensure that truly my database is clean before I do any one of these tests. Or, otherwise I have to be smart in my tests, so I can't validate the last step of it says no stories, I have to maybe do account, use CSS Account feature or something like that. Just understand that not all recordings are going to work as is or at least they might not work later on. And so you can't check every test failure as being a problem with the site, you really have to have smart testing which kind of does lean towards maybe not just using IDE tests. If you use a code base test, either Java or C# or Python or any of the options that are out there, that will allow you to do even better, integrate with the database and check things in even more detail. So, you know exactly the state of your testing, so use these facts and tips may be get your test to be a little bit smarter and a little bit better in the automation.   
*On the web page, the presenter points to the three Test Story radio buttons. He then right-clicks the first Test Story radio button and selects the Inspect Element option from the shortcut menu. The source code for the element displays in a pane at the bottom of the web page. He highlights the relevant code, which is <input type="radio" value="514005555885900" name="id"></input>.  
  
He then points to the first Test Story radio button.  
  
Next the presenter clicks the third Test Story radio button and clicks the Delete News Item button. A page titled Delete Story contains a message confirming that the story will be deleted, as well as Delete and Cancel buttons. The presenter clicks Delete and the View News Stories page opens, now with only two Test Story radio buttons.  
  
The presenter selects the second Test Story radio button and clicks the Delete News Item button, and then clicks the Delete button on the Delete Story page.  
  
On the View News Stories page, the presenter selects the last remaining Test Story radio button, clicks the Delete News Item button, and then clicks the Delete button on the Delete Story page. The View News Stories page no longer lists any stories to view.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Using Any Browser

Learning Objective

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

* *run Selenium tests in any browser*

**1. Running tests in different browsers**

While the Selenium tool is an amazing tool allowing us quick recording and tweaking of test cases without having to know any specific coding language, it is limited in the testing must occur within the Firefox browser. It is to say test creation must occur in Firefox. We can use the Selenium test server to execute tests on many browsers from the command line, let's take a look. So, the Selenium test server is going to allow us to start with any little Firefox test suite that's out there, so we have a test suite, we have saved it off, this test suite is actually called All Tests and we are going to be able to add whatever test we want to, we can manage it off from the browser. And in order to get to the Selenium test server we go off to the Selenium web site here and we can download the Selenium server, you can see (formerly the Selenium RC Server) here and you can download it here. Now, the file after downloaded as we saw before comes in a single jar file. We can actually run this from command line. It's not a bunch to install, it's not a bunch of effort to get this going. So, any machine that can run Java can become our Selenium test server and so we are going to in this case utilize a little run script and that script is going to call the server for us. Now, the first command we are going to execute is just the basic command here, so we have a java start up here, we are going to start up a jar file which is that standalone server jar. And we are going to be running in htmlSuite inside of here and you can run that server and you can do a lot of remote testing with that server. But in this mode, we are going to run it in the htmlSuite mode, and so we are going to show the options for browsers here, that's what we're kind of have going on here and this \*show isn't the command actually it's just an error to get out there and get it going. And then you have some additional things, we will talk about those in just a second.   
*In an Explorer window, the presenter has selected the executable file named selenium-server-standalone-2.41.0.  
  
The presenter switches to the Selenium IDE, where the assertEval test suite is open. It contains seven test cases. The first test case, named login, is selected and the recorded steps display in the table on the Table tabbed page.  
  
The presenter switches to the Firefox browser where the Selenium web site is open on the Download tabbed page. The presenter points out the option for downloading Selenium Server, which reads Download version 2.41.0, where 2.41.0 is a link.  
  
The presenter goes back to the Explorer window to highlight the selenium-server-standalone-2.41.0 file again.He then clicks the runTest file in the Explorer window and it opens in Notepad.  
  
It contains the following code:  
  
@echo off  
rem java –jar selenium-server-standalone-2.41.0.jar –htmlsuite "\*firefox" "http://localhost:8080/" "..\ide\allTests.html" "firefoxResults.html"  
rem java –jar selenium-server-standalone-2.41.0.jar –htmlsuite "\*chrome" "http://localhost:8080/" "..\ide\allTests.html" "chromeResults.html"  
rem java –jar selenium-server-standalone-2.41.0.jar –htmlsuite "\*iexplorer" "http://localhost:8080/" "..\ide\allTests.html" "ieResults.html"  
java –jar selenium server-standalone-2.41.0 jar –htmlsuite "\*show" "http://localhost8080/" "..\ide\allTests.html" ieResults.html"  
  
He highlights the first command that will be executed, which is java –jar selenium server-standalone-2.41.0 jar –htmlsuite "\*show" "http://localhost8080/" "..\ide\allTests.html" ieResults.html."  He highlights the relevant parts of this code.*   
  
So, I just want to start here by showing you when we run this which browsers we can go off and execute. So, when I run the test you can see I get a failure and it says Browser not supported in \*show and I put show in there to say show. But this will give us the list of browsers out there, so you can run in Firefox in a mock browser, a Firefoxproxy, different versions of Firefox but it can run in chrome or several versions of IE. It can run in safari, or konqueror or all the Firefox. So, you can see there is a lot of different options, opera, webdriver all these different options can be run for these tests. So, how do I run the tests and let me go and comment this line out, and I have three sets of tests inside of here. The first that we are going to run here is running them remotely from Firefox. Now, there is a lot of different reasons we want to run it this way. And just simply regression testing, hey go off and see if the thing is still working, we might do that instead of having to open a Firefox by hand. We have an environment already set up to do that, so here my htmlSuite is pointing to the firefox executor. And then I give it the base URL I want to test off and this is the great thing. If I want to turnaround and go test the test environment or a user acceptance environment or even production, I can just point this to the base server.   
*The presenter then opens a command prompt interface to run the test. The current directory is E:\Apps\Dropbox\Selenium\Server. The presenter runs the command runTest.bat. The output includes the line  
  
java.lang.RuntimeException: Browser not supported: \*show  
  
The output also includes a list of supported browsers, some of which the presenter mentions.  
  
The presenter switches back to the runTest file in Notepad. He navigates to the line  
  
java –jar selenium server-standalone-2.41.0 jar –htmlsuite "\*show" "http://localhost8080/" "..\ide\allTests.html" ieResults.html"   
The presenter changes the line to  
  
rem java –jar selenium server-standalone-2.41.0 jar –htmlsuite "\*show" "http://localhost8080/" "..\ide\allTests.html" ieResults.html"  
  
Next the presenter focuses on the line  
  
rem java –jar selenium-server-standalone-2.41.0.jar –htmlsuite "\*firefox" "http://localhost:8080/" "..\ide\allTests.html" "firefoxResults.html"  
  
He changes the line to  
  
java –jar selenium-server-standalone-2.41.0.jar –htmlsuite "\*firefox" "http://localhost:8080/" "..\ide\allTests.html" "firefoxResults.html"  
  
He then highlights the parts of this code to explain what they mean. The base URl is http://local host8080/.*   
  
And then from there, I go and point to the suite file and this suite file, you can see it's one directory up in a folder called ide and it's stored as an HTML file. That's what we save from our ide, so let me show you this guy running. So, if I run my test again, it's going to go off and it's actually going to launch off our browser. And this is going to launch off a couple of browsers and up top it's giving us this guy running, down to the bottom it is giving us the test results inside of here. And then it ran and it closed everything, so what happened there, well what happened is we need to go back over to here and then we have our results. So, our results file was set up in our script here and our results file in this case is pointing to firefoxResults.html, it's just word wrapped here it's not actually in the next line, it's just word wrapped there. But, the Firefox results if I go and I open this up now I can see all the results everything passed and it shows you step-by-step everything that happened inside of these two test suites. And so I only have a couple of them here that is actually executing, but I have any number of tests instead of here and it's going to show me the results of what passed, what failed on all of those. I'm not limited straight to the Firefox browser then and let me show you one other before we get out of here. And that is going to be – let's do the Chrome browser, just as another one that we can do inside of here.   
*The presenter switches to the command prompt and runs the command runTest.bat again. A browser launches which lists the steps being executed in the IDE and the test runs. Inside the command prompt, the output ends with the information messages "Killing Firefox" and "Shutting down."  
  
The presenter switches to the Explorer window, which currently lists the files chromeReults, firefoxResults, and ieResults, as well as the runTest and selenium-server-standalone-2.41.0 files.   
  
He opens Notepad again to explain where in the script you find the results specified, highlighting "firefoxResults.html" in the following code:  
  
java –jar selenium-server-standalone-2.41.0.jar –htmlsuite "\*firefox" "http://localhost:8080/" "..\ide\allTests.html" "firefoxResults.html"  
  
The presenter goes back to the Explorer window and double-clicks the file firefoxResults to open it. The results display on a web page titled Test suite results. The results are split into sections, with an overview at the top. The second section shows the login.html results, the third section shows the assertEval.html results, and the final section shows the test log.*   
  
And so I'm going to save this and there you can see I have switched to the \*chrome browser, the rest of this is all the same, I am actually changing the results file just so we can save the previous one. But the rest of it is all the same, we are just changing what server we run to, so this allows us to even create tools to parameterize this and be able to run tests to say which browser do you want to test for.And so now you can see it's going to kick it off, it does the exact same thing, it runs through all the testing and then we just test it on the Chrome browser. It looks like everything passed, it may or may not have passed, but we have a feel, we can do this for any of those browsers and like the one we saw before. And then quickly use our created Selenium test suites to run across to all browsers and check that they are really being interpreted the same. It doesn't entirely limit the need to have a human go back and look at these sites, but it certainly does limit the scope by which they need to look at it. And we can test the functional items automated here and then just check the look and feel and human behavior type stuff at the last of each browser on its own. So, we can save so much time by doing this.   
*The presenter switches to the runTest file in Notepad. He focuses on the line  
  
rem java –jar selenium-server-standalone-2.41.0.jar –htmlsuite "\*chrome" "http://localhost:8080/" "..\ide\allTests.html"  
  
The presenter changes the line to  
  
java –jar selenium-server-standalone-2.41.0.jar –htmlsuite "\*chrome" "http://localhost:8080/" "..\ide\allTests.html"  
  
The presenter switches to the command prompt and enters the command runTest.bat. The Chrome browser opens and shows the test being executed.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Input Controls

Learning Objective

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

* *check input controls in the Selenium IDE*

**1. Testing HTML inputs**

When dealing with HTML inputs, we often adjust their control and presentation to customize their behavior. Being able to test the customizations in an automated way will help us ensure the quality throughout the life of the site. So, for this test here, we're going to be checking some real simple things such as taking boxes inside of our site and making them non-editable. So, by clicking this button here, you can see I cannot change the value of the site. And clicking this one again I can go and edit it again. So, that's the fist thing we want to focus on here. How do we ensure that a box is editable as we are expecting it to? To add a wrinkle to it, there's many ways to make a box non-editable. You can disable it, you can enable it, you can editable, there's several properties which will do the same thing. And so Selenium helps us within that by allowing us to add the command **assertEditable**. And this is going to allow us to go to any box here and assert that yes, we can indeed edit that. So we're going to simplify this, we're going to select directly out of here, our text box that we are going to do. And you can see, if I double-click this, it is indeed editable. If I click the **Make disabled** button and I run this check, you can see it fails, it's not editable. And so this check will ensure, if we expect a box to be editable, then we can have it be editable. And we can actually even go further and check functionality. So if there's something I might do, some value I might enter, something on the site I might press that would change that, like we have on this site, I can record that feature. So, I click this button, and then I can assure this is now **assertNotEditable**.   
*An untitled test case is open in the Selenium IDE. No steps have been recorded yet.  
  
In a separate window, a web page titled Editable forms is open in Firefox. The page contains a text box, a Make disabled button, a Make editable button, and two checkboxes – Empty Box and Checked Box. The presenter clicks the Make disabled button and then clicks the Make editable button.  
  
He then navigates to the Selenium IDE and clicks a blank row in the table to create a new step. For the new step, he types assertEditable in the Command text box. He clicks clicks the text box on the web page and hits the Select button alongside the Target text box in the IDE. In the IDE, the step now has the target id=swappable. He then double-clicks the new step in the Selenium IDE to check that it runs successfully.  
  
On the web page, the presenter clicks the Make disabled button. Switching to the Selenium IDE and double-clicking the step with the command assertEditable, it has turned red to indicate that assertEditable does not apply for this text box.  
  
The presenter checks functionality by clicking the Record button and selecting the pressable Make disabled and Make editable boxes to confirm that they correlate with the commands assertNotEditable and assertEditable.*   
  
And then we can go by and select that item again. And sometimes it's a little bit buggy in here. So, I'm just going to type it in by hand. That's a little bit of a defect in the IDE, sometimes that won't work exactly. But what I can do is type it in and hit the **Find** button. You can see yes, it's indeed showing you the item I want to, and I can double-check this. Oh see, it's indeed not editable. I can go back to **Recording**, I can click the **Make editable** button again. Alright, and then I can come down here and I can actually just copy this line. I'm going to **CTRL+C** copy this and then paste it back down inside of there. And then check, yes it's indeed editable again. And so let's go ahead and insert a statement right here. And we'll start over here with the open command. So, we'll just say open right here. And then we'll put in our page, I think that's what we need right here. And let's try that out real quick, yes perfect. And now we'll make a little test case out of this. So we test out and see, it runs almost instantly there. But I'll slow it down a minute and that should run this through there. But we can open the page, we can assert, assertEditable, make it non-editable, assert it's no longer editable, make it editable again and assert that that's working. So, again we are checking all that functionality inside of there. Just straight, should I be able to edit this? Now, there's other wrinkles we want to be able to control within this as well. As far as checkboxes go, I want to be able to assert that checkboxes either come to me as checked or unchecked. And that's not something that's always quite that easy. So what I can do is I can say **assertChecked** inside of here. And I can select the text box I want in the question, alright. And I can even assert the other one and say **assertNotChecked** inside of here. We'll do both at the same time. And then we can say that this one should not be checked.   
*The presenter uses Record button and then clicks the Make disabled button which shows that pressing the Make disabled tab has the corresponding tab assertEditable.  
  
In the Selenium IDE, a step has been captured in the Table tabbed page. The step lists the command click  and the target css=#testForm > button. The presenter clicks below the recorded step to create a new step. For the new step, he types assertNotEditable in the Command text box. He clicks the Select button alongside the Target text box and he attempts to select an interface element on the web page in Firefox. Instead the Selenium IDE highlights the entire interface. The presenter returns to the IDE and, in the Target text box, types the target id=swappable. The presenter clicks the Find button alongside the Target text box and the text box on the web page in Firefox flashes yellow.  
  
Next the presenter clicks the Make editable button on the web page. In the Selenium IDE, a step has been added to the Table tabbed page, it has the command click and the target //button[@onclick="swa. He copies the step with the command assertEditable and the target id=swappable, and pastes it below the step with the click command. The presenter clicks Insert New Command to insert a new command above the first step. He types open in the Command text box and /web site/content/editable.html in the Target text box. He then double-clicks the step. It runs successfully. The presenter clicks the Play current test case button and the test case runs successfully.  
  
The presenter drags the Playback Speed slider to a slower setting and then clicks the Play current test case button. The test case runs successfully again.  
  
The presenter creates a new step below the final step. He types assertChecked in the Command text box, clicks the Select button alongside the Target text box, and clicks the Checked Box checkbox on the web page. The new step now has the target id=filled.  
  
The presenter creates another new step below the previous step. He types assertNotChecked in the Command text box, clicks the Select button alongside the Target text box, and selects the Empty Box checkbox on the web page. The last step now has the target id=empty. The presenter then drags the Playback Speed slider to its fastest setting and clicks the Play entire test suite button.*   
  
And so one should be checked, the other should be not checked. And so let me make it really, really fast here and I will run through this and you can see, this indeed validates that they're checked or not checked. If they were off and I ran these, you can see they would fail. If they're not what they expected they would fail. But when I run the whole test from scratch, they start off the way we want them to be. And so that's the next little simple check, is checking on checkboxes. Another item I might do inside of here is as I'm editing, I might switch focus and as I switch focus, I might want to place the cursor at a specific location.   
And I can actually check that as well, I can say **assertCursorPosition**. And then I have to pick which item I want to do it for, so I'll check this one here, the swappable. And I pick a numeric value, so I'll say 0. I want the cursor to be at the start of this box. And so when I check this here, oh there's no cursor on this page because I'm leaving it right here. So, let me try that again. No, it's losing it in the tab there. But if I would do that through JavaScript, let me try this here.   
*The test case runs successfully leaving all of the steps highlighted in green. On the web page, the presenter clears the Checked Box checkbox and selects the Empty Box checkbox. He returns to the Selenium IDE and runs each of the two newly-created steps. Both of them fail. He clicks the Play current test case button and all of the steps in the test case run successfully.  
  
Next he checks cursor location by typing assertCursorPosition in the command line in the Selenium IDE. He then selects id=swappable web page text box. He types 0 in the Value box in the IDE.*  
  
Yeah it's losing it here. So for whatever reason, the IDE is not catching it. Sometimes it will do it, where you can switch over here and it doesn't leave it. But as I'm moving around, it seems to be changing that around a little bit and not letting me do – oh there you go. The actual value of 7 did not match 0. Okay, so it's something about the **Alt+Tab** I think that's doing it to me instead. So, let me do a mouse move inside of there and do it, there you go, now we see the assert value does indeed match the 0. So, when I hit **Alt+Tab**, it's leaving that box, leaving the focus. That's my fault inside of my test right there. But again, in some cases I'd have some sort of JavaScript Ajax that would focus that in. The last thing I want to show you though is just we're not limited to just these features. Let's say we have some other attribute that we know out there. I can also say **assertAttribute**. And for instance, I want to assert maybe the value of the box. It's a real simple little check. But I could be checking out anything inside of here, I could be checking on a stylesheet, I could be checking on some sort of decorative element, the class or something like that. And so I want to say, "Assert that this input box right here, its value is swapping." So, the last thing I want to show you here is how you can check any attribute inside of here.   
 *The presenter runs the step and it fails. He uses a different method of switching between the IDE and the browser, and then the step runs successfully.*   
  
So for instance, I can use the **assertAttribute** tag here and I can put in either a CSS Locator or an Xpath Locator. So, I'm going to choose an Xpath Locator. And it says, "I'm going to go to an input field with the ID of swappable." Well that's our text box up here, and I want to look at the value attribute of that. I can be looking at any attribute, anything that's valid in HTML. So, I could be looking at the name, I could be looking at the value, I could be looking at the stylesheet, I can be looking at any of those attributes inside of there. In this case, that value is going to be Swap Me. Now, this is a long way around to get the value inside of here. There's easier ways to do it. But just to show you, I can get any attribute this way. And so when I check this one, I did a capital Me here. So you can see swap lowercase me did not match that. But I change this to a lowercase me and all of a sudden, yes indeed I can find any attribute and validate its value. So, using these, I can totally control the flow and the results of my execution. And so I can get a very clear idea of how my user is seeing the site and if the site is representing to the user what they should be able to do from a flow of controlling visibility standpoint. So, use these to great effect.   
*The presenter clicks the space below the last step to create a new step. He types assertAttribute in the Command text box, css=input@value in the Target text box, and Swap Me in the Value text box. He then deletes the new step and creates another new step. He types assertAttribute in the Command text box, xpath=//input[@id='swappable']@value in the Target text box, and Swap Me in the Value text box.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

List Boxes

Learning Objective

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

* *test list box functionality in the Selenium IDE*

**1. Interacting with and validating list boxes**

As HTML gives us many options for dealing with drop-down and list boxes, we need at least as many options in Selenium to interact with and validate them. Within the IDE, we can use a series of commands to align our tests to the design of the data, and then ensure that our actions are aligned as well. So, looking at this page, you can see there's the four basic types of list boxes we can do there. The single-select drop-down box, the multiple-select boxes – and they come in three different incarnations, where basically what you see here as the value is what is stored behind the scenes. Or what you see here as the value is backed with either an ID, or a separate value behind the scenes. We actually take a peek at this real quick, if we look at the source of this page. So, you can see the first list box inside of here is just a basic list box with the options. The second one uses a "multiple" option turned on, which allows it to be, you know, opened up like this. Instead of the drop-down box, you can see all the options and select multiples of them. And then you can see I can add in for these values, and this works for both types of list boxes. For the options, I can put in an id on each one of these, or a value on each one of these. So, the text you see does not necessarily mean the text that gets saved and sent back to the web site, or what not. So, those are a bunch of different variations inside of there and so we need to align our tests towards the type of list box we're working with.   
*An untitled test case is open in the Selenium IDE. No steps have been recorded yet.  
  
In a separate window, a page titled Selections is open in Firefox. It contains a table with four columns – Drop down, Basic Multiple List Box, Multiple List Box with ID's, and Multiple List Box with Values. The Drop down column contains a drop-down list box that's currently set to Alabama. The Basic Multiple List Box, Multiple List Box with ID's, and Multiple List Box with Values columns each contain a list box that lists states in alphabetical order, with blank space to the side of it.  
The other columns are empty.  
  
The presenter opens the HTML source code for the page and scrolls through the code. The first part of the code that's visible is as follows:  
  
<div id="includedContent">  
  
<h1>Selections</h1>  
  <br/>  
  <br/>  
    
<table  border="1" >  
 <tr>  
   <th>Drop down</th>  
   <th>Basic Multiple List Box</th>  
   <th>Multiple List Box with ID's</th>  
   <th>Multiple List Box with Values</th>  
 </tr>  
 <tr>  
   <td>  
<select id="states">  
   <option>Alabama</option>  
   <option>Alaska</option>  
   <option>Arizona</option>  
   <option>Arkansas</option>  
   <option>California</option>  
   <option>Colorado</option>  
<select>  
  </td>  
    
  <td>  
<select multiple="multiple" id="statesMultiple">  
   <option>Alabama</option>  
   <option>Alaska</option>  
   <option>Arizona</option>  
   <option>Arkansas</option>  
   <option>California</option>  
   <option>Colorado</option>  
<select>  
  </td>  
    
  <td>  
<select multiple="multiple" id="statesWithIDs">  
   <option>Alabama</option>  
   <option>Alaska</option>  
   <option>Arizona</option>  
   <option>Arkansas</option>  
   <option>California</option>  
   <option>Colorado</option>  
<select>  
  </td>  
  
The presenter points out the the Drop down box corresponds to the section of code that has id="states" and that the Basic Multiple List corresponds to the section of code that has multiple="multiple" id="statesMultiple".  
  
Next he highlights id="AL"> in the id="statesWithIDs" section. He highlights value="AL" in the id="statesWithValues" section of the code. He then highlights Alaska</option> <option id="AZ">Arizona in the id="statesWithIDs" section of the source code.*   
  
So, the basic test we're going to deal with can be started with a recorder, we don't have to do anything too terribly fancy. So, we can start here and go through and select an option, so we select an option from the drop-down list. And so again, we just do a select from the id, whichever label we want off there, just as long as we're selecting one – we're good to go. Now, for dealing with these multiple boxes and you can see we can select more than one, I can select individuals, or I can hold down **Ctrl** and select multiples inside of here. Here actually I have three of them selected. So if I go back here you can see I added a selection, I removed a selection, and then I added three more selections. If each one of these sequences is important then keep it, if you don't care about the add and remove, those are just extra steps, we can go back and we can delete those and just see the ones you added. It just depends on how you want to check the behavior. If there's a behavior adding and removing, you can do that. If there is just adding and you just made a mistake, you can centralize it. And then you can see it points to the specific drop-down box, so I can select ones in these other drop-down boxes and it's going to add statements for that. So, inside of our commands, we can both add and remove selections, and do that within singles or multiples inside of here. Again I could come down and select a different value and that would change the value down here at the bottom, actually it added it here where my cursor was, but it changed that value right there. So, I don't – there is a lot of flow and ebb I can have towards what the user actions would be. It's not the end result that matters, it's the user actions. But that doesn't get us to validate any contents of these guys, so once I've done this stuff, either because I clicked on stuff or maybe because some backend functionality is preselected or Ajax is auto selected stuff, I need to be able to validate the contents.   
*The presenter switches to the Selenium IDE and clicks the Record button. On the Selections web page, he selects Arkansas from the drop-down list box. Two rows are listed in the Selenium ID. Row 1 lists the command open and the target /website/content/selection, Row 2 lists the command select, the target id=states, and the value label=Arkansas.  
  
Next the presenter selects Alaska in the Basic Multiple List Box column. He removes the selection and then selects California, Alaska, and Arkansas. He switches to the Selenium IDE, where the five steps taken have been recorded. The first added step lists the command addSelection and the target id=statesMultiple, the second step lists the command removeSelection and the target id=statesMultiple, the third added step lists the command addSelection and the target id=statesMultiple, the fourth added step lists the command addSelection and the target id=statesMultiple, the fifth added step lists the command addSelection and the target id=statesMultiple.  
  
The presenter deletes the third and fourth steps in the list using command removeSelection command.  
  
On the web page, the presenter selects Arizona from the Multiple List Box with ID's column, and then selects Alaska and Arkansas in the Multiple List with Values column. He returns to the Selenium IDE, where the steps have been recorded. Three steps have been captured in the Selenium IDE. They each have the command addSelection and the target id=statesMultiple.  
  
On the web page, the presenter selects Colorado from the Drop down column. He returns to the Selenium IDE, which lists the step which has the command addSelection and the target id=statesMultiple.*  
  
So, if I right-click on here you can see there's some asserts inside of here, but it doesn't really tell me what was selected. So, the better option is to do it by hand. And we can add in the command here to assertSelected and we can choose from here. So, if we have just the basic values inside of here, I can use, say assertSelectedLabel, if I want to do a single one. And so if I can select my single drop-down box here, you know inside of here, and so the value inside of here should be in this case Colorado. That's the one that was selected. So, if I run this you can see yes, indeed it's pointing to that, and the value inside of there is Colorado. If I change the value inside of this and run it, it's not Colorado – so it's a problem. And so for a single one, it's really easy, assertSelectedLabel. For multiples it's not that much more difficult, but we needed to use assertSelectedLabels instead. And again we select the box we want, so we can look at this box inside of here and we'll look at the whole box inside of there. And oops, it didn't pick it up, so that's okay, what I can do is, let me go back to my source code, I'll check my source code and say this guy is called "statesMultiple". So, I'm going to copy that id inside of there, and just say id=statesMultiple. So, sometimes that select doesn't want to work right but I can validate, yes it's indeed the same one inside of there.   
*On the web page, the presenter right-clicks in the Basic Multiple List Box to open a shortcut menu but does not click any of the options. The options in the shortcut menu are Back, Forward which is grayed out, Reload, Bookmark This Page, Save Page As, View Background Image which is grayed out, Select All, View Page Source, View Page Info, Inspect Element, /website/content/selection.html, verifyValue which is grayed out, storeValue which is also grayed out, assertText//select[@id='statesMultiple']/option[2] Alaska, verifyTable css=table1.1 Alabama Alaska Arizona Arkansas California, Colorado, and the Show All Available commands submenu.  
  
In the Selenium IDE, the presenter clicks the space below the last step to create a new step. For the new step, he types assertSelected Label in the Command text box. He clicks the Select button alongside the Target text box and clicks the Drop down column on the web page. The new step now has the target id=states. He then types Colorado in the Value text box.  
  
The presenter clicks the Play current selections button and the new step runs successfully.  
  
The presenter changes the entry in the Value text box for the step to Arizona and runs it again. This time the step fails. He changes the value back to Colorado, clicks Play current selections, and the step runs successfully.  
  
The presenter clicks below the newly created step to create another new step. He types assertSelectedLabels in the Command text box, clicks Select alongside the Target text box, and clicks the web page in Firefox. He then copies the text statesMultiple from the source code into the Target text box so that the target now reads id=statesMultiple.*  
  
Now, this guy has three of them selected, so how do I know how to do this? Well, you just do them in order basically. Alaska,Arkansas…and I forget the third one, let's check it, and yes…California. And if I execute this one, you can see yes they indeed did match up. If I put a space inside of here it wouldn't have matched up, it has to have exact matches inside of there, for the text. It's not super smart but it's still clever enough to able to notice there's multiples. That's this guy right here that uses the labels as the check. Now if we are doing ids, I can do that slightly differently. I can go in inside here and say assertSelectedIds – either single ID or multiple IDs. And so this one target, if we go back to our source code, we can see this is "statesWithIDs", so I'm going to copy that over, and so the ID of this guy is statesWithIDs. And the value for this one… what did we select, just Arizona, should be AZ inside of here. So let's check that one out.   
*The presenter shows the web site where the Basic Multiple List column has three states highlighted. Turning back to the IDE, he types Alaska,Arkansas,California in the Value text box for the assertSelectedLabels command and the id=statesMultiple target. The presenter clicks Play current selections and the step runs successfully.c  
  
Next he creates a new step by typing assertSelectedIDs in the Command text box, copying the text statesWithIDs from the source code into the Target text box, and typing AZ in the Value text box. He runs the step and it executes successfully.*  
  
Yip that's what matches inside of there, if I select another one inside of here and run this you can see AL,AZ is what's expected. It's not what was out there, but that's just showing you the failure. The last type inside of here as I said, is a WithValues inside of here. If I go back to the source code of this, you can see the last one is WithValues, I'm going to copy that over. And so then I can go to my last assertSelectedValues inside of there. And then again the id of this guy, I'm going to pop inside of there. And this one has selected Alaska and Arkansas, so really it's AK,AR. Is that right? Oops no space in between there, let's double check it. And the beautiful thing about this is when I run it, oh yes it indeed did match up inside of there. So, as we can see I have full control over list boxes. However they're designed, I can align to them within Selenium and be able to drive them, and validate them exactly as I need, based off my tests.   
*On the web page, the presenter selects Alabama from the Multiple List Box with IDs column. He returns to the Selenium IDE, runs the last step again, and it fails.  
  
The presenter navigates to the source code for the web page and copies the code statesWithValues. He returns to the Selenium IDE and clicks the space below the final step to create a new step. He types assertSelectedValues in the Command text box, id=statesWithValues in the Target text box, and AK,AR in the Value text box. He clicks Play current selections and the step runs successfully.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Checkboxes

Learning Objective

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

* *work with checkboxes in the Selenium IDE*

**1. Validating checkboxes and radio buttons**

On the surface, checkboxes and their cousin: the radio button, seems like a very simple tool. They're easy for the user to interact with, but add a little bit of extra work for us as a designer and a tester. Thus, when we work with radio buttons or checkboxes we need to plan ahead and understand the tools available for us within the Selenium IDE. Let's take a look. So, on this page we have a couple of checkboxes and a couple of radio buttons. Now, obviously radio buttons are linked together; checkboxes are independent. But I can validate them as we come up. So, in my tool I'm going to set this to record. I can go click a checkbox and then I can right-click and say "hey – let's verify that this box is indeed clicked on." Alright? And so that's a really simple test inside of there. And I go back and look at this: you can see I am over at the site. I click it and I verify the value is on. I can also verify that – let's say this one is supposed to be off to start with – I can verify that this is off and I could say verify that this one is on. So, 2 should be on, 1 and 3 should be off. But after I click 1, it should then be on. A very simple little verification. I can basically keep going from here or I can run my test. So I run my test. Again, it all works out just fine. It works out quite easy and simple. I can also go off and do a search on these as well. Again, I can do this through recording or whatnot…but either just through commands. I could say assert inside of here, but notice here – the search works a little bit differently. The next thing about checkboxes that they verify, is that they should just show up right away – which they don't always do. And every single type of element that's out there, but they do on checkboxes. I can also go to the additional commands and do the assert. So, I can assert the values on or off. Remember – verify: just to check, keep checking all of them. Assert will fail the first time one of those comes up as being wrong.   
*An untitled test case is open in the Selenium IDE. No steps have been recorded yet in the table panel.  
  
In a separate window, a page titled Checkboxes and Radio Buttons is open in Firefox. The page contains three checkboxes – Box 1, Box 2, and  Box 3 – and three radio buttons – Radio 1, Radio 2, and Radio 3.  
  
The presenter selects and right-clicks the Box 1 checkbox, and then selects the verifyValue id-box1 option from the shortcut menu that opens.   
  
He returns to the Selenium IDE where the executed steps have been captured and are listed in the Table tabbed page. Row 1 lists the command open and the target /website/content/checkboxes.html, Row 2 lists the command click and the target id=box1, Row 3 lists the command verifyValue, the target id=box1, and the value on.  
  
On the web page, the presenter right-clicks the Box 3 checkbox and selects the option cerifyValue id=box3 off from the shortcut menu. The presenter also right-clicks the Box 2 checkbox and selects the option verifyValue id=box2.  
  
In the Selenium IDE, two steps have been recorded. The first added step lists the command verifyValue, the target id=box3, and the value off, the second added step lists the command verifyValue, the target id=box2, and the value on.  
  
On the web page, the presenter right-clicks the Box 3 checkbox,  opens the Show All Available Commands submenu. The submenu contains the following options: open/website/content/checkboxes.html, assertTitle Web Site, assertValue id=box3 off, assertText id=box3, assertTable which is grayed out, assertElementPresent id=box3, verifyTitle Web Site, verifyValue id=box3 off, verifyText id=box3, verifyTable which is grayed out, verifyElementPresent id-box3, waitForTitle Web Site, verifyValue id=box3 off, verifyText id=box3, verifyTable which is grayed out, verifyElementPresent id=box3, waitForTitle Web Site, waitForValue id=box3 off, waitForText id=box3, waitForTable which is grayed out, and waitForElementPresent id=box3. He closes the submenu.*   
  
Now, as I continue to build my tests deeper – if I set this back to **Record** – notice through my work I can click, unclick, click reclick, click this one. Every single action I take here, all these actions that are there is recorded. Because in the IDE, it doesn't know if you were intending to click and unclick or if you were testing a click and unclick on that. It doesn't set the net result. And if I'm dealing with a text box – typically if I edit it and re-edit it, it will centralize back down to the final edit; the final text I put inside of there. With radio buttons and checkboxes – and I will show you the same thing keeping up with radio buttons, I can click these over and over again. It will show me every single one of the clicks involved inside of there. It's going to pick an X path or a name based off the simplest one I could find. And you can see, in this case, 2-3, 2-3 – it's going to pick up that way – the first one being there, but I can rename this if I want to. I can go through and rename this to be radio1 or whatever I am going to have inside of there. But note it captures every single step. If some of these steps were an error, it didn't mean to click and unclick these guys, I can go back and delete them. If I just did something wrong; if I need to check some JavaScript behavior each time one of these is pressed, it can record it exactly as it is. That works just fine. Now, again, this is recording clicks. It's going to track each click. Let's say I didn't want any of that stuff. Maybe I just want to set a value exactly what it is. Well, in that case, there is a command just for that. So, instead of click inside of here, I can do the check item inside of here. And then I can pick the target here – **Select** the target. Say I am going to select that as a target.   
 *The presenter clicks the Record button in the Selenium IDE. On the web page, he clears the Box 1 and Box 2 checkboxes, selects both of them again, and then selects the Box 3 checkbox. The presenter switches to the Selenium IDE, where all the five steps have been recorded. The first added step lists the command click and the target id=box2. The second added row lists the command click and the target id=box1, the third added step lists the command click and the target id=box2, the fourth step lists the command click and the target id=box1, and the fifth step lists the command click and the target id=box3.  
  
On the web page, the presenter selects the Radio 1, Radio 2, and Radio 3 buttons in turn. Only one  radio button remains selected at a time. He then selects the Radio 2 radio button, followed by the Radio 1 radio button.  
  
In the Selenium IDE, four steps have been addedto the Table tabbed page. The first added step lists the command click and the target xpath=(//input[@name='radio1'])[2], the second added step lists the command click and the target xpath=(//input[@name='radio1'])[3], the third added step lists the command click and the target xpath=(//input[@name='radio1'])[2], and the fourth step lists the command click and the target name=radio1.   
  
The presenter selects the step with the target name=radio1. He expands the Target drop-down list to show the various target options, including //input[@name="radio1"] and //input[4]. He closes the drop-down list.  
  
The presenter selects and removes all the click steps that follow the third and final verifyValue step. He clicks the blank space below the final step to create a new step. For the new step, he types check in the Command text box.*  
  
So, I'm going to check**Radio 2**. So let me click this. You see it's automatically going to check this guy. I can also say uncheck, okay so I can do inside of here, come into the command: uncheck. Alright, and then I can **Select** the same item and uncheck this. Now, you can't actually uncheck a box from the UI. If I click this and unclick it, click it again, I cannot unclick it – there is nothing I can do there. In the IDE, though, it will let me actually reset some of these things. The cool thing about this is that it lets me test web sites that shouldn't be able to do that. So, technically speaking if I have a default radio button selected, I should never be able to not submit a radio value back to the web site. But if I want to test my site just behind the scenes. Like – what if somebody went and hacked my site, this is what I can do there. I can go through and do that and I can submit the page and see what fails by unchecking something out there. So, the click is the most common way to capture exactly what the user can do, but the check and unchecks allows me to really get underneath the skin of testing and get past the HTML – even to the point of potentially hacking a little bit – without having to add my own JavaScript. These are all capabilities of both checkboxes and radio buttons. And you can use those in your tests however you need to as you are going off and validating these elements on your screens.   
*He clicks the Select button alongside the Target text box and, on the web page, clicks the Radio 2 radio button. The Target text for the step is then automatically set to xpath=(//input[@name='radio])[2].  
  
The presenter clicks below the newly created step to create a new step. He types uncheck in the Command text box, clicks the Select button alongside the Target text box, and, on the web page, selects the Radio 2 radio button. In the Selenium IDE, the Target text box for the new step is automatically set to xpath=(//input[@name='radio1'])[2].  
  
He demonstrates that you cannot uncheck the Radio 2 box from the UI by clicking and unclicking.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Checking Popups

Learning Objective

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

* *check popups in the Selenium IDE*

**1. Testing popups**

If the web site we're testing uses JavaScript popups, we must make special accommodations for them in our test case. While other elements can be ignored, Selenium requires you – as the tester – to declare that you expect an alert, and then allows you to confirm the contents or even interact with the alert popups. Let's take some examples of these alerts in action. So, for this page, we have three different alerts. Now, obviously, the first button doesn't do anything and it's not very interesting. So, let's take a look at the next one. So, let's say I'm recording a test and I come along and I have an alert shown. And that's all I record for the moment. Let me just show you what would happen if I play this back. You see, I get an error. There's an unexpected alert, okay? We did not accommodate for that…this test case will fail, no matter what's going to go on and happen afterwards. When the click happens, if an alert happens, we don't accommodate for it, it's going to fail. So, let's start this all over again. So, we're going to start with a new test suite and start from scratch. Now, I'm doing this because sometimes the IDE is a bit glitchy on how it records, after you start it and rerecord and stuff like that. So, let me just go ahead and start from scratch. So, what I'm going to do, is show an alert again – and this is where it gets glitchy; you can see it's not popping up again. So, I'm actually going to **Close** this out and I'm going to open it back up, just to show you in case you don't remember these commands as you go forward. So, again, I'm going to record it, I'm going to hit **Show Alert**, and now the popup happens. I'm going to confirm with the **OK** button, and then you can see I have the assertAlert command. That's asserting this text that happened inside of here.   
*An untitled test case is open in the Selenium IDE. No steps have been recorded yet.  
  
In a separate window, a web page titled Alerts is open in Firefox. The web page currently displays the word "false." Below the word are four buttons – Do Nothing, Show Alert, Show Confirm, and Show Prompt.  
  
In the Selenium IDE, the presenter clicks the Record button.  
  
On the web page, he then clicks the Show Alert button. A popup box with an OK button displays the message "popup happened."  
  
The presenter switches to the Selenium IDE, which has recorded two steps – one with the command open and the target /websites/pages/alerts.html, and the other with the command click and the target id=showAlert. The presenter clicks the Play entire test suite button and the test case runs. Below the steps, the Log tabbed lists the error message  
  
[error] Error: There was an unexpected Alert! [popup happened]  
  
The presenter selects File - New Test Suite. A dialog box with Save, Don't Save, and Cancel buttons asks if the presenter wants to save the changed test case. The presenter clicks the Don't Save button. A new untitled test case opens in the IDE.  He clicks the Record button again.  
  
On the web page in Firefox, the presenter clicks the Show Alert button. The popup does not appear as it did when he clicked the Show Alert button previously.   
  
Meanwhile, the same two steps are captured in the IDE, one with the open command and the other with the click command. He then closes the Selenium IDE and clicks the Selenium IDE button in the toolbar of the Firefox browser to reopen it..  
  
The presenter clicks the Show Alert button on the web page and a popup box with an OK button again displays the message "popup happened." The presenter clicks OK.  
  
The presenter returns to the Selenium IDE where three steps have been captured in the Table tabbed page. Row 1 of the steps lists the command open and  the target /website/pages/alerts.html, Row 2 lists the command click and the target id=showAlert, and Row 3 lists the command assertAlert and the target popup happened. The presenter clicks the space below the last recorded step to create a new step.*   
  
So, this is the way I'm going to go back and look at it. Now, just to show you, I could've added that by hand – and there's a number of them I can put out there. There's assertAlert. I can say the assert is present or not present. So, when I hit the button, if the alert was not supposed to happen, I could say, Hey, it wasn't supposed to happen, and I can continue my test case then by doing that. Now, I don't have to assert it; I can also verify the alert. I could verify the alerts there. The assert would fail the test case. The verify would cause a failure, but it wouldn't stop a test case. You could continue testing from there within your test…do your test plan inside of there. So, those are the commands. Again, if you use the IDE and the IDE's behaving, it'll automatically get added to there. If not, you can add them by hand. And so this is the first test case. So, let's go ahead and **Save** this Test Case real quick, and we're going to name this the alert test case. All right; so now we'll create a new test case, and we'll go on to the confirmation dialog. Now, the confirmation dialog does something a little bit different, in that the confirmation dialog allows us to either accept or reject what's happening here. So, if we want to record another test inside of here, I'm going to hit **Show Confirm**. Now, I can hit **OK** on this or I can hit **Cancel** on this. If I hit **OK**…we've seen that, it's just like the other alerts. Let me just show you real quick, hitting **Cancel** inside of here. So, when I hit **Cancel**, I still have the assert here, but now I have assertConfirmation. And then, up above, since I hit **Cancel**, you can see chooseCancelOnNextConfirmation. I don't do that when it pops up, because assertConfirmation is kind of an interactive call in response to an alert expected to be there. So, what I have to do is…ahead of time, is say Hey, I'm going to cancel this next one; I'm not going to choose that one appropriately.   
*For the new step, the presenter types assert in the Command text box. A drop-down list appears with several options, including assertAlert, assertNotPresent, assertAlertPresent.  
  
The presenter removes assert and types verify in the Command text box. A drop-down list lists the options verifyAlert, verifyAlertNotPresent, and verifyAlertPresent. He removes the verify text from the Command text box.  
  
The presenter selects File - Save Test Case. In the Save as dialog box, he types alert in the File name text box. He then presses Enter to save the test case. The test case alert now appears in the Test Case pane.  
  
In the Selenium IDE, the presenter selects File - New Test Case to open a new, untitled test case. The presenter clicks the Record button and, on the web page, clicks the Show Confirm button. A popup box with OK and Cancel buttons contains the message "confirm this." The presenter clicks Cancel.*   
  
And so, that's the next wrinkle we can throw in there for the confirmation dialogs. Had I chosen just to hit **OK** inside of there, I can either take this out or I can just not hit this. Then it's just going to say, Okay yes, a confirmation happened, and I'm going to hit **OK**. That's the default behavior inside of there. All right, so let me go ahead and **Save** this one real quick; and this is going to be the confirm, and then let's add one more out there. So, I can say **New Test Case**, and then I'm going to record this one – and let's look at the Prompt. So, now when I hit Prompt; inside the Prompt, it's going to ask me for a value – a single value, but it asks me for some value – and so I can say I am entering this prompt. Okay; and so within this, I again have to tell it when I get the next prompt, this is the text I want you to put out there. And it's going to add that text; it's going to hit **OK** and, in our case, the JavaScript you can see actually validates this up front. So, I come back to this bottom step here…I could actually put a step out here to assert that text is out there, and see that the text is entered here and then, actually exists, and shows up there. All right; so let me go ahead and **Save** this Test Case again real quick, as prompt. And now, I have a suite that shows us all the different steps I can put out here. And so one quick **Run** here…you can see…actually, I ran into an issue on this guy. I have an extra step inside of here; that's my bad. Okay, I deleted that and so let's do that one more time – and now you can see I have a test suite that can test all different types – these prompts. I have to be proactive; I have to know about them and plan for them. But, otherwise, it's pretty simple to handle popups inside of JavaScript.   
*In the Selenium IDE, four steps have been added to the Table tabbed page. The first added step lists the command open and the target /website/pages/alerts.html. The second step lists the command ChooseCancelOnNe... The third row lists the command click and the target id=showConfirm, and the fourth added step lists the command assertConfirmation and the target confirm this.   
  
He highlights the assertConfirmation command on the Table tabbed page.  
  
The presenter then expands the Command column to reveal the full command for row two and he highlights this command, which is  chooseCancelOnNextConfirmation. He highlights the command assertConfirmation as the default behavior when OK is clicked.  
  
He then saves the test case as confirm. It appears in the Test Case pane.  
  
He creates a new untitled test case using the File-New Test Case option.  
  
The presenter clicks Record and, on the web page, clicks the Show Prompt button. A popup box with OK and Cancel buttons contains an Enter a value text box. The presenter types "I am entering this prompt" in the text box and clicks OK.  
  
The text the presenter entered appears above the Do Nothing and Show Alert buttons on the web page. The presenter highlights one of the steps that now appears in the IDE. It has the command answerOnNextPrompt and the target I am entering the prompt.  
  
The presenter selects and right-clicks the prompt text on the web page, and selects the option assertText id=answerText I am entering this prompt from the shortcut menu.  
  
In the Selenium IDE five steps have been added to the Table tabbed page. Row 1 lists the command open and the target /website/pages/alerts.html, Row 2 lists the command answeronNextPrompt and the target id=showPrompt, Row 3 lists the click command and the target I am entering this prompt, Row 4 lists the command assertPrompt and the target Enter a value, and Row 5 lists the command assertText.   
  
The presenter selects File - Save Test Case. In the Save as dialog box, he types prompt in the File name text box and presses Enter. Prompt  now appears in the Test Case pane.  
  
In the Selenium IDE, the presenter highlights all the steps and then clicks the Play entire test suite button. An error occurs in the alert test case. The presenter selects and deletes an extra step in the test case. He clicks the Play entire test suite button again and the test runs successfully.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Multiple Windows

Learning Objective

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

* *work with multIple windows in the Selenium IDE*

**1. Creating multiple-window tests**

It might be a little annoying sometimes, but it's not uncommon to find sites that utilize multiple windows to display information or even to allow users to conduct multiple simultaneous activities. As most tests assume all actions happen concurrently on the same page, we need to add into our toolbox the ability to swap between windows to fully test when there is multiple windows involved. So, in this case, we are going to record a little test. We are going to set it up to record and the recording normally works pretty well. So, I'm going to go here, I'm going to open a window. You can see it opens up the second window here. And I'm just going to go through here and validate, "yes, this text is on this other window." Okay, a very simple little test. So as I come through here, notice it has to add in the option here to selectWindow and it's going to throw that in between our test here. So, after we waited for the PopUp window 1, it needs to go through and select a window before I can go and run this test. So, if close this window again, and I go and run my test here – just to make sure things are all going on very happily – you can see yes, it indeed does everything perfectly fine. Good, so I can keep recording my test. If I come back to this page now, I'm going to put in some text here. So, send this to the other window. All right, and then I can hit **Send Text**. And if I come back inside of here, you can see it comes back and it looks pretty good. It is saying I'm going back to the main text and I am sending those things over there. And then I can come back over here and I can make sure that text is out there. So, I can say assert that text is indeed back there.   
*An untitled test case is open in the Selenium IDE. No steps have been recorded yet.  
  
In a separate window, a web page titled Windows is open in Firefox. The page contains a text box, which contains the text "Window1", and an associated Open Window button. There is also a text box with the default text "Send this updated text", and an associated Send Text button.  
  
The presenter clicks the Record button in the Selenium IDE and, on the web page, clicks the Open Window button. A new web page, titled Another Windows, opens in the browser. It displays the text "I am here to be updated."  
  
The presenter selects and right-clicks the text, and selects the option assertText id=textLocation I am here to be updated from the shortcut menu that opens.  
  
In the Selenium IDE five steps have been captured in the Table tabbed page. Row 1 lists the command open and the target /website/content/windows.html.Row 2 lists the command click and the target id=makeVisible. Row 3 lists the command waitForPop, the target Window1, and the Value 3000. Row 4 lists the command selectWindow and the target name=Window1. Row 5 lists the command assertText, the target id=textLocation, and the value I am here to be updated.   
  
The presenter selects the step with the command selectWindow. He then clicks the Play current test case button and the test case runs successfully. Then he returns to the web page and replaces the text in the second text box with the text "Send this to the other window." He then clicks the Send Text button.  
  
In the Selenium IDE, three additional steps have been recorded: the first lists the command selectWindow and the target name=mainWindow, the second lists the command  type, the target id=sendTextValue, and the value Send this to the other window, the third lists the command click and the target id=sendText.   
In Firefox, the page titled Another Windows displays the text "Send this to the other window." The presenter selects and right-clicks the text, and selects the option assertTextid=TextLocation Send this to the other window from the shortcut menu.*   
  
So, again, it goes back to window 1 and all of my test is fine. Now, sometimes when I'm working through the IDE there is a bug in there, that this select window won't show up. If that's the case, then you have to go through and add that by hand. You have to make sure it's out there. In certain window types that will happen and especially if the...the windows are not named properly. It's not truly given unique names out there, then there could be some problems inside of doing that. So, again, my test right now – oh, look at that – it fails at this point when it goes off and says select name=mainWindow. Well, as I said – there is some bugs inside there. It's not all working perfectly and that's because there is a couple of main windows open. I have to really be careful when I do this that I start with the main browser. And even there, it's…it's failing on main window. So, if it doesn't like this one, I can change it. I can say, instead of main window, you can see there is some other options: there is a title. If it has a unique title I can say title= and the other one's title is Web Site. And so we can try this out, let's see if this works. Ah! See now my…now that one work, passed. And when selected the window, it went by there and it worked just fine. So, it didn't like that other one for whatever reason and it worked on "Web Site" just fine. It's going to allow me to go off and select those items.   
*In the Selenium IDE, two steps have been added to the Table tabbed page. The first step lists the command selectWindow and the target name=Window1, the second lists the command assertText and the target id=textLocation.   
The presenter clicks the Play current test case button and an error occurs at the step where command is selectWindow and target is name=mainWindow.  
  
The Log tabbed page is open below the Table pane, and includes the line (in red):   
[error] Window does not exist. It directs the user to a web page with potential workarounds if this looks like a Selenium bug.  
  
. The Referenced tabbed page is displayed, and it includes tips on how locators can be used to specify window objects and a list of possible locators that can be used, such as title and name.   
  
The presenter selects the command selectWindow step, and changes the Target to title=Web Site. He clicks the Play current test case button and the test runs successfully.*  
  
So, the last little thing here is just to know that this feature is really useful. It's occasionally a little bit buggy, but as long as I manage the windows and manage the names, it's going to work out just fine. I might need to go back to my developers and say, "hey, I can't automate this properly" or if you are the developer, you can fix it yourself. I need you to tweak the JavaScript. I need you to tweak something inside of them. I need you to make sure the windows are named uniquely or appropriately inside of there, because – again – I can go through inside of my test and I can kick this off to Record again. And come back to here and I can say this is now Window2. And Open Window and this one should work fine, because – again – it's allowing me to create a unique window out there. And then I can go through and validate that: yes, this is truly the unique test that's out there. And go on from there. So, yet again, we run our test. Oh, the mainWindow here. Again, I have to go back and change this to be title=Web Site and then I can try. And let me close these windows so it doesn't get confusing. And then everything runs just fine again. You can see it has both windows open at this point, it's updating them both. And I can test all my features, but this feature and ID is very powerful. But does require a little bit of understanding, a little bit of tweaking – perhaps until they fix it someday. Or, at least you have to know how to make your site so that way things are truly unique. It's not smart enough to be able to determine between the windows, because – really – the only determining factor is either the name or the title and if those are redundant and there is multiples of them, it doesn't know what to do. And so it would rather fail than do something wrong. So, again, that's how we deal with multiple windows. So, have fun testing that when you need to and deal with multiple windows.   
*The presenter clicks the Record button. On the Windows web page, he changes the text in the first text box to Window2 and clicks the Open Window button. The page titled Another Windows opens. It displays the text "I am here to be updated."  
  
The presenter switches to the Selenium IDE, where steps have been added to the Table tabbed page.  One command is type with target id=windowName, another is command click and target id=makeVisible, and a third is command waitForPopup and the target Window2.  
  
On the web page titled Another Windows, the presenter selects and right-clicks the text "I am here to be updated", and selects assertText id=textLocation I am here to be updated in the shortcut menu that opens.  
  
In the Selenium IDE, a step with the command assertText and the target id=textLocation has been added to the Table tabbed page. The presenter clicks the Play current test case button. The test encounters an error at the second step, which lists the command selectWindow.  
  
With the selectWindow step selected, the presenter changes the entry in the Target text box to title=Web Site. He closes the web pages opened due to the testing process, switches to the IDE, and clicks the Play current test case button. The test case runs successfully.  
  
The presenter switches between the two web pages opened during the course of running the test to show that they have both been updated.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Drag and Drop

Learning Objective

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

* *use drag and drop in the Selenium IDE*

**1. Testing drag-and-drop actions**

Using advanced JavaScript on our pages allows designers to imagine screens where users can, very visually and hands on, manipulate information. One key to this ability, is the chance to drag and drop screen elements around the page. In order to test this though, we must provide basic mouse actions as part of our Selenium tests. So, within the IDE, it will not actually record drag-and-drop actions. So, if I have my brand-new test here – and I have my page here, I'll show you how it works with the Record on. So, this'll allow me to drag an element around the screen and then drop it. And I have a couple of locations, just to kind of tell you some spots you might want to drop it. Or, in this case, I can just drag it wherever I want to on the page. But here, notice, nothing was recorded. Nothing at all was recorded as part of my test. This doesn't start recording until it notices some key actions inside of there. So, if I want it to drag and drop, I have to code that by hand. So, let me start here; let's just do the opening command, just to make sure we are starting with a nice clean window there – and then we're going to open that page. And, from there, there is a dragAndDrop command. Now, there's also mouse commands. I can say mouseDown, mouseDownAndWait, mouseDownAt…there's all sorts of mouse commands I can do directly. I can string these together to achieve drag and drop or, what I can just do is drag and drop itself, which will then take the mouse to that location, click, hold the mouse down, move it to another location, release the mouse, and then go from there. So, I can do all that, or I can just use dragAndDrop.   
*An untitled test case is open in the Selenium IDE. No steps have been recorded yet.  
  
In a separate window, a web page in Firefox contains two empty panes and an image of a block that's subdivided into four quadrants, each a different color. The top-left quadrant is colored red, the top-right is green, the bottom-left is purple, and the bottom-right quadrant is yellow. First, the presenter drags the colored block to the center of the empty pane on the left, then to the center of the pane on the right, and then to four other locations on the page. Finally he drags it back to its original location.  
  
The presenter switches over to the Selenium IDE and no steps have been recorded yet. The presenter clicks an empty row in the test case table on the Table tabbed page to create a new step. With the step selected, he types open in the Command text box to open a new clean page in the test. Next the presenter selects the colored quadrant image by adding the code http://localhost:8080/web site/pages/drag.html as an argument in the Target text box. Then the presenter clicks below the step to create another new step. He types drag in the  Command text box and a drop-down list opens to present suggestions for completing the command. The command options include dragAndDrop, dragAndDropAndWait, and dragAndDropToObject.  
  
The presenter then deletes the drag-and-drop command and starts typing mouse in the Command text box. This opens a drop-down list of mouse command suggestions, including mouseDown, mouseMove, and mouseOut. The presenter then retypes dragAndDrop in the Command text box to add this command to the test case.*  
  
Now, finding the element to drag and drop, I don't have to know the XY location. I can start by just selecting which item I want to drag and drop. So, again, I can come to my picture down here. That has an ID of drag1 inside of there; that's the guy I'm going to drag around. And so I can drag and drop it to any location on the screen; it's relative to where I'm at now. So, I want to move this, let's say, 10 forward – that's to the right, here. And we're going to move it up, so that's going to be a negative. Let's go 250 up, because that's about the size of the picture there, and so move it up this way. So, let me go ahead and **Play** my scenario here and we can see what happens. Boom, it moved it up. Now, we don't actually see it drag because it moved so quickly. If we do it slow, I don't think that's going to make much of a difference there. No, you see it just pops over there, but it's the net rate there of being able to move up inside of there. So, I can continue to drag this guy around the screen if I want to. So I can move it; drag and drop it here. I can then, as I said, do another dragAndDrop to some other location on the screen. I have to go back and select the item again. And then, let's say I want to move this 300 to the right and maybe 10 down. That would be a little bit down from what it was. I can **Play** the whole thing from the beginning; and you can see, it skips right by the middle location, but then it's moved it over the right. I happen to be able to hit these boxes nicely.   
*For the new drag-and-drop step in the Selenium IDE, the presenter clicks the Select button to choose a Target element, which is then entered as an argument associated with the drag-and-drop command. Switching over to the Firefox window, he clicks the block of colored quadrants on the web page. Switching back to the Selenium IDE, this results in the Target  being set to id=drag1. The presenter then enters the values 10,-250 as arguments in the Value text box to specify the coordinates of the position that he wants to move the element to. Then he clicks the Play current test case button. The colored block moves to the center of the empty pane on the left.  
  
In the Selenium IDE, the presenter then drags the Playback Speed slider to its slowest setting and then clicks the Play current test case button again. The colored block is reset in its original position and then jumps to the empty pane on the left.  
  
Back in the Selenium IDE, the presenter adds another new step to the test case. Again, he types dragAndDrop to add another drag-and-drop command, then clicks the Select button to switch to Firefox and he clicks the image of the colored blocks on the web page. He then types 300, 10 in the Value text box to set the location he wants to move the element to . The presenter clicks the Play button and the colored block moves to the center of the empty pane on the right of the screen.*   
  
Now, in theory, Selenium IDE gives us the ability to drag and drop to a location. I could, inside of here, say id=div2. Now, div2 is this element inside of here. I can actually do it…an **Inspect Element** out here, just to show you. It's div2; that's the box that we're looking at, that target box that I'm trying to hit. Now, this is legally syntactically available. If I ran this, what would end up happening is, there's a bug in Selenium and this would just hang in the IDE…it….this'd just hang, it wouldn't go anywhere. It actually works just fine in the coded version of it. It just doesn't work in the IDE for right now – for whatever reason. So, I won't show you that, but I want to show you the capability of that. The last thing I do want to show you though, is…remember, drag and drop is not inherent on pages, it has to be coded. If I look at the Page Source over here, there's a bunch of JavaScript that is enabling my drag-and-drop functionality.   
*For the last recorded drag-and-drop step in the IDE, the presenter now replaces the location value by entering id=div2 in the Value text box. Back on the web page in Firefox, the presenter points to the empty pane on the right of the web page when explaining that div2 represents this element.  
  
The presenter right-clicks the div2 element and then selects Inspect Element from the shortcut menu. This opens the Inspector tab and displays the source code for the element.  
  
When the presenter moves the cursor to the div2 element on the web page, a tag containing the value div#div2.target appears highlighted onscreen. The presenter then closes the pane at the bottom of the web page.  
  
Finally, on the web page, the presenter right-clicks to open a shortcut menu and then selects View Page Source. The source code for the page opens in a separate window. It contains the following source code including the JavaScript that has been added for the drag-and-drop functionality:  
  
<style>  
<!--  
.target  
width: 250px;  
height:  250px;  
padding: 10px;  
border: 1px solid #aaaaaa  
}  
-->  
</style>  
<style>  
<!!--  
.dragme{position:relative;}  
-->  
</style>  
<script language="JavaScript1.2">  
<!--  
  
var ie=document.all;  
var nn6=document.getElementByID&&!document.all;  
  
var isdrag=false;  
var x,y;  
var dobj;  
  
function movemouse (e)  
{  
 if (isdrag)  
 {  
   dobj.style.left  = nn6  ? tx + e.clientX - x  :  tx + event.clientX - x  
   dobj.style.left  = nn6  ? ty + e.clientX - y  :  ty + event.clientY - y  
   return false;  
 }  
}  
  
function selectmouse (e)  
{  
 var fobj       = nn6 ? e.target : event.srcElement;  
 var topelement = nn6 ? "HTML" : "BODY";  
   
 while (fobj.tagname != topelement && fobj.className  !="dragme"  
 {  
   fobj = nn6 ? fobj.parentNode : fobj.parentElement;  
 }  
   
 if (fobj.className=="dragme")  
 {  
   isdrag = true;  
   dobj = fobj;  
   tx = parseInt(dobj.style.left+0);  
   ty = parseInt(dobj.style.top+0);  
   x = nn6 ? e.clientX : event.clientX;  
   y = nn6 ? e.clientY : event.clientY;  
   document.onmousemove=movemouse;  
   return false;  
 }  
}  
  
document.onmousedown=slectmouse;  
document.onmouseup=new Function("isdrag=false");  
  
//-->  
</script>  
<div id="status"></div>  
<table>  
   <tr>  
       <td>*   
  
So, if I use drag-and-drop functionality, right now Selenium only recognizes basic drag-and-drop functionality. Right here, if I look at this Page Source – this is HTML5 drag-and-drop capability. So, inside of HTML5, I can mark an item as being draggable, and then have different drag events that are pulled from this. Now, if I run this as drag5 instead…I can just show you real quick. In HTML5, I can drag and drop, I can put it on here, and you can see it's actually…I put a little bit of JavaScript where I'm telling it which box it's popping into – left or right box inside of there. So, if I run the same code on the HTML5 right here, and…actually, let me take this out because I don't want it to die. But just simply drag it up into the first one; nothing happens. You can see, it didn't say left or right – it didn't move it. Right now, Selenium is not HTML5 compatible. So, just to be aware of in your testing. If you're going to do it this way, you're probably going to have to test that part by hand – if your drag and drop is enabled by HTML5. Past that though, drag and drop is a feature you can certainly automatedly test as part of your test suite.   
*In Firefox, the presenter opens a new URL by entering localhost:8080/web site/pages/drag5.html in the address bar. The web page that opens shows the colored blocks image and empty panes, but the borders around the empty panes display in red rather than gray. The presenter right-clicks the interface and selects the View Page Source option from the shortcut menu.  
  
A file containing the source code for the page opens in a separate window. It contains the following code:  
  
<style>  
<!--  
.target {  
width: 250px;  
height: 250px;  
padding: 10px;  
border: 1px solid #aaaaaa  
}  
-->  
</style>  
  
<script>  
function allowDrop(ev)  
{  
   ev.preventDefault();  
   }  
     
function drag(ev)  
{  
   ev.dataTransfer.setData("Text",ev.target.id);  
}  
  
function drop(ev, name)  
{  
   ev.preventDefault();  
   var data = ev.dataTransfer.getData("Text");  
   ev.target.appendChild(document.getElementByID(data));  
   document.getElementByID('status').innerHTML = name;  
}  
</script>  
</head>  
<body>  
  
<div id="status"></div>  
<table>  
   <tr>  
       <td>   
The presenter highlights HTML5 code for drag-and-drop functionality in the source code. Then the presenter switches to the Selenium IDE and selects the open command in the test case. He changes the entry in the Target text box from http://localhost:8080/web site/pages/drag.html to http://localhost:8080/web site/pages/drag5.html to prepare to run the test on the HTML5 web page instead.  
  
Next, the presenter returns to Firefox and manually drags the colored block to the center of the pane on the left. The word "left" appears above the pane. The presenter drags the colored block to the pane on the right, and the word "right" appears above the pane. The presenter moves the colored block back and forth between the two panes to demonstrate the HTML5 drag-and-drop functionality, and each time the block is in one of the panes, the relevant word displays.  
  
Finally, the presenter switches to the Selenium IDE and removes the last step. He then  Plays  the HTML5 test case but nothing happens on the web page in Firefox as Selenium is not HTML5 compatible.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Clicking Links

Learning Objective

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

* *testing links in the Selenium IDE*

**1. Testing basic link navigation**

The backbone of navigating between pages is the link. It's inevitable that somewhere in our testing we're going to want to navigate between pages using links or the deluxe option in the image map. So, while recording works really well for creating these flows, we also want to understand the exact steps being captured behind the scenes so we can control the flow of our testing exactly. So, let's start by recording a quick sequence of links inside of here. So we're going to have our **Record** on, we're going to start on this page and we're going to go let's say to the **news** page. And then we can go check the **calendar**, then we can go to the **Login** and we can test here. We can do the admin and the **Password** inside of here, a **Login** and we come back to this page. Now, I'm going to click one more link just to complete my test. I'm going to click the **Logout** link inside of here, just because if I don't do that, then it's going to actually screw up my test. Now, coming back to my test script, you can see I recorded it. I open the page and then I click and I wait. And this is important. clickAndWait defines hey I'm going to click on this thing and I'm going to wait for the whole page to refresh. So, I do that for the **news**, do that for the **calendar**, do that for the **Login** page, type in some items, click the button again and wait, because the page is going to refresh itself, and then I'm going to log out. So, let me go ahead and run that whole sequence, perfect… logs in, logs out. This is important because if I forget this last item, if I take that last item out and I run it, it'll work fine the first time, but the second time it will fail because the link login is not there. Because when I say link=login right here, it's looking for the text that's in this link. It's not looking at the href, it's not looking at where it's going, it's looking literally for that text out there. So, if I don't keep here, I'm going to record this last one again. I don't keep this login, logout at the end I can't rerun my test every time.   
*An untitled test case is open in the Selenium IDE. No steps have been recorded yet.  
  
In a separate window, a web page titled Image Map is open in Firefox. The page contains a block that's subdivided into four smaller squares, each a different color. It also contains a menu with home, news, calendar, contact us, login, and no popups links.  
  
The presenter clicks the news link and a web page titled News opens. The main part of the page is blank, but the same menu links are available.  
  
The presenter clicks the calendar link. This opens a page titled View Calendar, which displays a calendar for the current month.  
  
The presenter clicks the login link and a login page opens. It contains Username and Password text boxes, and a Login button. The presenter types admin in the Username text box, types a password in the Password text box, and clicks the Login button. The page titled Administrator Home opens. On the page, the login link in the menu has been replaced by a logout link. The presenter clicks the link and is returned to the login page.  
  
Next the presenter switches to the Selenium IDE, where all eight steps have been captured in the Table tabbed page. Row 1 lists the command open and the target /website/content/imagemap.html, Row 2 lists the command clickAndWait and the target link=news, Row 3 lists the command clickAndWait and the target link=calendar, Row 4 lists the command clickAndWait and the target link=login, Row 5 lists the command type, the target id=usernameInput, and the value yes, Row 5 lists the command type, the target id=passwordInput, and the value test, Row 6 lists the command clickAndWait and the target name=submit, Row 7 lists the command clickAndWait and the target link=logout.  
  
The presenter now selects the open command and then selects the clickAndWait command and waits for the web page to refresh. He clicks the Play current test case button and the test case runs without any errors. As the test runs, Firefox navigates from the Image map web page to the News,  and Calendar pages.  
  
The presenter selects and then deletes the last step in the test case - the clickAndWait command to logout. Then he clicks the Play current test case button to run the test again and it runs successfully. As the test runs, Firefox navigates from the Image map web page to the News, Calendar and login pages, the username and password appear in the relevant textboxes on the login page and the Administrator Home page opens.  
  
However, when he clicks the Play current test case button a second time, the test case encounters an error at the step with the target link=login as this link is no longer there.   
  
Finally, on the web page, the presenter clicks the logout link to return to the login page and reset the test. In the Selenium IDE, the clickAndWait command with the target link=logout step has now been restored.*   
  
It doesn't – it's just opening the page, it's not resetting my session, it's not resetting anything that's on the server side going on inside of there. So again, this time I've kind of screwed up my test, the sequence, the timing is all off, but there I run it one time, I'm also happy again. I can go – I can go do this. Selenium IDE is really, really sensitive to timing which is I have to be, why I have to be very aware of clicking and waiting. Alright, so there's some basic navigation. I can check things, I can go back and forth. And in this case, I was clicking a bunch of links but there is nothing that say, I couldn't do steps in between. Working with image maps really is essentially the same. If I go back to my original site, I go back to my recording. Here in this case my image map isn't going to navigate me to another location, it's just going to calls some JavaScript. And when I click on this the JavaScript is going to go through and it's going to open up this box right here and put the color that I just clicked on. So, I can actually navigate and check out this color. So, if I look over here you see I am going to click and wait and then I'm going to assert the text. I'm going to keep recording here and I'm going to go green, purple, yellow I'm going to let it go at that. You can see I'm clicking on all these things. Now, notice here, the target of what I'm clicking on here is called an id=red. You might expect on the image map, an image map I would be clicking on 10, 25 or wherever it is. You notice no matter where I clicked on here, it did not pick an XY location. Well that's because I don't need to as a Selenium IDE tool. If I look at the source of this guy, if I go through and inspect it, you can see I have an image map defined right here, as a map.   
*The presenter replays the recorded test case and the test encounters the same error as before. As the test runs, Firefox navigates from the Image map web page to the News and Calendar pages.  
  
He clicks the Play current test case button once more and this time the test runs successfully. As the test runs, Firefox navigates from the Image map web page to the News, Calendar and login pages, the username and password appear in the relevant textboxes on the login page and the Administrato Home page opens.  
  
Next the presenter removes every step after the   open command in the test case. He switches to Firefox where the page titled Image Map is open. He clicks the top left-hand corner of the colored square and the word red appears above the square. He highlights the word red, right-clicks to open a shortcut menu, and then clicks the assertText id=color red option in the menu.   
  
The presenter switches back to the Selenium IDE and two steps have now been added below the open command to the Table tabbed page. The first of the new steps lists the command clickAndWait and the target id=red, and the second step lists the command assertText, the target id=color, and the value red.   
  
Back in Firefox, the presenter clicks the top right-hand quadrant of the square and the word green appears above the square and the word green appears onscreen, then he clicks the bottom left-hand quadrant of the square and the word purple appears, and then he clicks the bottom right-hand quadrant and the word yellow appears. He switches to the IDE where three additional steps have been added to the Table tabbed page. The first new step lists the command clickAndWait and the target id=green, the second added step lists the command clickAndWait and the target id=purple, and the third added step lists the command clickAndWait and the target id=yellow.   
  
The presenter goes back to Firefox, right-clicks the colored square and clicks Inspect Element.  
The Inspector panel opens at the bottom of the Firefox interface and it displays the following code:  
  
map id="myMap" name="myMap">  
 <area id="red" target="\_self" style="outline:none;" coords="0,0,91,101" shape="rect" href="?color=red" title="Red" alt="red"  
 ></area>  
 <area id="green" target="\_self" style="outline:none;" coords="92,0,200,103" shape="rect" href="?color=green" title="Green"  
 alt="green"></area>  
 <area id="purple" target="\_self" style="outline:none;" coords="0,100,95,200" shape="rect" href="?color=purple"   
 title="Purple" alt="purple"></area>  
 <area id="yellow" target="\_self" style="outline:none;" coords="91,102,200,200" shape="rect" href="?color=yellow"   
 title="Yellow" alt="yellow"></area>  
</map>*   
  
And it's defined as a bunch of areas. Now those areas might be rectangles, they maybe be circles, it might be free-form. But my tool is not smart enough to see the picture behind. It can't tell you if the image map is properly aligned to that, that's somebody else's tool. So instead, what I'm going to do is go to the area you clicked on, whichever one you happen to click on, and I'm going to target that ID of that area or whatever other identifier I can get out of this. In this case, I still have ID, so it works out very easily. And then I'm going to jump straight to that. I'm going to jump straight to that link and it's going to work really well that way. Now, when I go and run this test it's a little bit goofy inside of here. Because when I say clickAndWait, well it's going to wait but it's going to go through and it's looking for that text. But when is it going to wait for? Well the page is not refreshing. So, I'm into this little bit of a time sequence where it's going too fast. If I slow my test down here and go step-by-step, it'll actually work just fine. Because of the JavaScript behavior, a little bit Ajaxy inside of there. Sorry I don't know what I recorded there accidentally. But that little Ajaxy type behavior inside of there, if I go too fast, it'll actually fail my test. So, the click option in itself, and the clickAndWait option itself only handles general page navigation. It's only going to handle when the whole page refreshes. If you want to deal with Ajax based behaviors, where I am hitting JavaScript or a JavaScript sitting at server and reloading part of page, that's outside the scope of the click and the clickAndWait. You'd have to look up other mechanisms to do that. Now we won't talk about that here. But we can test basic link navigations when I'm refreshing entire pages and I'm doing navigation that way, underneath my IDE using these simple commands.   
*The presenter closes the Inspector at the bottom of the Firefox interface and then goes back to the Selenium IDE where a step has been added. The step lists the command click and the target id=toolbox-close. He clicks the Play current test case button. The test fails at the step with an assertText Command. As the test runs, the Image map page shows the text "red" above the colored square. The presenter drags the Playback Speed slider closer to its slowest speed setting and then clicks the Play current test case button again. The test fails at the step with a click Command and a Target of id=toolbox-close. The presenter removes the step as it was added as an error and the test will now run successfully.*

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

Exercise: Basic Automated Web Testing using Selenium

Learning Objective

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

* *check page elements in the Selenium IDE*

**1. Exercise overview**

It's time to put what you've been learning into practice. See how you do with the exercise that follows.

You have been asked to perform some basic tests on a new area of your company's web site.

This involves the following tasks:

* running a basic test
* finding and matching page elements
* asserting a variable's presence
* testing in a different browser

**2. Web testing automation in Selenium**

Question

You need to begin running basic tests on your site, and for the first pass you just want to see all of the conditions that are not being met.  
  
Which command should you use to do this on one pass?

**Options:**

1. verifyText
2. assertText

Answer

***Option 1:****Correct. The verify command will verify the condition, and if it’s not match an error will be displayed in the Log area and the test will proceed along to the next commands.*

***Option 2:****Incorrect. The assert command will stop remaining script execution if a condition is not met.*

**Correct answer(s):**

1. verifyText

Question

You are evaluating ways to find and match the elements of your page that you needs to interact with.  
  
Match each locator type with its characteristics.

**Options:**

1. Id
2. CSS
3. DOM
4. XPath

**Targets:**

1. Will work even if the structure of the page changes
2. Allows for selection of elements by their surrounding context
3. Relies on the structure of the page
4. The standard navigation tool for XML

Answer

*The Id locator type works with the id and name attributes of your html tags. It doesn’t on the structure of the page and will work even if it changes but as it easily matches several elements (e.g. id, name) you need to be careful when using it.*

*Using CSS selectors to find elements on the page is a fast and widely used location method which allows for selection of elements by their surrounding context. Use of this method is dependent on you knowledge of CSS.*

*Using DOM to locate elements that match the JavaScript expression referring to that element does rely on the structure of the page. But using JavaScript also allows you to build dynamic locators.*

*You can use this method anywhere there is XML. You are relying on a browser’s XPath implementation which may not be complete, but the locators are very precise.*

**Correct answer(s):**

Target 1 = Option A

Target 2 = Option B

Target 3 = Option C

Target 4 = Option D

Question

During your site testing you need to assert the presence of a variable, totalBeforeDiscount. As that variable will be changing every time you run the test you need to assign the variable a value.   
  
Enter the correct code on the Value field to assign the variable to a value.

**Code**  
INSERT THE MISSING CODE

Answer

*Using the ${variable\_name} method gives us access to the storedVars map. So, in this case Selenium will take the variable totalBeforeDiscount and translate it to a value.*

**Correct answer(s):**

1. ${totalBeforeDiscount}

Question

You’ve created a suite of tests, saved as mySiteTests.html, for a web site using Selenium IDE in Firefox. You now want to test the site in Chrome using the Selenium Test Server, and save the results in an HTML file called testsResultsCH.  
  
Complete the command line code that will allow you to do this.

**Code**  
java -jar selenium-server-standalone-2.41.0.jar -htmlsuite "INSERT THE MISSING CODE" "http://localhost:8080/" "..\mySiteTests.html" "testsResultsCH.html"

Answer

*By specifying \*chrome, in the command line code, the test will launch the Chrome browser, run the tests, and save the results in the specified HTML file.*

**Correct answer(s):**

1. \*chrome

[Back to top](http://xlibrary.skillport.com/courseware/Content/cca/sd_slwt_a01_it_enus/output/html/course_transcript.html#top)

© 2017 Skillsoft Ireland Limited