XsdToClasses

Michael Coyle, Blue Toque Software

# Overview:

**XsdToClasses** is a simple tool that attempts to generate code from an XML Schema. This process is called Code Generation and is useful for a number of reasons. For example, if you need to specify a schema for the generation of web services or to adhere to a standard, generating the code from such a schema simplifies coding and saves a step. The XML Data can be validated against the schema before deserialization. Generating schemas and the code to serialize and deserialize them allows the programmer to specify rich object models that also have easily verifiable XML storage models.

XsdtoClasses was written based on the framework tool xsd.exe. It uses the same framework classes , and gives the added value that edits to the schema in Visual Studio result in generated code when the user saves the schema or when the project is built. In addition, this tool allows the user to tune the generated code to fit a number of scenarios:

* Control generation of DataBinding code
* including externally generated code files
* Control code generation by excluding types from being generated
* Extend code generation by writing custom CodeModifiers
* Handle the case where one schema references another schema for which you’ve already generated code

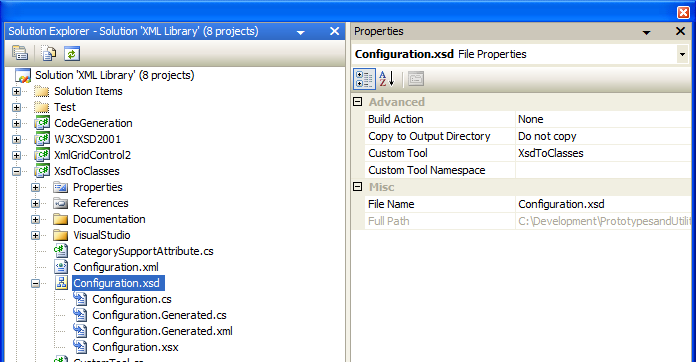
# Usage:

**XsdToClasses** installs as a Visual Studio Add-In.

Select an XSD File. In the Properties dialog for the file there is an entry named “Custom Tool”.

Enter ”**XsdToClasses**” in this entry.

If you wish the code to be generated in a specific namespace, in order to avoid collision of generated code type names with other names, you can either place the schema file in a subdirectory (in which case the schema’s namespace will take on the value of the diresctory), or specify the namespace using the “Custom Tool Namespace” field in the propersty grid.



*Figure 1: XsdToClasses configuration*

After entering the text, the Add-In will activate for the first time, and attempt to generate code from your schema. It will generate two files:

***<schemaName>.Generated.cs***: This file is the generated code. In the case that there was an error generating the code, the error and call stack information will be written as a comment in this file. See below for error information.

***<schemaName>.Generated.xml***: this file contains configurable parameters that affect the code generation. This is where parameters are set that change how or what code is generated.

# Programming:

All code generated using the **XsdToClasses** tool is generated as partial classes. This means each class is simple to extend by adding your own partial class.

Right-clicking on the project, and choose “Add Class”. Name the class the same name as the schema file. This will ensure that the class file is “nested” as a child of the schema file (see figure 1 – *Configuration.cs*). Edit the generated class, removing and adding classes to this file as partial classes are necessary.

# Configuration:

The configuration schema for **XsdToClasses** is included in the installation of the program. Basic features are as follows.

**EnableDataBinding**: causes the code generator to create databinding compatible code. Each class derives from INotifyPropertyChanged, each property calls the PropertyChanged event.

**GenerateOrder**: generates an order attribute that causes the XmlSerializer to enforce the order of elements in the serialized XML.

**GenerateProperties**: causes properties to be created for each element and attribute. If this is set to false, the generator creates public fields. Setting this to false will have the side effect of disabling data binding (see above)

**CodeModifiers**: code modifiers are the heart of the code generator. This configuration section specifies a set of code modifiers that are run in order after the basic code is generated. Each code modifier inspects the CodeDOM and can modify it in some way. Each code modifier can have it’s own configuration section nested in the code modifier description.

**SchemaImporterExtensions**: this set of references can change how the schema is imported. For example, it can change how a custom type is mapped to a framework type, and can even map a ComplexType to an existing framework class.

# Example:

## Schema imports:

When one schema imports another, the code generator can not tell if code has already been generated for that import. In fact, since a schema can contain multiple namespaces internally, the code generor has no knowledge of the import, only that there are types specified which need to be generated.

How XsdToClasses solves this is by two code modifiers: RemoveSpecifiedTypes and ModifyImports.

RemoveSecifiedTypes has a configureation that allows the user to specify which types from the imrpoted schema to skip. The modifier does this by performing a text match on the class name of the generated type.

ModifyImports adds a “Using” statement to the code to reference the generated code in another namespace. If the code is generated in the same namespace then this will not be necessary.

Below is an example of this strategy.

<?xml version="1.0" encoding="utf-16"?>

<Configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://BlueToque.ca/XsdToClasses.Configuration">

<CodeModifiers>

<CodeModifier Type="CodeGeneration.CodeModifiers.ConvertArraysToCollections"

Assembly="CodeGeneration, Version=1.0.1.0, Culture=neutral, PublicKeyToken=46a422f5b652f20b" />

<CodeModifier

Type="CodeGeneration.CodeModifiers.RemoveDebuggerAttribute"

Assembly="CodeGeneration, Version=1.0.1.0, Culture=neutral, PublicKeyToken=46a422f5b652f20b" />

<CodeModifier

Type="CodeGeneration.CodeModifiers.RemoveSpecifiedTypes"

Assembly="CodeGeneration, Version=1.0.1.0, Culture=neutral, PublicKeyToken=46a422f5b652f20b" >

<RemoveSpecifiedTypesOptions xmlns="http://BlueToque.ca/CodeGeneration/CodeModifiers/RemoveSpecifiedTypes.Options">

<Type Name="TagType" />

<Type Name="TagsType" />

<Type Name="TagTypeCollection" />

</RemoveSpecifiedTypesOptions>

</CodeModifier>

<CodeModifier

Type="CodeGeneration.CodeModifiers.ModifyImports"

Assembly="CodeGeneration, Version=1.0.1.0, Culture=neutral, PublicKeyToken=46a422f5b652f20b" >

<ModifyImportsOptions xmlns="http://BlueToque.ca/CodeGeneration/CodeModifiers/ModifyImports.Options">

<AddImport

Name="EmerGeo.MessageDataLib" />

</ModifyImportsOptions>

</CodeModifier>

</CodeModifiers>

<SchemaImporterExtensions />

</Configuration>

*Figure 2: a basic configuration file*

The first part of the file defines the root element and the namespaces.

The CodeModifiers element defines the group of code modifiers. The four code modifiers present are the defaults that are found in the file when it is generated. The “RemoveSpecifiedTypesOptions” element causes the code generator to remove all instances of the given types from the CodeDOM after the basic code is generated. Each <Type> element specifies the name of a type to remove from the CodeDOM.

The ModifyImports element allows the user to specify “using” statements for the generated code. Each <AddImport> element specified the namespace to import. Note that this does not add the reference to that namespace to the project; the programmer must still perform that step.

# Errors:

Possible reasons for an error generating code

* Schema is not valid. If the schema is not valid the call stack will most likely have some sort of schema parsing error.
* Schema does not resolve to a valid representation in code.  
  There are many examples of schemas that cannot be represented as code. In general the XSD language can represent structures that do not have a singular code representation. The **XsdToClasses** code uses framework classes to attempt the resolve these into code structures, but in many cases this is not possible.