Validating or Storing Values

This topic explains how to extract a value so you can validate it or store it for use in another scenario step or tool.
Sections include:

Understanding Extractions and Validations

In the Browser Contents Viewer tool that is automatically added to each scenario step recorded from the browser, you can click on page elements within a rendered view and automatically set up functional tests on those elements. If a validation is not satisfied in a subsequent playback, the associated scenario step will fail.

In addition, you can "extract" and store data from those elements, then use those extracted values in additional tools (for instance to populate form fields or to validate data). This allows you to easily set up scenarios where dynamic data validation is important. Extracted data can be used in both Web tests and service/API tests.

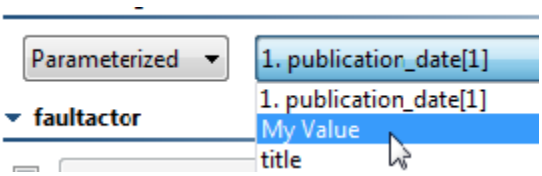
You can also extract path elements from within your Web browser *WHILE* you are recording a path. To do so, right-click an element from within your browser and select **Configure Validations** from the shortcut menu. The **Validation Options** dialog displays. The options within this dialog will be the same as the options in the Browser Contents Viewer tool that appears *AFTER* a path has been recorded and replayed at least once.

Validating or Storing a Value

To validate or store a value represented in the rendered page, complete the following from the Browser Contents Viewer's tool configuration panel (accessible by double-clicking the tool's node) or the Post-Action Browser Contents tab (for extraction only):

1. Right-click the page element whose value you want to validate or store (for example, right-click a link), and choose **Extract Value from <element> Element…** from the shortcut menu.
2. In the wizard that opens, ensure that the desired element is selected in the **Property name** box.
3. If you want to "zoom in" on the value to extract, complete the isolate partial value wizard page. If you want to validate or store the entire string, you can ignore this.
    - Sometimes you may only want to validate or send a portion of a property value to a data source. If this is the case, you can isolate the part of the property value to use by selecting the **Isolate Partial Value using Text Boundaries** check box. You then enter **Left-hand** and **Right-hand** text that serve as boundaries for the value that you enter. The preview pane shows you what value will be used based on the boundary values that you have entered. For example, assume that the property value is "`Click here to log in`":
        - To isolate the value "`Click`", leave the left boundary blank and enter " `here`" (including the space) in the right boundary.
        - To isolate the value "`here`", enter "`Click` " in the left boundary and "`to`" in the right boundary (again including spaces).
        - To isolate the value "`in`", enter "`log` " (including the space) as the left boundary and leave the right boundary blank.
4. To validate a value:
    a. Select **Validate the value**.
    b. Choose from the following expected value options:
        - **equals:** Validates that the property value exactly matches the expected value.
        - **does not equal**: Validates that the property value does not match the specified value.
        - **contains**: Validates that the property value contains the expected value somewhere within it.
        - **starts with**: Validates that the property value starts with the expected value.
        - **ends with**: Validates that the property value ends with the expected value.
        - **is less than**: Validates that the property value is less than the specified value.
        - **is greater than**: Validates that the property value is greater than the specified value.
        - **is not present**: Validates that the specified property does NOT appear on the page, and will report an error if it does. This is useful for cases when a web application is showing an error message that it should not be.
        - **matches color**: Validates that the color values correspond to names of colors specified in the validation colors mapping file. See Validating Color Elements for details.
    c. Choose **Fixed, Parameterized,** or **Scripted,** then specify a value.
        - If the Parameterized option is chosen, then you can specify a column name from that data source. When the scenario is played back, the expected value will be taken from the appropriate row and column in the data source. Column names will only be shown for one data source, so if you have multiple data sources in your project, you will need to go to the chained Browser Validation tool and modify the data source being used at the top of that panel. If other extracted column names are available because they were set up by extracting from a different HTML page, they will also be in the list of available column names, even if the project does not define a data source.
5. To send the value of the selected property to a data source (so that the value can be used later in another scenario or in another tool):
    a. Select **Extract the value to a data bank**.
    b. Specify which data source column you want to contain the extracted value.
    Custom column name: Tells SOAtest the name of the data source column in which to store the value. Values are stored in an internal data source unless you specify otherwise (e.g., if you select **Writable data source column** or **Variable).** This is the name you will use to reference the value in other places. For example, if it is stored in a data source column named **My Value,** you would choose **My Value**

as the parameterized value. You could also reference it as `${My Value}` in literal or multiple response views.



Writable data source column: Tells SOAtest to store the value in a writable data source column (see Configuring a Writable Data Source for details). This allows you to store an array of values. Other tools can then iterate over the stored values.

Write to all columns that match: Tells SOAtest to store the value in all columns whose name contains the given string. When extracting multiple values from a message, each value will be written across all the columns that match. In contrast, if you pick a single writable data source column (the above option), then the values will be written down the column across multiple rows.

Variable: Tells SOAtestto save the value in the specified variable so it can be reused across the current test suite. The variable must already be added to the current test suite as described in Defining Variables. Any values set in this manner will override any local variable values specified in the test suite properties panel.

When the scenario step is executed, the property value will be extracted from the page and placed into a temporary data source within a column with the specified name. When later parts of the scenario reference the column name, the value stored in the temporary data source will be used for those scenario steps. You can both validate and send a property value to a data source at the same time if desired.

6. Click **Finish.**

The value will be validated or stored when the scenario is executed.

What if I don't see the value I want to validate or extract?

If the Browser Contents Viewer tool does not display the value you want to extract or validate—for example, because the related scenario step failed or because the item is not visible in the rendered page (e.g., it is a title), you can manually add a Browser Validation tool or Browser Data Bank tool as described in Adding Test Outputs.

Need to handle dynamic attribute values?

One way to handle dynamic attribute values in validations is to use partial string XPaths. To use XPaths, ensure that the Browser Validation tool's Element Locator is set to **Use XPath**, then specify an appropriate XPath. For details on using XPaths, see XPath Reference**.** For details on using partial string XPaths, see Handing Dynamic Attribute Values: Partial String Matching Using XPath**.**

Want to access the HTML content as a string?

You can retrieve the HTML for a browser window or frame using input.getHTML(). See the Javadoc for com.parasoft.api.BrowserContentsInput. The Javadocs can be accessed by choosing **Parasoft> Help**, then looking for the book titled "Parasoft SOAtest Extensibility API".

For example, here is a JavaScript that searches for an RFC title.

```
// input: com.parasoft.api.BrowserContentsInput.
// context: com.parasoft.api.ExtensionToolContext.
function validateRfcText(input, context) {
  var html = input.getHTML();
  var rfc = context.getValue("ds", "rfc");
  // Extract the numeric part of the RFC.
  // From "RFC5280" extract "5280".
  // From search("\\d") returns the index of the first digit in rfc.
  // See a reference on JavaScript regular expressions.
  // Alternatively hard-code rfc.substring(3),
  var rfcNumber = rfc.substring(rfc.search("\\d"));
  var title = "Request for Comments: " + rfcNumber;
  if (html.indexOf(title) < 0) {
    context.report("HTML does not contain title: " + title);
  }
}
```

If you configured a validation...

A Browser Validation tool will be chained to the test. This tool will performing the validation. If you later want to modify the validation, you can do so by modifying this tool's settings.

The element that is the source of a validation will be highlighted with a solid red border in the Browser Contents viewer, and in the Post-Action Browser Contents tab of the Browser Validation tool.

If you configured an extraction...

A Browser Data Bank tool will be chained to the Browser Playback tool. This tool will store the extracted value. The extracted value can be used wherever parameterized values are allowed, such as the value to type into an input in a subsequent scenario step or another tool. If you later want to modify the stored value, you can do so by modifying this tool's settings.

The element that is the source of an extraction will be highlighted with a solid gray border in the Browser Contents viewer, and in the Post-Action Browser Contents tab of the Browser Data Bank tool.

If you configured both...

A Browser Validation tool and a Browser Data Bank tool will be chained to the test as described above. In addition, a dotted purple border will be used to highlight the source element.

Specialized Extractions/Validations

Validating or Extracting Text

To validate text that appears on a page (or to extract text to a browser data bank), complete the following:

1. Select the text you want to validate or extract.
2. Right-click the selection, then choose one of the following:
    - To configure a validation for that text, choose **Validate Selected Text** from the shortcut menu.
    - To configure an extraction for that text, **Extract Selected Text into Data Bank** from the shortcut menu.
3. Ensure that the desired validation/extraction settings appear in the dialog that opens.
4. Click **Finish.**

Validating Color Elements

To create a test that validates the color on a page, complete the following:

1. Right-click the page element for which you would like to create a test (for example, right-click a link), and choose **Extract Value from <element> Element…** from the shortcut menu.
2. In the wizard that opens, ensure that **style_color** is selected in the **Property name** box, then click **Next** two times.
3. In the Validate or Store Value wizard page, select **matches color** from the **Expected Value** drop-down menu and enter a color in the text field (e.g. "red"). The **matches color** option validates color values corresponding to names of colors specified in the validation colors mapping file. These mappings are either in hex notation, or RGB notation `-- rgb(0, 255, 0)`. For more information, see Validation Colors Mapping File.
4. Click **Finish.**

The color validation will be performed when the test is executed.

Validation Colors Mapping File

There is a file in the product installation called the Validation Colors Mapping file. This file defines how SOAtest validates colors by name. It is located in `<SOAtest_Installation_Directory>/plugins/com.parasoft.xtest.libs.web_<soatest_version>/root/validation/validationColors.txt`.

Each line of the file defines a color by name, along with ranges for each component of the RGB color model. The specified ranges tell SOAtest that if a color is validated and falls within the ranges for each of the components of the RGB color model, then the color being validated matches the color that is defined within those ranges. For example, a line in this file may look like the following:

```
red, b0-ff, 00-30, 00-30
```

This line defines the valid ranges for the color "red". The ranges are specified using hex notation. In the above example the valid R range for red is between the hex values b0 and ff. The valid G and B ranges for red are between the hex values 00 and 30. In other words, if an element has a hex value of #c80000, then it will be considered to be red, since the R value, which is c8, falls between b0 and ff, and each of the G and B values, which are 00, fall between 00 and 30. However, if a validation is set up on an element that is expected to be red, but the element's color has a hex value of #909090, then SOAtest will display a message that the element has the incorrect color.

The mapping file has a few standard colors already defined. However, if you would like to specify additional colors, you can simply modify the file. There must be only one color defined per line. Also, if you want to change the valid RGB ranges for a defined color, you can also modify the mapping file. Ranges can be specified with a hyphen (b0-ff, as already described), or they can be a single value (ff). If they are a single value, the range of valid values only includes one value. As mentioned before, the ranges must also be in hex notation.

SOAtest must be restarted in order for changes to this file to take effect.

Validating Style Properties

To validate a style property, open the Browser Validation tool's configuration panel, then set the validation's **Element Property** to the value `"style_" + <the JavaScript name of the property>`. For example, to validate the `text-decoration` style property, you would specify `style_textDecoration` (`textDecoration` is the JavaScript way to specify the style property `text-decoration`) in the **Element Property** field, and specify the desired value of the property using the **Expected Value** controls. In the `text-decoration` case, the expected value might be equal to `line-through` or `underline`.

Validation Styles List File

If you want a certain style property to display as an available property in the validation wizard, you can add that style to the Validation Styles List.

The Validation Styles List file, located in `<SOAtest_Installation_Directory>/plugins/com.parasoft.xtest.libs.web_<soatest_version>/root/validation/validationStylesList.txt`, specifies runtime style properties that can be validated.

The format of this file is to have one property per line. By default, the color property is specified in this file; however, you can add any valid style properties that you would like to validate. SOAtest must be restarted in order for changes to this file to take effect. Once it is restarted, you will see the properties specified in this file in the validation dialog when right-clicking on an element. The properties will have "style_" appended to each of the properties to tell you (and SOAtest) that these refer to the runtime value individual style properties.

A validation set-up using one of these properties will validate the runtime value of that style property. This is the runtime value of the property after all inline styles and styles defined in CSS files have been applied. Because of this, the value may differ from what is defined inline in the element. For example, these runtime style validations allow you to validate the actual color that is seen by a user after all styles have been processed by the browser.

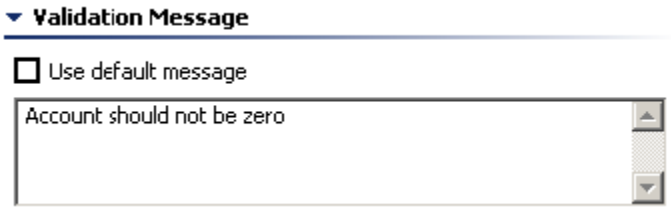
Customizing the Message Shown when the Validation Fails

For each validation, SOAtest automatically configures a message to be shown in the Quality Tasks view (and in reports) if that validation fails. This message typically looks something like "Validation failed for property [property_name]: Actual value found on the page [actual value]" must be equal to expected value [expected value]."

You can easily configure custom messages that are more meaningful in the context of your application. For instance, assume that you are working with an online banking application, and you are adding a validation to check that the account balance does not reach zero. In this case, you might want to spec-ify the custom validation message "Account should not be zero" instead of using the default built-in message, which might be "//span[text() = "account" was not found".

To configure a custom validation:

1. Open the configuration panel for the Browser Validation tool you want to configure.
2. Clear the **Use default message** option.
3. Specify your desired message in the text field. You can use any text you like, as well as two keywords:
    - ${ActualValue} - Will be replaced with the actual value found on the HTML page.
    - ${ExpectedValue} - Will be replaced with the expected value (can be fixed, parametrized, or scripted).



Viewing Stored Variables Used During Test Execution

You can configure the Console view (**Window> Show View> Console**) to display the stored data bank variables used during test execution. For details, see Console view**.**

Configuring Custom Validations with XPaths

XPaths can be used to specify complex validations. To use XPaths, ensure that the Browser Validation tool's Element Locator is set to **Use XPath**, then specify the appropriate XPath. See XPath Reference for details on using XPaths.

Configuring Custom Validations with Scripting

If you want to perform complex validations that cannot be properly representing using the GUI controls, you can express them using scripting. To do this, you change the Browser Validation tool's Element Locator to **Use Script**, then specify an appropriate script.

For example, suppose that you want to validate all of the rows in a table. That table could be of variable length. You can attach an Extension tool to a browser functional test and pull values from the document provided by input.getDocument(). Here's a sample JavaScript script that accomplishes that.

```
var Application = Packages.com.parasoft.api.Application;
var WebBrowserTableUtil = Packages.webking.api.browser2.WebBrowserTableUtil;
var WebKingUtil = Packages.webking.api.WebKingUtil;
// Verify that all values in a table column are equal to a previously
// extracted value.  For example, we searched for all places in which
// widget 11 is sold, and we want to make sure that all results are // for widget 11.
// input: com.parasoft.api.BrowserContentsInput.
// context: com.parasoft.api.ExtensionToolContext.
function validateTable(input, context) {
  // For the column we want to validate.   var widgetColumnIndex = 0;
  // We extracted through a Browser Data Bank in a previous test
  // the expected value to data bank column "widgetName".
  // The value was extracted, not from a data source, so use "" or
  // null (None in Python) as the name of the data source.
  var expectedWidgetName = context.getValue("", "widgetName");

  var document = input.getDocument();
  // Table should have some unique identifying attribute (e.g., id).
 var table = WebBrowserTableUtil.getTable("id", "mytable", document);
  // If the first row of the table contained column headers, we could
  // use getCellValuesForColumn(String, Element).  For example if the
  // widget name column was named "Widget Name", then we could use
  // getCellValuesForColumn("Widget Name", table).  In either case,
  // getCellValuesForColumn returns an array of String objects.  See
  // the JavaDocs for more information.
  var values = WebBrowserTableUtil.getCellValuesForColumn(widgetColumnIndex, table);

  if (values.length == 0) {
    context.report("No rows found!");
}
  for (var i = 0; i < values.length; ++i) {
    if (values[i] != expectedWidgetName) {
         var errorMessage = "Widget name (column " + widgetColumnIndex + "): "
      + "Expected value '" + expectedValue
          + "', but found '" + values[i] + "'.";
     context.report(errorMessage);
   }
}
```