Modifying User Actions Simulated by a Web Scenario

This topic explains how to modify the user actions simulated by a web scenario.

Sections include:

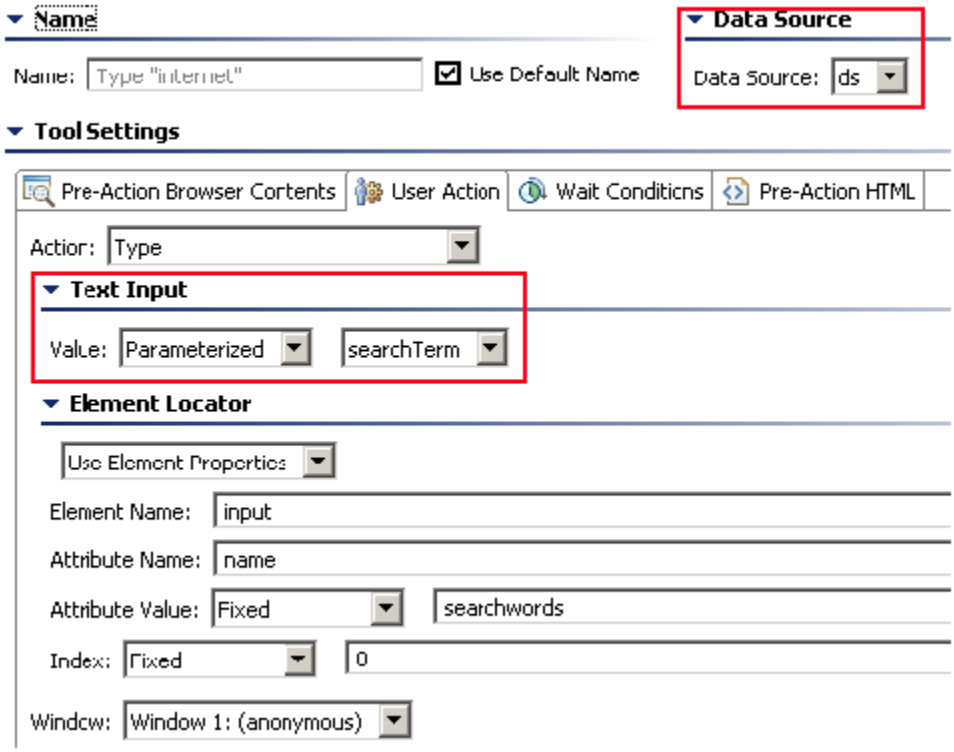
Configuring Actions

To view and modify the action taken by a specific scenario step:

1. Double-click the scenario step whose actions you want to configure.
2. In the configuration panel that opens in the right side of the GUI, open the **User Action** tab.
3. Review the existing actions (initially, the ones captured during recording) and modify the settings as needed to specify the actions you want performed. You can choose from the available pre-set actions, or define a custom one.

Using Data Sources to Parameterize User Actions

You can use data sources to parameterize user actions—for instance, to have a type action iterate through a set of different values stored in an Excel spreadsheet.



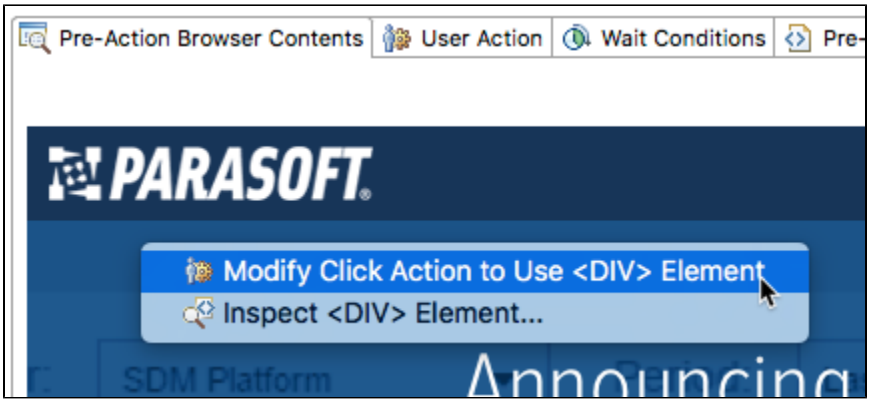
For details on how to add and use data sources to parameterize tests, see Parameterizing Tests with Data Sources, Variables, or Values from Other Tests.

Identifying Elements Associated with User Actions

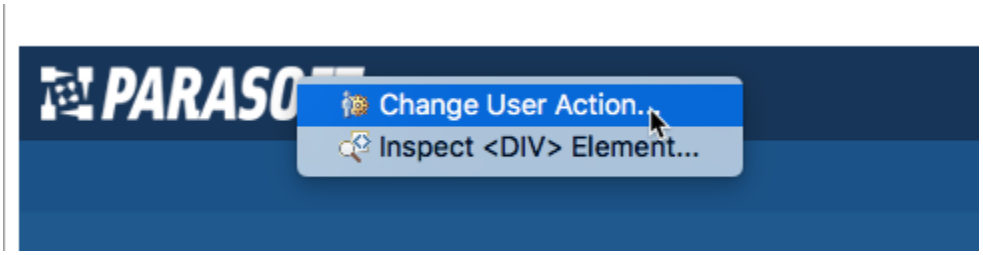
The element that is the source of a user action will be highlighted with a solid blue border in the scenario step's Pre-Action Browser Contents tab.

Changing the Target of a User Action

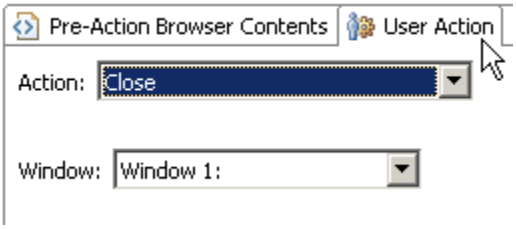
To quickly change the target of a user action, right-click the related element in the **Pre-Action Browser Contents** tab, then choose the appropriate **Modify** command.



If the user action that you want to change is not associated with a specific element (for instance, a "close" or "navigate" action), you can right-click anywhere in the **Pre-Action Browser Contents** tab, then choose **Change User Action**.



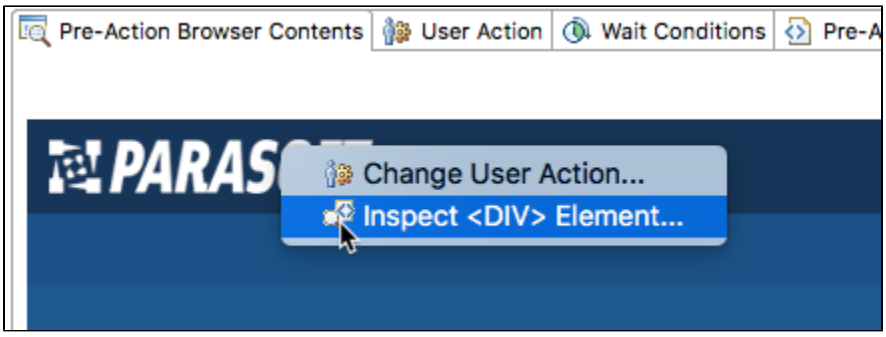
This opens the **User Action** tab, which allows you to modify the target.



Inspecting the HTML for Elements

As you create and modify user actions for page elements, you may want to inspect the HTML to determine if you are adding actions to the appropriate elements.

To see the HTML for a given element by right-click that element, then choose **Inspect <Element>** from the shortcut menu.

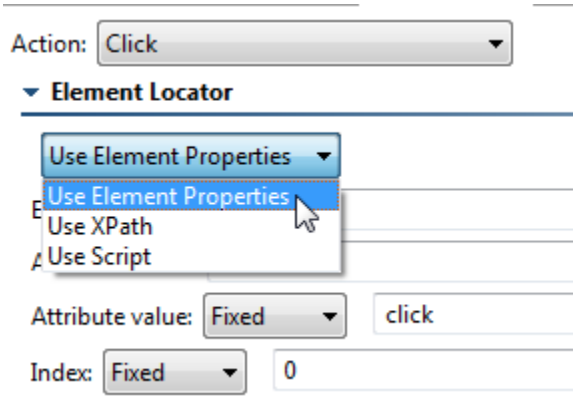


Understanding Preset Actions

You can configure most common user actions by selecting from the list of preset actions, then customizing them as needed to suit your needs. The available preset actions are described below.

Common Configuration Controls

The configuration section for many preset user actions contains an "Element Locator" section. This section allows you to specify the element associated with the action via element properties, XPath, or script.



Use XPath allows you to enter an XPath to be used as an identifier.

Use Script allows you to enter a script that defines the desired user action.

Use Element Properties allows you to specify properties via the following controls:

- **Element:** Specifies the element name (for example, "img", "div", or "a") that the action should apply to. To allow any element, enter "Any" into this field.
- **Attribute Name:** Specifies the attribute name to identify the element (for example, "title", "id", or "name"). You can configure this value using one of the following mechanisms.
- **Attribute Value:** Specifies the expected value for the attribute supplied by the Attribute Name field.
    - If you want to specify a fixed value, select the **Fixed** option, then specify the desired value in the text box.
    - If you want to use values defined in a data source, select the **Parameterized** option, then specify the data source column that contains the values you want to use. Note that this option is only available if the project contains at least one data source.
    - If you want to use the return value of a custom method, select the **Script** option. Click the **Edit** button to create or edit the method(s) and choose the desired method for use from the **Method** drop-down menu in the popup dialog. If there are two or more methods, you can also select a different method for use from the drop-down menu in the form panel.
- **Index:** Specifies the element that matches the previous criteria. Entering "0" means that the first element that matches the "Element," "Attribute Name," and "Attribute Value" criteria will be used. Entering "1" means that the second element that matches will be used, and so on.
    - If you want to specify a fixed value, select the **Fixed** option, then specify the desired value in the text box.
    - If you want to use values defined in a data source, select the **Parameterized** option, then specify the data source column that contains the values you want to use. Note that this option is only available if the project contains at least one data source.

- If you want to use the return value of a custom method, select the **Script** option. Click the **Edit** button to create or edit the method(s) and choose the desired method for use from the **Method** drop-down menu in the popup dialog. If there are two or more methods, you can also select a different method for use from the drop-down menu in the form panel.

Also, note that many actions allow you to configure **Window Name**. This allows you to specify the name of the window you would like the action to occur in. Leaving this field blank indicates that the default window will be used.

Specifying Specialized Element Locators (CSS, ALT attribute, etc.)

With Selenium, you can specify a variety of specialized element locators by setting **Element Locator** to **Use XPath**, then setting one of the available locator prefixes to the desired value. Available prefixes are:

- alt=<alt attribute value>
- class=<class attribute value>
- css=<css-style locator>
- dom=<dom-style locator>
- id=<id attribute value>
- identifier=<id or name attribute value>
- link=<link text>
- name=<name attribute value>
- xpath=<xpath to element>

For example, to use the CSS locator `tag.classA`, use the XPath locator `css=tag.classA`

Errors for Popup Dialogs

If one of the three types of JavaScript popup dialogs (alert, confirm, and prompt) opens in an action other than Accept Script Dialog, Dismiss Script Dialog, or Type into Script Dialog, then an error will be reported—unless you are using the Selenium engine and this is the last test in your scenario.

If a popup dialog opens during a wait action on the Selenium engine, an error will not reported.

Accept Script Dialog

Selenium Only

This action presses the OK button on any of the three types of JavaScript popup dialogs: alert, confirm, and prompt.

Addselection

For a multiselect combo box, this action adds one option to the selection. This can be the first selection or an additional selection. To select multiple items, use this action multiple times.

Answeronnextprompt

Deprecated for 9.8—not recorded with the current engine

This action adds text to a prompt dialog. This action is deprecated; use Accept Script Dialog, Type into Script Dialog, and Dismiss Script Dialog instead.

Configuration notes:

- This action must be added to the scenario immediately before the user action that causes the prompt dialog to open.

Assertalert

Deprecated for 9.8—not recorded with the current engine

This action validates the text in an alert dialog message. This action is deprecated; use Accept Script Dialog, Type into Script Dialog, and Dismiss Script Dialog instead.

Configuration notes:

- This action must be added to the scenario immediately after the user action that causes the alert dialog to open.

Assertconfirmation

Deprecated for 9.8—not recorded with the current engine

This action validates the text in a confirmation dialog message. This action is deprecated; use Accept Script Dialog, Type into Script Dialog, and Dismiss Script Dialog instead.

Configuration notes:

- This action must be added to the scenario immediately after the user action that causes the confirmation dialog to open.

Assertprompt

Deprecated for 9.8—not recorded with the current engine

This action validates the text in a prompt dialog message. This action is deprecated; use Accept Script Dialog, Type into Script Dialog, and Dismiss Script Dialog instead.

Configuration notes:

- This action must be added to the scenario immediately after the user action that causes the prompt dialog to open.

Check

This action checks a check box.

Choosecancelonnextconfirmation

Deprecated for 9.8—not recorded with the current engine

This action presses the Cancel button in a confirm dialog. This action is deprecated; use Accept Script Dialog, Type into Script Dialog, and Dismiss Script Dialog instead.

Configuration notes:

- This action must be added to the scenario immediately before the user action that causes the confirm dialog to open.

Click

This action clicks the specified element.

Configuration notes:

- Use **Key Modifiers** to specify if you want to mimic the user pressing the Alt, Ctrl, or Shift keys during the click.

Close

This action closes the specified window.

Dismiss Script Dialog

Selenium Only

For popup dialogs (alert, confirm, and prompt), this action either presses the Cancel button (for confirm and prompt) or presses the X (for all 3 types of alert dialogs).

Doubleclick

This action double-clicks an element.

Dragdrop

This action drags one element to another location.

Configuration notes:

There are 4 ways to indicate how to move an element. For all of the following indicators, actions start at the element that will be dragged. All numbers are in pixels. The 4 options are

- **Move to target element: <some locator, like //div[@id="example"]>** Drag the source element by clicking the top-left corner of the source element and dragging to the top-left corner of the target element.
- **Start at offset and move by delta: offsetX,offsetY|deltaX,deltaY** Drag the source element starting at offsetX, offsetY from the top-left corner of the source element and moving deltaX, deltaY. Negative deltas move the element up and to the left.
- **Move by delta: deltaX,deltaY** Drag the source element by clicking down in the center of the element and moving deltaX, deltaY. Negative deltas move the element up and to the left.

- **Move to offset within element: <some locator>|offsetX,offsetY** Drag the source element by clicking the top-left corner of the source element and dragging to a point in the target element offset by offsetX,offsetY from the top-left corner of the target element. Offsets 0,0 would be the top-left corner exactly and would function the same as "Move to target element." Negative offsets would be a point above, or to the left of, the target element's top-left corner.

Execute JavaScript

Selenium Only

This action executes the specified JavaScript within the context of the current web page. The specified JavaScript will execute in the same frame as the element defined by the element locator. If no element is defined, the JavaScript will execute in the top-most frame.

To configure this user action:

1. Indicate what JavaScript should be executed. You can copy/type JavaScript into the text box or provide the path to a JavaScript file.
2. Click **Evaluate** to check that the script will run (does not contain syntax errors). Errors from parsing functions in the JavaScript syntax will appear in the **Error message** field.
3. Select the appropriate argument (the function from your script that you want to run) from the **Method** box at the bottom of the panel. This list include all function definitions contained in your script.

Configuration notes:

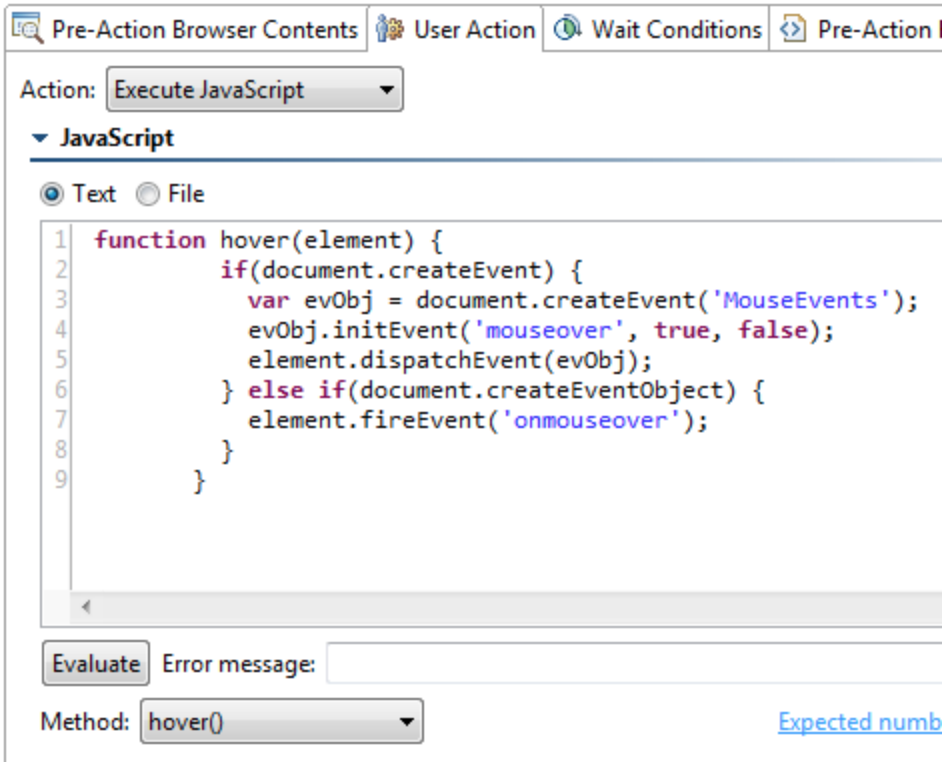
- The locator that is specified in the Element Locator section will be passed as the first parameter to the specified function. For example, assume you have
  ```
  function userFunction(locator) {
     locator.click();
  }
  ```
  The element specified in the element locator would be passed to the function as the locator parameter in this script so that you can act on it as you wish.
- 0 or 1 arguments are expected. When clicked, the dialog should say something like "0 or 1 arguments are expected in this script. When there is one argument, the argument is the element that was specified by the element locator in this user action.

Example - Hovering over elements:

1. Enter the following code into your Execute JavaScript action:

   ```
   function hover(element) {
     if(document.createEvent) {
       var evObj = document.createEvent('MouseEvents');
       evObj.initEvent('mouseover', true, false);
       element.dispatchEvent(evObj);
     } else if(document.createEventObject) {
       element.fireEvent('onmouseover');
     }
   }
   ```

2. Select **hover()** from the **Method** selector.



3. Enter the Element Locator criteria for the element upon which you would like to trigger a hover.

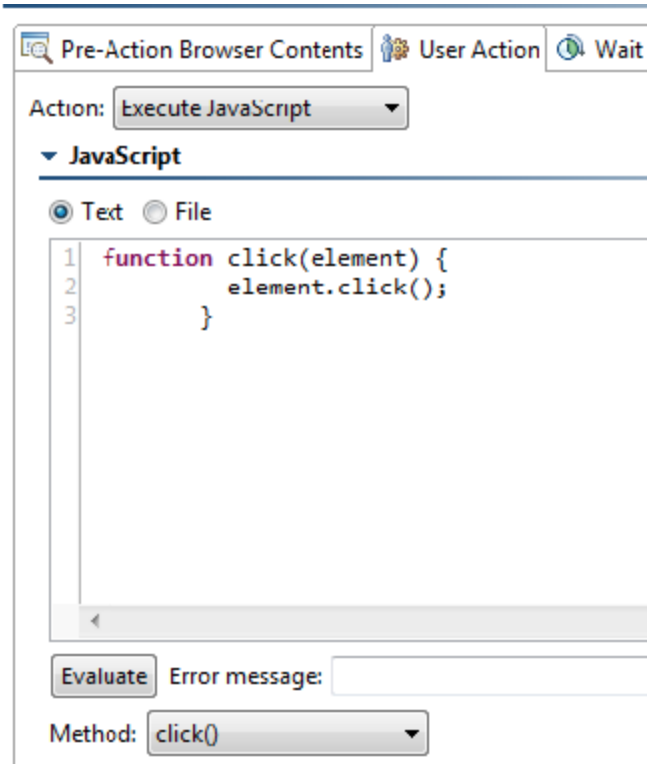
Note that this code executes an "onMouseOver" event on an element. In most cases, this will cause the hover behavior to trigger. In some cases, this is insufficient and more code will be necessary to trigger the hover.

Example - Clicking hidden elements:

1. Enter the following code into your Execute JavaScript action:

```
function click(element) {
  element.click();
}
```

2. Select **click()** from the **Method** selector.



3. Enter the Element Locator criteria for the element upon which you would like to trigger a click.

Note that this code executes the click function on an element. In most cases, this will cause the click behavior to trigger. In some cases, this is insufficient and more code will be necessary to trigger the click.

Fireevent

Targeted for Legacy

This action fires a JavaScript event on a particular page element. This covers JavaScript events that are defined in the code of the page (not visible to the user).

Configuration notes:

- Use the Text Input/Value field to specify the event.
- Specify the event without the "on" prefix. For example, to trigger onkeypressed listeners, you would enter "keypressed."

Go Back

This action presses the browser's Back button. No arguments are needed.

Go Foward

This action presses the browser's Forward button. No arguments are needed.

Keydown / Keypress / Keyup

Targeted for Legacy

These actions fires keydown, keypress, or keyup JavaScript events on an element.

Configuration notes:

- The element locator specifies the element.
- The key locator specifies which key is pressed.

Maximize Window

Selenium Only

This action maximizes the specified browser window.

Configuration notes:

- For Chrome on Mac, the Maximize Window action is interpreted as "size-to-best-fit" instead of "fill-screen," so the browser will only maximize vertically. This is the same behavior as if the user clicks the green 'plus' button to expand the window.

Mousedown / Mouseover / Mouseup

Targeted for Legacy

These actions fire mousedown, mouseover, or mouseup JavaScript events on an element.

Configuration notes:

- The element locator specifies the element.

Navigate

This action navigates to the provided URL as though it was entered in the browser's URL bar.

Configuration notes:

- In the **URL** field, you can enter a **Fixed**, **Parameterized** (if a data source is available), or **Scripted URL**. To enter a scripted URL, select **Scripted**, then click the **EditScript** button to enter a script method to return the URL that should be navigated to in the selected action.

New Browser

Legacy Only

This action opens a new browser populated with the page at the specified start URL.

Other

Enables you to add a custom action.

Refresh

This action has presses the Refresh button.

Removeselection

In a multiselect combo box, this action removes one option from the selection. This is the equivalent of shift-clicking on a selected combo box selection. To remove multiple items, use this action multiple times.

Scroll By

Selenium Only

This action scrolls the web application by the number of pixels specified for the x and y axis. Negative values indicate "scroll left" (for the x-axis) or "scroll up" (for the y-axis).

Scroll To

Selenium Only

This action scrolls the web application to the specified pixel position (x and y axes).

Select

In a single-select combo box, this action selects a single option in the combo box. If an option is already selected, this action will change the selection.

Configuration notes:

- You can specify the option to select by the value (defined in the code) or label (shown in the UI as well as defined in the code).

Submit

This action submits a form, or an element within a form, to the remote server.

Configuration notes:

- Use the element selector to indicate which form to submit (e.g., if there are multiple forms on a page).

Type

This action types the specified text into the specified element.

- Use **Value** to specify what text you want typed. You can enter a **Fixed**, **Parameterized** (if a data source is available), or **Scripted** value. To enter a scripted value, select **Scripted**, then click the **Edit Script** button to enter a script method to return the value that should be typed in the selected action.
- If you want to send special characters to the browser, either type them directly into the **Value** field (valid only when typing produces an output or white space—such as tab, space, semicolon, add, etc.), or use special character mappings as outlined in Special Character Mappings for Type Actions below.

Type (Without Focus) Legacy Only

This action types the specified text into the specified element *without any focus*.

- If you want to send special characters to the browser, either type them directly into the Value field (valid only when typing produces an output or moves the cursor—such as tab, space, semicolon, add, etc.), or use special character mappings as outlined in Special Character Mappings for Type Actions below.

Type into Script Dialog Selenium Only

This action enters text into a prompt dialog.

Configuration notes:

- It must be placed immediately after the action that opens the dialog, and followed by an Accept Script Dialog action (which submits the specified text).
- If you want to send special characters to the browser, either type them directly into the **Value** field (valid only when typing produces an output or moves the cursor—such as tab, space, semicolon, add, etc.), or use special character mappings as outlined in Special Character Mappings for Type Actions below.

Type Password

This action types specified text into the specified element. The text will be masked in the field as well as encrypted upon storage.

- Use **Value** to specify what text you want typed. You can enter a **Fixed**, **Parameterized** (if a data source is available), or **Scripted** value. To enter a scripted value, select **Scripted**, then click the **EditScript** button to enter a script method to return the value that should be typed in the selected action.
- If you want to send special characters to the browser, either type them directly into the **Value** field (valid only when typing produces an output or moves the cursor—such as tab, space, semicolon, add, etc.), or use special character mappings as outlined in Special Character Mappings for Type Actions below.

Type Password (Without Focus)

Legacy Only

This action types the specified text into the specified element without any focus. The text will be masked in the field as well as encrypted upon storage.

- Use **Value** to specify what text you want typed. You can enter a **Fixed**, **Parameterized** (if a data source is available), or **Scripted** value. To enter a scripted value, select **Scripted**, then click the **EditScript** button to enter a script method to return the value that should be typed in the selected action.

Uncheck

This action unchecks/clears a checked check box.

Wait

This action simulates the browser waiting the specified number of milliseconds before continuing to the next step.

- For **Milliseconds**, specify how long you want the browser to wait. You can enter a **Fixed**, **Parameterized** (if a data source is available), or **Scripte** value. To enter a scripted value, select **Scripted**, then click the **EditScript** button to enter a script method to return the value that should be typed in the selected action.

Special Character Mappings for Type Actions

You can use the following special character mappings to specify special characters within Selenium-supported type actions:

```
NULL                \uE000
CANCEL              \uE001
HELP                \uE002
BACK_SPACE          \uE003
TAB                 \uE004
CLEAR               \uE005
RETURN              \uE006
ENTER               \uE007
SHIFT               \uE008
CONTROL             \uE009
ALT                 \uE00A
PAUSE               \uE00B
ESCAPE              \uE00C
SPACE               \uE00D
PAGE_UP             \uE00E
PAGE_DOWN           \uE00F
END                 \uE010
HOME                \uE011
LEFT                \uE012
UP                  \uE013
RIGHT               \uE014
DOWN                \uE015
INSERT              \uE016
DELETE              \uE017
SEMICOLON           \uE018
EQUALS              \uE019
```

Number pad keys

```
NUMPAD0             \uE01A
NUMPAD1             \uE01B
NUMPAD2             \uE01C
NUMPAD3             \uE01D
NUMPAD4             \uE01E
NUMPAD5             \uE01F
NUMPAD6             \uE020
NUMPAD7             \uE021
NUMPAD8             \uE022
NUMPAD9             \uE023
MULTIPLY            \uE024
ADD                 \uE025
SEPARATOR           \uE026
SUBTRACT            \uE027
DECIMAL             \uE028
DIVIDE              \uE029
```

Function keys

```
F1                   \uE031
F2                   \uE032
F3                   \uE033
F4                   \uE034
F5                   \uE035
F6                   \uE036
F7                   \uE037
F8                   \uE038
F9                   \uE039
F10                  \uE03A
F11                  \uE03B
F12                  \uE03C

META                 \uE03D

ZENKAKU_HANKAKU      \uE040
```

These character mappings can be used with a scripted "Text Input" value such as the following script, which will type the word "test" and then send the tab key:

```
def typeTestThenTab() {
        return "test\uE004";
}
```

Note that this does NOT act as a key modifier. For example, you CANNOT perform a "control-click" by sending the control character and then performing a click. Sending the character simulates a user pressing and releasing the corresponding button.

Custom Actions - Migration Note

With SOAtest 9.9.2 and earlier, it was possible to add custom user actions that could be called from Browser Playback tools. These custom actions applied only to executing browser scenarios using the legacy engine. They could be added in two ways: by defining new functions within UserCustomizableOptions.js or in BrowserDriver.js. Parasoft generally recommend defining the custom actions in UserCustomizableOptions.js, but in some cases we suggested using BrowserDriver.js in order to support Chrome.

These functions would take the form

```
_wk_BrowserDriver.prototype.doXXX = function(locator) { ....
```

where XXX is the name of the custom user action.

For example, you could have UserCustomizableOptions.js define an action such as the following:

```
_wk_BrowserDriver.prototype.doContextMenu = function(locator) {
var element = this.findElement(locator);
_wk_HTMLUtil.triggerMouseEvent(element, 'contextmenu', true);
};
```

You would then reference this custom action in SOAtest by choosing an **Other** action and typing in `contextmenu` (legacy engine only).

Starting with SOAtest 9.9.3, adding custom actions like this is no longer supported on Firefox (due to new Firefox extension requirements). Properly migrated custom user actions are still supported for the legacy engine and for playback on Chrome or Internet Explorer.

To migrate your existing custom actions:

- Ensure that they are added in UserCustomizableOptions.js (not BrowserDriver.js).
- If your script references any variables that begin with `_wk_`, change them to reference `com.parasoft.extension`. In other words, change `_wk_XXX` to `com.parasoft.extension.XXX`

For example, the example script above changes to

```
com.parasoft.extension.BrowserDriver.prototype.doContextMenu = function(locator) {
var element = this.findElement(locator);
com.parasoft.extension.HTMLUtil.triggerMouseEvent(element, 'contextmenu', true);
}
```

Note that `wk_BrowserDriver` became `com.parasoft.extension.BrowserDriver` and `_wk_HTMLUtil` became `com.parasoft.extension.HTMLUtil`.