General Procedure of Adding an Extension 1

This topic explains the basic procedure for using Parasoft's Extension Framework to extend Parasoft's built-in transports and message formats.

Sections include:

- Adding an Extension
- Adding Multiple Extensions at Once
- API Documentation
- Additional Considerations

Adding an Extension

Parasoft's Extension Framework is standardized across all extension types. Once you understand the basic process, you can apply it to add support for all the various message formats and transports your team is working with. All extension work is done using Java.

1. Add `[install dir]/plugins/com.parasoft.xtest.libs.web_[version]/root/com.parasoft.api.jar` to your Java project classpath. You can also create a new project by going to**File> New> SOAtest>Custom Development> SOAtest Java Project** or**File> New>Virtualize> Custom Development> Virtualize Java Project**; this will create a new Java project that already has that .jar file added to the classpath.

   Building with Maven?

   If you're building your Java project with Apache Maven, you can add the following to your project's pom.xml:

   ```
   <project>
     ...
     <dependencies>
       <dependency>
         <groupId>com.parasoft.soavirt</groupId>
         <artifactId>com.parasoft.api</artifactId>
         <version>9.9.0</version> <!-- should match product version -->
         <scope>provided</scope>
       </dependency>
     </dependencies>
   ...
   <repositories>
     <repository>
       <id>ParasoftMavenPublic</id>
       <name>Parasoft Public Repository</name>
       <url>http://build.parasoft.com/maven/</url>
     </repository>
   </repositories>
   ...
   </project>
   ```

2. Implement the appropriate interfaces. See the following sections for details on implementing the appropriate interfaces:

   Interfaces to Implement for Custom Transports in SOAtest

   Interfaces to Implement for Custom Message Formats in SOAtest

   Interfaces to Implement for Custom Tools in SOAtest

   Interfaces to Implement for Custom Listeners in Virtualize

   Interfaces to Implement for Custom Message Formats in Virtualize

   Interfaces to Implement for Custom Tools in Virtualize

3. Create a `parasoft-extension.xml` file in the default package of your Java project and configure it as appropriate for the item (or items) you're adding.

   a. One parasoft-extension.xml file is expected for each Java project.

b. If you want to add multiple extensions at once, you create one parasoft-extension.xml file that covers all of the extensions you're adding—and you use the top-level `<extensions>`element; see Adding Multiple Extensions at Once for details. Parasoft looks for parasoft-extension.xml files in its classpath under the default package, so each of your parasoft-extension.xml files need to be included on the classpath—by putting the directories each live in (or the jar files each are contained in) on the classpath in the System Properties preferences.

   For details on how to configure this file for a specific type of extension in Virtualize, see:

   Defining parasoft-extension.xml for a Custom Message Format

   Defining parasoft-extension.xml for a Custom Listener

   Defining parasoft-extension.xml for a Custom Tool

   For details on how to configure this file for a specific type of extension in Virtualize, see:

   Defining parasoft-extension.xml for a Custom Transport

   Defining parasoft-extension.xml for a Custom Message Format

   Defining parasoft-extension.xml for a Custom Tool

2. Build your Java project and add it to your system properties classpath area—either as a jar file, a Java project within your SOAtest or Virtualize workspace, or a class folder.
3. Restart SOAtest/Virtualize and verify that the extension appears in the appropriate area.

   For details, see the following Virtualize sections for details:

   Verifying the New Listener

   Verifying the New Message Format

   For details, see the following Virtualize sections for details:

   Verifying the New Transport

   Verifying the New Message Format

Adding Multiple Extensions at Once

If your team is using a number of extensions, you might want to add multiple extensions at once—for example, a bundle that includes all of your custom message formats, custom transports, and message listeners.

To add multiple extensions at once:

1. Instead of creating multiple parasoft-extension.xml files (one for each extension), create one parasoft-extension.xml file in the project and ensure that it describes all of the extensions. Be sure to use the top-level xml element `<extensions>` and define the specific custom extensions as children of this top-level element.
2. Include all of the extensions in your Java project, build it, then add the resulting jar file, Java project, or class folder to the system properties classpath (as described in step #4 above).

For example, a single extension might have a parasoft-extension.xml file such as:

```
<?xml version="1.0" encoding="UTF-8"?>
<extension xmlns="urn:com/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"
     ...
</extension>
```

For a project that includes multiple extensions, you would have a parasoft-extension.xml file like:

```
<?xml version="1.0" encoding="UTF-8"?>
<extensions xmlns="urn:com/parasoft/extensibility-framework/v1/extension">
    <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">
     ...
     </tool>
     ...
    </extension>
    <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 n" label="field n"/>
     </section>
    </form>
   </extension>
     ...
  <extension>
   ...
  </extension>
</extensions>
```

API Documentation

You can access documentation for the Extension framework API via the **Parasoft> Help** menu and look for the Parasoft SOAtest or Virtualize Extensibility API book.

Additional Considerations

- Custom extensions can depend on Java libraries other than com.parasoft.api.jar. For details in SOAtest, see Configuring External Dependencies. For details in Virtualize, see Configuring External Dependencies.
- You can externalize any string displayed in the GUI to support different languages. For details in SOAtest, see Localizing GUI Text.For details in Virtualize, see Localizing GUI Text.
  If you are using an existing extension and later create a new version of that extension that has a different set of GUI options, you can use a version updater to update your saved .tst (SOAtest) or .pva, .pvn (Virtualize) files to adapt them to the new set of options. For details on updating SOAtest artifacts, see Updating GUI Fields for a New VersionFor details on updating Virtualize artifacts, see Updating GUI Fields for a New Version.