Updating GUI Fields for a New Version 1

This topic explains how to update saved SOAtest artifacts (.tst files) and Virtualize artifacts (.pva and .pvn files) that reference existing Extension Framework extensions to adapt them to a new version of the extension that contains a different set of GUI fields. This allows these artifacts to be seamlessly upgraded to a new version of a custom extension without needing to recreate them. Sections include:

About the Version Updater

If you create a new version of an existing extension that has new GUI options, you can use a version updater to transfer or set values to new fields that have been added. To configure this updating, you need to implement the IVersionUpdater and include the <version> element in parasoft-extension.xml.

IVersionUpdater Implementation

This is an optional class that can be provided if you create a new version of an existing extension that has new GUI options. The updater can be used to transfer or set values to new fields that have been added.

If your new version of the extension adds or modifies GUI options without providing an updater class, then the extension GUI fields will only show fields that defined in the current version of parasoft-extension.xml, and will not show any old fields that only existed in an old version of parasoft-extension.xml. Any field values that were saved for those fields will also get removed the next time that the field values for the extension are saved.

The updater passes a saved version number that represents the version of the extension at the time the options were last saved. If a version was not previously specified in parasoft-extension.xml, then the expected version number is zero. The passed-in configuration class can be used to get and set field values, as well as remove any unnecessary fields. Remember that fields which are no longer in the XML will be removed automatically and do not need to be manually removed in the updater.

For example, let's say that you changed your options in the XML to have one field instead of two. In addition, let's say this new field contains the concatenation of the original two field values. Here is an example how this might be done:

```
updateVersion(int savedVersion, ICustomConfiguration config) {
     if (savedVersion == 1) {
         String valueOne = config.getString("oldId1");
         String valueTwo = config.getString("oldId2");
         config.setString(newKey, valueOne + valueTwo);
     }
}
```

<Version> Element in parasoft-extension.xml

This is an optional element that can be included to define the following:

- the version of your extension
- an optional updater class that will update saved Parasoft artifacts from one version of an extension to a new version of the extension

There are two applicable attributes:

- id - An integer version that specifies the current version of the extension. This version will get saved in any Parasoft artifacts that store the field values. The saved version saved in the Parasoft artifacts is what will get passed to the class that implements IVersionUpdater.
- updaterClass - Holds the fully qualified name of the class which implements IVersionUpdater

The field id values under the section elements are used as keys to retrieve values from the ICustomXMLConverterConfiguration object that is passed to the conversion methods. This allows for the user-provided values to be passed into your implementation.

For example:

```
IXMLMessage toXML(INativeMessage nativeMessage, IConversionContext context) throws CustomConversionException
{
     String optionValue = context.getConfiguration().getString("fieldId");
}
```

The field labels appear in the GUI that is constructed based on this XML. They are also used to persist the user-provided values to the .tst or .pva 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.