Adding a Custom Tool 1

This topic explains how to extend SOAtest and Virtualize with custom tools.

Sections include:

About Custom Tools

SOAtest and Virtualize include a framework for adding your custom tools. Custom tools can be configured to perform custom executions on input. For example, a custom tool might modify a variable, or remove browser cookies.

Interfaces to Implement for Custom Tools

After setting up your environment for a custom extension, implement the following interfaces (described in the Extensibility API documentation):

- com.parasoft.api.tool.ICustomTool

ICustomTool Implementation

This is a required class that will be used to implement the tool execution logic. There are three methods to implement in this interface:

- execute()
- acceptsInput(IToolInput, ICustomToolConfiguration)
- isValidConfig(ICustomToolConfiguration)

You will implement your tool's execution logic using the execute() method, which provides you with an IToolInput and IToolContext. The IToolInput contains the input message being sent to your tool, such as XML or traffic from a request or response. The context provides access to the UI configuration and output manager. Returning true indicates a successful tool execution, while false indicates a tool failure.

The output manager can be used to send an IToolInput to tools that are chained to instances of your custom tool. You can either construct your own implementation of an IToolInput, or use a default implementation provided by Parasoft.

For example, assume that you defined two outputs in your parasoft-extension.xml:

```
<output key="output_1" name="traffic header"/>
<output key="output_2" name="traffic body"/>
```

In SOAtest or Virtualize, you could select your custom tool, click the **Add test or output** (SOAtest) or **Add responder or output** (Virtualize) button, and add tools to the selected output. The following is an example of constructing inputs and passing them to the chained output tools:

```
public boolean execute(IToolInput input, IToolContext context) throws CustomToolException
{
    String charset ="UTF-8";
    String mimeType ="text/plain";
    String header ="";
    String message ="";
    . . .
    DefaultTextInput headerOutput = new DefaultTextInput(header, charset, mimeType);
    context.getOutputManager().runOutput("output_1", headerOutput, context);
    . . .
   DefaultTextInput msgOutput = new DefaultTextInput(message, charaset, mimeType);
   context.getOutputManager().runOutput("output_2", msgOutput, context);
   . . .
   return true;
}
```

If errors are encountered during tool execution, one or more errors should be reported using IToolContext.report(String). If a fatal error is encountered that prevents the tool execution from completing, a CustomToolException should be thrown; this will abort the execution of the tool and cause it to fail, in addition to aborting scenario execution (if the scenario is set to abort on fatal errors). Errors reported in either manner will get reported to the Quality Tasks view.

Additional methods to implement:

- **acceptsInput(IToolInput, ICustomToolConfiguration)**: Returns a boolean. You will use this method to determine whether or not your tool accepts the given input. If true is returned, then execute() will be called; otherwise execute() will not be called.
- **isValidConfig(ICustomToolConfiguration)**: Returns a boolean. You can use this method to determine whether or not the UI configuration is properly set up before run time. If this method returns false, then the tool will be unable to be run.

When working with IToolInputs, you will need to check the specific type of IToolInput that gets passed to the tool. In the majority of cases, SOAtest or Virtualize will pass an instance of a subinterface of IToolInput. However, there may be some cases where the input cannot be converted to a more specific subinterface of IToolInput. In those cases, a simple IToolInput will get passed to the tool. IToolInput has no method, but there is a way to get access to the internal object being used. All IToolInputs will be instances of an internal interface called com.parasoft.tool.IToolInputWrapper. You can cast to that interface and then call the getToolUsable() method to get access to the internal object being used.

A similar case might occur when passing inputs to the outputs of a custom tool. You may need to pass an object from the internal API, rather than an instance of a more specific subinterface of IToolInput. In that case, you would return an instance of the internal interface com.parasoft.tool.IToolInputWrapper, which returns a com.parasoft.tool.ToolUsable.

Defining parasoft-extension.xml for a Custom Tool

After you have implemented the necessary classes, define parasoft-extension.xml as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<extension xmlns="urn:ocm/parasoft/extensibility-framework/v1/extension"
                   type="tool"
                   name='the name of your tool, appears in menus'
                   description='A more detailed description'>
<class>com.myCompany.MyTool</class> <!-- implements ICustomTool -->
<version id='your version ID' updaterClass="com.myCompany.myUpdater"/>
<tool xmlns="http://schemas.parasoft.com/extensibility-framework/v1/tool"
           icon="myIcon.gif"
           useInputTab="true"
           successIndicator="true"
           category="myCategory"
           supportXmlConversion="true"
           showInToolList="common"
           showInOutputList="common">
           <outputs>
                     <output key="key" name="output name"/>
                     ...
           </outputs>
           </tool>
<form xmlns="urn:com/parasoft/extensibility-framework/gui">
                   <section label="field group 1">
                             <field id="key 1" label="field 1"/>
                             <field id="key 2" label="field 2"/>
                             <field id="element" label="Select an element" type="xpath"/>
                   </section>
                   <section label="field group 2">
                             <field id="key 3" label="field 1"/>
                             <field id="usr" label="Username"/>
                             <field id="pwd" label="Password" type="password"/>
                             . . .
                   </section>
                   . . .
</form>
</extension>
```

<tool> Element

This element is unique to this extension type and must be valid and correct for your custom tool to be imported.

Attributes:

- icon - Optional attribute that allows you to provide the name of a custom icon to be displayed next to the tool name in menus and next to the assets using the tool in the Test Case Explorer (SOAtest) or Virtual Asset Explorer (Virtualize). If not provided, a default icon will be used.
- useInputTab - Optional attribute that allows you to specify whether or not your custom tool will have an input tab. The default is false.
- successIndicator - Optional attribute that allows you to specify whether or not this tool acts as a success indicator for its parent. If true, success of the tool will determine the success of its parent. If false, whether the tool passes or fails has no bearing on whether its parent tool passes or fails.
- category - Optional attribute that allows you to either place the tool in a category predefined by Parasoft, or define a new category for your custom tool. Existing categories are:
    - SOA/Messaging
    - Virtualization
    - Web
    - Validation
    - Database
    - Scripting
    - Data Exchange
    - Java
    - BPEL
    - WSDL
    - Logging/Monitoring
    - Transformation
    - XML Security
- supportXmlConversion - Optional attribute that specifies whether non-XML input is converted to XML before the defined execute() method is called. The default is false.
- showInToolList - Optional attribute that specifies the availability of your tool in the add tool wizard dialogues. The possible values are:
    - common - Appears in the "Common Tools" folder when adding a new tool.
    - all - Appears in the "All Tools" folder when adding a new tool.
    - no - Will not appear as an available tool.
- showInOutputList - Optional attribute that specifies the availability of your tool in the add output wizard dialogues. The possible values are:
    - common - Appears in the "Common Tools" folder when adding an output.
    - all - Appears in the "All Tools" folder when adding an output.
    - no - Will not appear as an output tool.
- <output> - Defines an output type for your custom tool.
    - key - A string identifier for your output type - this must be unique among the keys for all outputs defined for your custom tool.
    - name - Used to display the name of your output type.

<form> Element

This element is unique to this extension type and must be valid and correct for your custom tool to be imported. It defines input fields for your custom tool.

Attributes:

- <section> - Defines a category of fields.
    - id - The internal name identifying the category of fields.
    - label - The category name/label that is displayed in the UI.
- <field> - Defines a specific input field that is shown in the configuration UI.
    - label - The field name/label that is displayed in the UI.
    - type - The type of field. This can be set to `string` (a normal field), `xpath` (provides XPath Chooser functionality) or `password` (hides the value as you type and encrypts the value when saved).

Tips

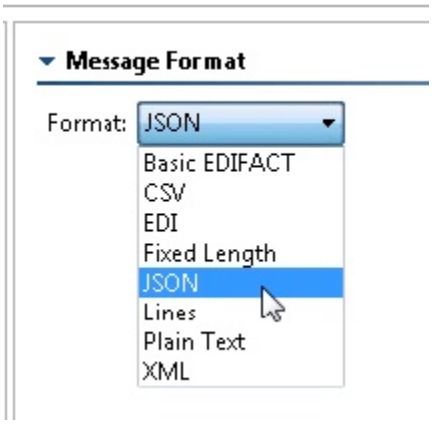
- The values provided to the extension GUI are saved as a name-value String map. As a result, rearranging the fields in the form element in parasoft-extension.xml will not affect how the user values are saved; however, changing the ids will affect this. The ids are used to save/load the values so they need to be unique. If you change them, then previously-saved configurations will not load the previous values and will become empty. However, you can use a version updater to migrate old settings saved with old ids to a new set of ids.
- Fields can specify a `type` attribute. This can be set to:
    - `string`: A normal field.
    - `xpath`: Provides XPath Chooser functionality.
    - `password`: Hides the value as you type and encrypts the value when saved.
- Only GUI fields with string values are supported in the custom form GUI. If your extension requires integers or other types, then you may convert the string content to the desired type in the extension implementation.
- If you want a GUI field to serve as a password field (with inputs masked and the specified password saved securely), give that field element a `type` attribute which is set to `password`. For example, the following sets the `pwd` field to password mode:

```
<form xmlns="urn:com/parasoft/extensibility-framework/gui">
    <section label="Main Settings">
        <field id="usr" label="Username"/>
        <field id="pwd" label="Password" type="password"/>
    </section>
</form>
```

- If you want a GUI field to be implemented as an XPath chooser, give that field a `type` element which is set to xpath. For example, the following sets the `element` field to an XPath chooser:

```
<form xmlns="urn:com/parasoft/extensibility-framework/gui">
    <section label="Main Settings">
        <field id="element" label="Select an element" type="xpath"/>
    </section>
</form>
```

- If you want non-XML input to be converted to XML before the defined execute() method is called, enable the `supportXmlConversion` option in parasoft-extension.xml. Tool users will be able to select the desired message format in the tool's configuration panel. Additionally, if an XPath selector is implemented (see above), its tree view will display messages in the format appropriate for the selected message type. (Note that the the message format is set in the Message Format section of the tool's configuration panel).



- Tables or lists can be implemented as comma-separated values in the string fields.
- Data Sources, Data Bank values, environment variables and test or responder suite variables can be referenced in the extension GUI fields using the `${var_name}` syntax. The standard "Parameterized" and "Scripted" GUI controls can also be used to parameterize fields.

Verifying the New Tool

Verify that your new tool was created and listed among other Parasoft defined tools.