Populating and Parameterizing Elements with Data Source Values

This section applies to the Populate option, which is available for Form Input and Form JSON trees.

Why Populate and Parameterize Elements with Data Source Values?

Let's say you have a complex request message that looks something like this.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SOAP-ENV:Body>
  <addNewItems xmlns="http://www.parasoft.com/wsdl/store-01/">
   <book>
    <id xmlns="">0</id>
    <title xmlns=""></title>
    <price xmlns="">0.0</price>
    <authors xmlns="">
     <i></i>
     <i></i>
    </authors>
    <publisher xmlns="">
     <id myAtt="attVal">1</id>
    </publisher>
   </book>
  </addNewItems>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

You will want to parameterize most of the above elements—and even some arrays that may vary in the number of elements they contain (such as the list of authors in the above example). This is data-driven testing.

However, manually creating an Excel spreadsheet with data and parameterizing each individual element can be time-consuming and tedious.

If you already have an existing data source, you can use the Populate feature (described below) to automatically map data source values to message elements.

If you do not already have a data source with these values, you can use the Populate feature (described below) to automatically generate simple values for a set of form fields. You can also automatically generate and then complete a data source template as described in Generating a Data Source Template for Populating Message Elements.

Data Source Mapping and Naming Conventions

When parameterizing element or attribute values, there are three possibilities:

- Specifying a value
- Specifying that an element should be Nil or Null
- Specifying that an element should be excluded entirely.

Specifying Values

Matching data source columns to request elements is accomplished via certain naming conventions applied to the data source column names. In the example XML above, notice that there are 2 elements named "id." To distinguish them, we can use "book/id" as one column name and "book/publisher/id" as the other. This naming convention mimics a file directory structure or an XPath. Attributes are similarly identified, with the additional specification of an "@" symbol. In the example above, "book/publisher/id@myAtt" refers to the attribute "myAtt" of the publisher id.

Next, consider the case where several elements may have the same name, as demonstrated in the authors element above. The "book/authors/i" identifier applies to both elements, so we need a way to distinguish them. In this case, we can append array numbers within parentheses "()" to repeated elements of the same name. Hence, "book/authors/i(1)" identifies the first element, and "book/authors/i(2)" identifies the second.

Specifying Nill and Exclude

In some cases, you will want to specify that an element appear as Nil or that the element should not appear in the request message. By default, appending "XL" for exclude and "NIL" for nil values will accomplish this goal. For example, a column named "book/authors/iXL(2)" will allow you to indicate that the second child of the authors tag should not be sent, such as for the case where there is only one author.

How to Populate and Parameterize Elements with Data Source Values

The Populate option applies to Form Input and Form JSON trees.

In cases of large, complex message requests, the process of configuring each element and attribute item in the request message, and then parameterizing it with the correct data source column can be time consuming. Therefore, it is possible to expedite this process with the automatic populate and parameterize feature.

Ultimately, when this feature is used with the associated data source naming conventions, it can provide huge productivity gains, especially when dealing with messages containing hundreds of nested, complex elements. It also allows you to focus on designing use cases in a data-driven manner (using data sources) instead of focusing on the error-prone method of individually configuring each message parameter.

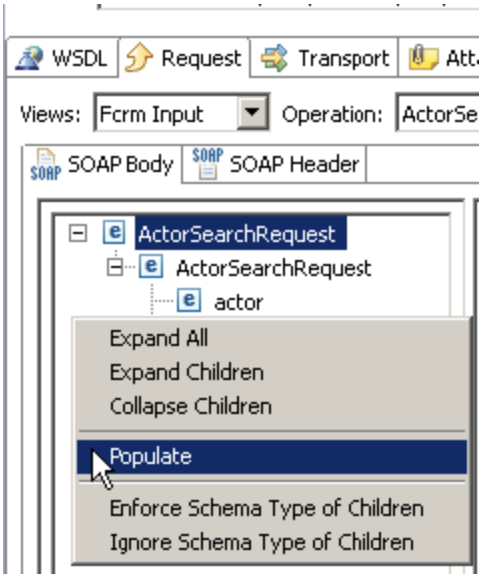
Note for SOAtest 5.x Users

In SOAtest 5.x, the populate feature was available for Form XML and Literal XML views. It populated Form Input, and then overrode the values in Form XML or Literal XML with values from Form Input.

In SOAtest 6.x and later, you should populate in Form Input, then switch views to have the same effect.

To automatically populate and parameterize elements with data source values, complete the following:

1. Right-click in a blank area of the tree (not on a specific element) and select **Populate** from the shortcut menu.



2. Enable **Map parameter values to data source columns** to tell SOAtest to automatically set each form input parameter to **Parameterized.**
3. Select the data source column with a name that matches the parameter name. For example, if the data source has a column name "`title`" and one of the form input elements has the same name "`title`", then the "`title`" element will be mapped to the data source column "`title`", and so on.
4. Customize the remaining options as needed. See Populate Wizard Options for details.
5. Click **OK**. A **Populate Results** dialog displays Summary and Detail information.

Populate Wizard Options

Option	Description
Map parameter values to data source columns	(Only enabled when a data source is present.) Indicates whether to automatically set each form input parameter to **Parameterized** and selects the data source column with a name that matches the parameter name. For example, if the data source has a column name "`title`" and one of the form input elements has the same name "`title`", then the "`title`" element will be mapped to the data source column "`title`", and so on.

Element Exclusion *Not applicable to repository data sources*	Indicates whether to also map the property **Use data source**: **Exclude [element name] with empty string** with a data source column. For more details about the exclude with empty string option, see Using Data Source Values to Configure if Optional Elements Are Sent**.** The following options are available from the **Element Exclusion** drop-down menu: • **Always Include**: Instructs SOAtest to always add new elements that are optional (schema type attribute minOccurs='0') as part of the populate process. The number of elements added is determined by the **Number of sequence (array) items** field. By default that is set to the value of 2. *This option is not applicable to Form JSON.* • **Leave Unchanged**: Leaves each sequence element at the current state. No new elements are added and no exclusion properties are modified. • **Use Data Source**: Instructs the populate process to map the **Use data source**: **Exclude [element name] with empty string** property of each Form Input element to the data source with the same name, but postfixed with the value XL. If the value in the specified data source column is an empty string, the optional element will not be included in the message. If it is an actual value, that value will be sent as part of the message. The postfix XL is specified in the **Exclude column name postfix** field; XL is the default value. For example, if the request XML message includes an element named "**title**" and that element type in the schema is defined with the attribute `minOccurs='0'`, the **Use data source: Exclude title with empty string** option becomes available in the Form Input view when right-clicking on the parent node of title. The populate feature would map the exclude property to the data source column named "`titleXL`", assuming that "`XL`" is the postfix specified.
Nillable Elements / Null Elements *Not applicable to repository data sources*	Similar to **Element Exclusion** except that **Nillable Elements** affects the **Use data source: Set nil with empty string** property. This Form Input property is available on elements that have their schema type set with `nillable="true"`. When the data source has an empty string, the nil attribute will be sent as part of the message. If a value is specified, the request element will include the specified value; the nil attribute will not be sent. If the data source has an empty string (e.g., ""), then the request element will have no value and will include the xsi:nil="true" attribute. For more information, see Using Data Source Values to Configure if Nil or Null Attributes Are Used**.** The **Nillable column name postfix** field of the dialog specifies the postfix. *The "Always expand" setting is not applicable to Form JSON.*
Attribute Exclusion *Not applicable to repository data sources or Form JSON*	Indicates whether optional attributes are automatically added by the populate process.

Data Source Mapping and Naming Conventions

For both sequence (Array) and nested types: The value mapping and exclude/nillable mappings are based on name-matching conventions between the element name and the column name. However, there are cases where the same element name is reused within the XML message, so mapping collisions need to be avoided if one-to-one mapping between each element and data source column is to be maintained.

For nested complex types: XPath-like data source column names can be used to disambiguate. For example, instead of using the data column name "`title`", you may use "`book/title`" as the column name and it would therefore be mapped to any "`title`" elements falling under "`book`". If that can lead to ambiguity, you may also use a column name such as "`addNewItem/book/title`" to further identify which element it is supposed to be associated with.

For sequence types (arrays where items with the same name are repeated): Item index numbers can be used to disambiguate. For example, in the Parasoft store service the `book` type has an `authors` element, which in turn can have many "`i`" elements indicating a list of authors. Only using the data source column name "`i`" would result in that data source column being mapped to all occurrences of element "`i`".

Using data source column name "`i[2]`" which results in that column name being mapped to all occurrences of "`i`" as the second item in the sequence. (The index numbers start from 1, not 0 as per the XPath specification). If the column name "`authors/i[2]`" is used, then it will be mapped only to the second item "`i`" element under the "`authors`".

Note: if there happens to be multiple "`authors`" elements in the XML message, then all of them would be mapped accordingly, unless the column names are disambiguated with enough XPath parent nodes to make the mapping one-to-one. Parentheses can be used as the numeric index syntax, so "`authors/i(2)`" will map to the same elements as "`authors/i[2]`". The () syntax is inconsistent with the XPath specification, but it is helpful when database data sources are used where `[]` is not a valid SQL character.

Exclude and Nillable mapping: Follows the same XPath and indexing conventions as values. For example, to exclude/include the "`i`" element based on a data source column, you may use the column name "`authors/iXL[2]`" to indicate specifically which elements it is intended for.