NAME

genxs – Mono's Xml Serializer Generator

SYNOPSIS

genxs configurationFile [destinationFolder]

DESCRIPTION

genxs is a tool for generating custom XML serialization writers and readers for classes.

configurationFile is configuration file which specifies several information, such as the class for which to generate the reader and writer, the name and namespace of the classes to generate, and a collection of hooks to apply. By using hooks it is possible to customize the behavior of the serializer without needing to modify the generated file, so you can safely regenerate it if the source class is modified.

destinationFolder specifies the folder where the files will be generated.

NOTE: This tool only runs in the Mono runtime, since it uses some internal classes not available in other runtimes.

CONFIGURATION FILE FORMAT

The configuration file is an xml document based on the following grammar ("?" means optional, "*" 0 or more):

```
<configuration>
        <serializer class="name" assembly="name"> *
                <reader>name</reader> ?
                <writer>name</writer> ?
                <namespace>name</namespace> ?
                <outFileName>name</outFileName> ?
                <readerHooks> ?
                        <hook ...> *
                </readerHooks>
                <writerHooks> ?
                        <hook ...> *
                </writerHooks>
        </serializer>
</configuration>
```

A configuration file can have multiple "serializer" elements, each of which specifies the class for which to generate a serializer together with several generation options. The source class is specified in the following attributes:

* *class* : name of the class (including namespace).
* *assembly* : assembly name. It can include the complete path.

Generation options are specified in child elements:

* *reader* : name of the reader class.
* *noReader* : if "true", it does not generate reader class.
* *writer* : name of the writer class.
* *baseSerializer* : name of the base xml serializer class. This item is 2.0 only.
* *implementation* : name of the serializer implementation class. This item is 2.0 only.
* *noWriter* : if "true", it does not generate writer class.
* *namespace* : namespace of the reader and writer classes.
* *generateAsInternal* : if "true", it generates classes as internal.
* *outFileName* : name of the generated file.

* *readerHooks* : a list of hooks to apply to the reader.
* *writerHooks* : a list of hooks to apply to the writer.

SPECIFYING HOOKS

Using hooks you can customize the behavior of readers and writers. A hook specification follows this grammar:

```
<hook type="name">
        <select> ?
                <typeName>name</typeName> ?
                <typeAttribute>name</typeAttribute> *
                <typeMember>name</typeMember> ?
        </select>
        <replace>source code</replace> ?
        <insertBefore>source code</insertBefore> ?
        <insertAfter>source code</insertAfter> ?
</hook>
```

The "type" attribute specifies the context in which the hook is applied. It can be one of the following:

* *attributes* : hook is applied where attributes are serialized/deserialized.
* *elements* : hook is applied where elements are serialized/deserialized.
* *unknownAttribute* : hook is applied where unknown attributes are processed.
* *unknownElement* : hook is applied where unknown elements are processed.
* *member* : hook is applied where a member is serialized/deserialized.
* *type* : hook is applied for the whole type.

The "select" element specifies the classes and members to which the hook has to be added. It can contain the following elements:

* *typeName* : the class with that name will be selected (must include namespace)
* *typeAttribute* : all classes which have that attribute applied will be selected (specify the full attribute class name, including namespace). Several attribute names can be specified.
* *typeMember* : name of the class member for which the hook must be added.

The hook source code can be specified using any of the following elements:

* *replace* : the provided source code will replace all serialization/deserialization operations in the hook context.
* *insertBefore* : the source code will be added before the hook context.
* *insertAfter* : the source code will be added after the hook context.

When writing the code for a hook you can use some special variables that are defined during the code generation process. The variables are the following:

* *$TYPE:* name of the class being generated, without namespace.
* *$FULLTYPE:* full name of the class being generated, including namespace.
* *$OBJECT:* the object being serialized or deserialized. When using a replace reader hook of type "type", the hook code must assign the deserialized object to this variable.
* -I $ELEMENT: name of the element of the object being serialized/deserialized.
* *$NAMESPACE:* namespace of the element of the object being serialized/deserialized.
* *$MEMBER:* name of the member being serialized/deserialized. Only valid in the "member" context.

HOOK EXAMPLES

The following example adds a call to a Validate method after the deserialization of any object:

```
<hook type="type">
        <insertAfter>
                System.Xml.Schema.XmlSchema.Validate$TYPE ($OBJECT);
        </insertAfter>
</hook>
```

This example specifies the code to be used to deserialize the XmlSchema class:

```
<hook type="type">
        <select>
                <typeName>System.Xml.Schema.XmlSchema</typeName>
        </select>
        <replace>
                $OBJECT = System.Xml.Schema.XmlSchema.Read (Reader, null);
        </replace>
</hook>
```

That one specifies the code to be used to read XmlSchema instances:

```
<hook type="type">
        <select>
                <typeName>System.Xml.Schema.XmlSchema</typeName>
        </select>
        <replace>$OBJECT.Write (Writer);</replace>
</hook>
```

With this two hooks the serializer will print some information when serializing the class "MyClass":

```
<hook type="type">
        <select>
                <typeName>MyNamespace.MyClass</typeName>
        </select>
        <insertBefore>Console.WriteLine ("Serializing MyClass");</replace>
        <insertAfter>Console.WriteLine ("MyClass serialized");</insertAfter>
</hook>
<hook type="member">
        <select>
                <typeName>MyNamespace.MyClass</typeName>
        </select>
        <insertAfter>
                Console.WriteLine ("Serialized member $MEMBER");
        </insertAfter>
</hook>
```

This hook writes an additional element for all types that have the custom attribute "MyAttribute":

```
<hook type="elements">
        <select>
                <typeAttribute>MyNamespace.MyAttribute</typeAttribute>
        </select>
        <insertAfter>
                Writer.WriteStartElement ("privateData");
                Writer.WriteString ($OBJECT.PrivateData);
                Writer.WriteEndElement ();
        </insertAfter>
</hook>
```

CONFIGURATION FILE EXAMPLE

This is the configuration file used to generate the serializer for ServiceDescription:

```
<configuration>
        <serializer class="System.Web.Services.Description.ServiceDescription" assembly="System.Web.Services">
                <reader>ServiceDescriptionReaderBase</reader>
                <writer>ServiceDescriptionWriterBase</writer>
                <namespace>System.Web.Services.Description</namespace>
                <outFileName>ServiceDescriptionSerializerBase.cs</outFileName>
                <readerHooks>
                        <hook type="unknownElement">
                                <select>
                                        <typeAttribute>System.Web.Services.Configuration.XmlFormatExtensionPo
                                </select>
                                <replace>ServiceDescription.ReadExtension (Reader, $OBJECT);</replace>
                        </hook>
                        <hook type="type">
                                <select>
                                        <typeName>System.Xml.Schema.XmlSchema</typeName>
                                </select>
                                <replace>$OBJECT = System.Xml.Schema.XmlSchema.Read (Reader, null);</replac
                        </hook>
                </readerHooks>
                <writerHooks>
                        <hook type="elements">
                                <select>
                                        <typeAttribute>System.Web.Services.Configuration.XmlFormatExtensionPo
                                </select>
                                <insertBefore>ServiceDescription.WriteExtensions (Writer, $OBJECT);</insertBefor
                        </hook>
                        <hook type="type">
                                <select>
                                        <typeName>System.Xml.Schema.XmlSchema</typeName>
                                </select>
                                <replace>$OBJECT.Write (Writer);</replace>
                        </hook>
                </writerHooks>
        </serializer>
</configuration>
```

AUTHORS

Lluis Sanchez Gual (lluis@ximian.com)

LICENSE

GenXS is released under the terms of the GNU GPL.

SEE ALSO

mono(1), mcs(1), sgen(1)