Standards DevKit 1.0

License notice
Standards DevKit, version 1.0

The following Energistics (c) products were used in the creation of this work:

- WITSML Data Schema Specifications, Version 1.4.1
- WITSML API Specifications, version 1.4.1
- WITSML Data Schema Specifications, Version 1.3.1.1
- WITSML API Specifications, version 1.3.1
- PRODML Data Schema Specifications, Version 1.2
- PRODML Web Service Specifications, Version 2.0

Quick Start

Generating the DevKit

Note: The DevKit generator requires access to the Energistics schema and API definition files.

1. Open Visual Studio 2010 (You can also use Visual C# Express 2010, but the Unit Tests project will not load)
2. Go to File -> Open -> Project / Solution
3. Open “DataAccess.sln”
4. From the Solution Explorer expand the SchemaGatherer project and double-click App.config
5. Edit and save this file so that it points to all of your external dependencies (schemas, API definitions, etc…). At a minimum, you will need to change “ROOT_FOLDER”.
6. Build the DevKit by going to Build -> Build Solution

Running the sample application

1. You must first generate the DevKit as instructed above
2. From the Solution Explorer right-click the SampleBrowserApp project
3. Click “Set as startup project”
4. Go to Debug -> “Start debugging”

Solution Structure

SchemaGatherer Project

This project contains the first part of the code that is run during the code generation process. It first integrates enumValues.xml into a new copy of type_catalog.xsd. Then it collects all XSD object files from each set (WITSML 1.3.1, WITSML 1.4.1, and PRODML 1.2.1) and runs them through XSD.exe to create the base classes representing the data schema and stores the output in the Generator project.

The SchemaGatherer Project uses settings in App.config to find all required files and folders.

Generator Project

This project is what actually creates the final DevKit code. As a pre-build event, it runs the output assembly from the Schema Gatherer Project to create classes necessary for the Generator to compile and run.

EnergisticsTextTemplate.tt is a Preprocessed T4 Text Template. A T4 Text Template is a mixture of the text as it will appear in the generated class, and fragments of program code. The program fragments supply values for the variable parts of the generated class, and also control conditional and repeated parts. The “TextTemplatingFilePreprocessor” custom tool is run on this file to create EnergisticsTextTemplate.cs, which is instantiated at runtime to do the code generation.

The EnergisticsTextTemplate template iterates through the classes generated from XSD.exe in the SchemaGather project. It renames classes and properties, expands choice elements, adds code summaries to support IntelliSense, and makes other various changes to make the resulting classes more developer friendly, all while still maintaining compliance with the original XSD’s.

The Generator Project uses settings from the same App.config from the SchemaGatherer to find all required files and folders.

EnumValuesExtensionGenerator Project

This project generates an executable that will generate an Extension class given a custom enumValues.xml file. (See “Generating Extension Class” under “Using EnumValues Extensions” below)

DataAccess Project

This project is the actual DevKit itself that produces a signed DLL. As a pre-build event, it runs the output assembly from the Generator Project to generate the XSD specific portion of the DevKit.

The DataAccess Project is divided into the following namespaces:

Namespace	Description
Energistics.DataAccess	Contains helper classes to do object/xml conversion and file manipulation for all supported Markup Languages. Also contains Web Service wrappers specifically for WITSML.
Energistics.DataAccess.WITSML131	Contains object representation of WITSML 1.3.1
Energistics.DataAccess.WITSML131.WMLP	Contains classes to access WITSML 1.3.1 Subscribe/Publish Application Program Interface (API)
Energistics.DataAccess.WITSML131.WMLS	Contains classes to access WITSML 1.3.1 Client/Server Application Program Interface (API)
Energistics.DataAccess.WITSML141	Contains object representation of WITSML 1.4.1
Energistics.DataAccess.WITSML141.WMLS	Contains classes to access WITSML 1.4.1 Client/Server Application Program Interface (API)
Energistics.DataAccess.PRODML120	Contains object representation of PRODML 1.2.0
Energistics.DataAccess.PRODML120.PROD_AssetRegistry	Contains classes to access PRODML 1.2.0 SAM Asset Registry Service
Energistics.DataAccess.PRODML120.PROD_GenericDataAccess	Contains classes to access PRODML 1.2.0 Generic Data Access Service
Energistics.DataAccess.PRODML120.PROD_GenericDataProcessor	Contains classes to access PRODML 1.2.0 Generic Data Processor Service

Issues / Areas for Improvement

- When xsd.exe parses typ_catalog.xsd, it converts simple types with no restrictions defined to strings. This causes an issue because the restrictions are provided later by EnumValues, which cannot be applied if the simple types have already been converted to strings. The current solution is to dynamically insert the restrictions from enumValues.xml into typ_catalog.xsd which preserves the simple type. However, this is more work than should be required as these inserted restrictions are discarded in the end.
- When parsing the .xsd files for annotation fields, the generator uses Regular Expressions. While this works in most cases, it would be safer to change this implementation to use XPath queries.
- The DevKit currently does not handle WITSML 1.4.1 references to the GML CRS standards
- While PRODML web services are accessible through the DevKit, there is no wrapper class like there is provided for WITSML.
- The WMLS wrapper class does not provide a way to set the “Options In” parameter.
- The WMLS wrapper class should cache the WMLS object instead of creating a new one each call
- No conversion values for UOM provided from witsmlUnitDict.xml
- When de-serializing an object that contains an EnumValue, if that EnumValue is not already defined, a custom EnumValue will be created automatically.
- Add RESQML
- DevKit is not CLS-compliant because multiple enumerations contain values differing only in case

