Adding a Custom Transport

This topic explains how to add a custom transport that enables a client tool to send messages over a transport that is not supported by default.

Sections include:

About Custom Transports

Parasoft SOAtest includes a framework that allows you to extend SOAtest's built-in transports and protocols. It can now support any transport or protocol that you are working with—for example, DDS, plain socket or TCP/IP-based messaging, file-based messaging, and messaging based on proprietary APIs. This extension is done using Java.

The SOAP Client, EDI Client, and Messaging Client tools can be customized to display a transport implementation that meets your group's specific needs. Once the custom transport implementation has been plugged in to SOAtest, you can still use the SOAP Client and other messaging client tools' usual WSDL, schema and request message construction capabilities. The actual delivery and reception of messages over the necessary protocol are handled by the transport implementation.

Interfaces to Implement for Custom Transports

After setting up your environment as described in General Procedure of Adding an Extension, implement the following interfaces (described in the Extensibility API documentation):

- com.parasoft.api.transport.ICustomConnection (optional)
- com.parasoft.api.transport.ICustomTransport
- com.parasoft.api.ICustomMessage<T> (or reuse/extend com.parasoft.api.DefaultCustomMessage<T>)

ICustomTransport Implementation

This is a required class. The createNewConnection() method should return an instance of your ICustomConnection class, or null if your protocol does not need it. The invoke() method takes:

- A connection object, which can be reused for multiple invocations.
- A configuration that is never null and that provides the transport implementation class with the values coming from the tool GUI.
- A request that implements ICustomMessage and includes the request message content as provided by the tool GUI.
- A context passed down to other methods in the configuration object when you invoke methods on that CustomTransportConfiguration class. It also includes methods for reporting errors, getting values from data source columns, variables, etc.

One-way (send only) transports can have an invoke() method which returns null.

ICustomConnection Implementation

This an optional class. That means that you can return null in ICustomTransport.createNewConnection() and expect to get null in the invoke() method. However, many protocols tend to have the concept of a session or connection, so the extensibility framework allows for managing that via this interface.

You can expect a new instance of your ICustomConnection object to be created for every invocation of the test if the **Keep connection alive** option is not selected in the transport configuration tool's GUI (under the connection management section). Otherwise, the same instance will be reused as the test execution flows from one test to another using this same transport...until a test closes the connection (If the **Close connection after test execution** option is selected) or until all execution is completed.

The connect() method is invoked by SOAtest before invoke(). It is expected to establish the transport connection within the connection object, or throw an exception if that process fails. close() is invoked at the end of the test execution, unless the "Keep connection alive" option in that test is selected.

The ICustomConnection is expected to encapsulate and carry your connection/session state, but it is not reused among multiple tests unless Keep connection alive is selected.

ICustomMessage

This interface needs to be implemented so that an instance of it can be returned by the ICustomTransport.invoke() implementation. Alternatively, if your message content format is best represented with a string, you can use the default implementation DefaultCustomMessage<T>. The parameterized type can be used for the property (or header) object types keyed with a string. The string returned by getHeaders() is what SOAtest displays in the Headers area of the traffic viewer. The string returned by getBodyString() is what SOAtest displays in the payload (Body) area of the traffic viewer.

Defining parasoft-extension.xml for a Custom Transport

After you have implemented the necessary classes, define parasoft-extension.xml (introduced in General Procedure of Adding an Extension) as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<extension xmlns="urn:com/parasoft/extensibility-framework/extension"
        type="transport"
        name='The name of your transport, appears in the transports menu' description='A more detailed
description'>
  <class>com.mycompany.MyTransport</class> <!-- implements ICustomTransport -->
  <form xmlns="urn:com/parasoft/extensibility-framework/gui">
        <!-- This describes the fields you wish to appear in your transport GUI -->
        <section label="field group 1">
                <field id="key 1" label="field 1"/>
                <field id="key 2" label="field 2"/>
                ...
                <section label="field group 2">
                <field id="key 3" label="field 3" />
        </section>
        <section label="field group 2">
                <field label="field 1" />
                ...
        </section>
        ...
  </form>
</extension>
```

The field labels under the section elements are used as keys to retrieve values from the CustomTransportConfiguration object that is passed in to the various API methods, and which allows for the values provided by the user to be passed in to your implementation. The field labels appear in the GUI that is constructed based on this XML and they are also used to persist the user provided values to the .tst file when it is saved. The section layout does not have a programmatic impact; it merely helps organize the various GUI fields into sections or categories for easy access and usability by the end user.

Verifying the New Transport

After building the project as described in General Procedure of Adding an Extension, restart SOAtest. Verify that the transport name (as specified in parasoft-extension.xml) appears in the Transports menu.

Examples

Example custom transport implementations are available on the Parasoft Marketplace (accessible at marketplace.parasoft.com or your organization's Environment Manager). To use any of these transports, add the jar to the system preferences classpath list, then create a new SOAP Client or other messaging client tool or reopen the .tst file you are working on.

The "Custom Extension" entry in the transports menu will be renamed to reflect an implementation's name. You may use that transport immediately in your tests. The example transport sources can be imported to SOAtest (or Eclipse) using **File> Import> General/Existing Projects into Workspace> Select archive file**. After the import, add [SOAtest install dir]/plugins/com.parasoft.xtest.libs.web_[version]/root/com.parasoft.api.jar to your Java project classpath in order to build the project.

Tips

- When a .tst file is saved with a configured custom transport extension, the file can still be opened with a SOAtest installation that does not have the extension available. However, an error message will be displayed in SOAtest's Console view and the tests using that transport will not execute. Resaving the .tst without the transport extension will result in the custom configuration being lost if the file is reopened again with the extension. This is to avoid "leaks" in a .tst file with unused or outdated configuration data.
- 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.
- The Connection Management section of the transport extension GUI will always appear—regardless of whether it is used.
- 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 that 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>
```

- Tables or lists can be implemented as comma-separated values in the string fields.
- Data Sources, Data Bank values, environment variables and test 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.