Handing Dynamic Attribute Values - Partial String Matching Using XPath

Partial string matching using XPath is helpful for handling dynamic attribute values.

For most element locators, you match using equality: find an element whose "id" attribute is exactly "foo" or whose text is exactly (possibly ignoring whitespace) "hello world".

In some cases, you might want to match only part of an attribute value or text content.

Examples

For example, some web frameworks commonly generate "id" or "name" attribute values in the form "constant_dynamic", such as "transid_2010". The attribute value always begins with "transid_", but "2010" is a dynamic value that might be "1984" the next time you load the page.

Assume that the constant "transid_" prefix is sufficient for allowing you to uniquely identify the element on the page. In that case, you can create a simple XPath locator the matches "transid_", but ignores the dynamic part of the attribute value.

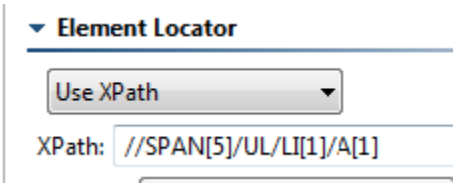
Consider this input element with the "name" attribute in the form "constant_dynamic" as described: `/descendant::input[starts-with(@name, 'transid_')]`

The XPath looks anywhere in the document and finds the first <input> element whose "name" attribute begins with the string "transid_". It ignores any characters after "transid_". The "starts-with" and all other XPath 1.0 functions are documented in the W3C Recommendation at http://www.w3.org/TR/xpath/ (browsers—and thus SOAtest browser tests—use version 1.0 of XPath).

The string functions are of particular interest for matching part of an attribute value or part of some text content; they are described at http://www.w3.org/TR/xpath/#section-String-Functions.

For instance, the "contains" function is similar to—but more general than—the "starts-with" function: it matches a substring anywhere in a string. The following XPath would find the desired input element— assuming there are no inputs with a name attribute value such as "something.transid_xyz": /descendant::input[contains(@name, 'transid_')]

For more XPath examples, you can examine the XPaths that SOAtest generates for Element Properties locators. Once you have saved an Element Properties locator, change the locator to **Use XPath**.



SOAtest will then displays an XPath that is functionally equivalent to the saved Element Properties locator.

Scripting or XPaths?

If you are not familiar with XPath, your first inclination when there is not an obvious Element Properties locator may be to use scripting to access the DOM: either a scripted "Attribute value" for an Element Properties locator, a "Use Script" locator.

Whenever you consider scripting, first consider whether you can create an XPath locator. Various search strategies lend themselves well to XPath, and XPath expressions are typically easier to create and maintain than scripts.

Preventing the Use of a Specific Dynamic Attribute

Note that if you simply want to avoid using the dynamic "name" attribute (or any other attribute) in locators, you can configure SOAtest not to use the "name" attribute when generating locators during recording. For details on how to achieve this, see Customizing Recording Options.

For example, some web frameworks generate "id" values that are essentially random values from SOAtest's point of view (for example, "gwt-123"). In these cases, you would probably want to ignore the "id" for locator creation.