

Management protocol

Contents

- [Contents](#)
- [1. CVS Versioning](#)
- [2. CVS upload procedure](#)
- [3. Who may change source code](#)
- [4. Test procedure](#)
- [6. The iterative development approach](#)
- [7. UML files](#)
- [8. Installing assemblies](#)
- [9. Change Requests, Support Requests and Bug Reports](#)
- [10. Maintaining the .NET and JAVA versions](#)
- [11. C # style guide](#)

1. CVS Versioning

Different versions of the OpenMI source code will be found underneath the "Source" module on the CVS server. The versions will be:

OpenMI.1.2.0 - a copy of the original OpenMI.1.1.0 compiled with Microsoft Visual Studio 2005, used to create patches to the released version. A patch to version 1.2.0 should be versioned 1.2.1.

OpenMI.1.3.0 - used to make improvements to the 1.x line

OpenMI.2.0.0 - used for the new implementation of OpenMI

All versions should be compiled under Microsoft Visual Studio 2005. The Java implementations will be located in parallel to the C# code under each version.

2. CVS upload procedure

Before uploading any source code to the CVS server all code must be compiled locally. There cannot be any compiler errors in any other part of the solution prior to uploading the source code.

3. Who may change source code

Any developer accepted on the SourceForge website may make small changes to the code. A suitably descriptive comment should be added to the CVS upload comment so it can be used in the release notes. Major changes, and ALL changes made to the Standard, have to be decided beforehand at the technical meetings.

4. Test procedure

All OpenMI software is to be unit tested. The tests are to be made using NUnit for .NET code and JUnit for Java code. NUnit and JUnit are unit-testing frameworks that are distributed freely as OpenSource.

The official homepage for NUnit is: www.nunit.org NUnit may be downloaded from this site.

The official homepage for JUnit is www.junit.org JUnit may be downloaded from this site.

The basic idea of unit testing is to have a test class for every developed class and to have test methods for every (public) method within the developed class.

Test namespace:

The namespaces for the test code should be identical to the namespaces used for the source code with .UnitTest appended.

Example:

- Source Code: org.OpenMI.Utilities.Spatial
- Test Code: org.OpenMI.Utilities.Spatial.UnitTest

Arranging test code:

Test code is found in a sub-folder named 'UnitTest' underneath each individual project. All test code and corresponding test data should be placed in the UnitTest folder.

What to test:

All public methods should be tested as a minimum. Further, protected methods may be indirectly tested by inclusion of a wrapper class in the test class.

Test methods for protected methods are named Protected_<Method Name of tested method>.

An example of protected method testing may be found in org.OpenMI.Utilities.Buffer.UnitTest\SmartBufferTest.cs.

6. The iterative development approach

The developments of software packages will be done iteratively. At each of the planned meetings we will complete an iteration and start the next iteration. In each iteration we will perform the following tasks

- Make implementations needed for the suggested improvement
- Make changes to the overall architecture when needed
- Review the requirements and adjust when needed
- Make code review

7. UML files

Visual Studio will be used to create UML diagrams from the source code. UML diagrams will be created every time there is a new implementation and put on SourceForge in the documentation for every class. An overall UML diagram will be created as a pdf document and also put on SourceForge.

8. Installing assemblies

Strong Names:

In order to make an assembly Strong Named a key pair (public/private key pair) must be referenced. The key pair is referenced from the AssemblyInfo.cs file by setting the AssemblyKeyFile property.

[assembly: AssemblyKeyFile(<RelativePath>"OpenMI.snk")]

The file key pair file to be referenced is: \\SourceForgeOpenMI\Source\OpenMI.x.x.x\DotNet\OpenMI\Deployment\OpenMI.snk where OpenMI.x.x.x is the OpenMI version.

Versioning:

.NET opens the possibility of having multiple versions of an assembly installed. This is to be used within the OpenMI.

OpenMI works with a three digit version number (major, minor, build).

The version is to be specified in the AssemblyInfo.cs file by setting the AssemblyVersion property to e.g. 2.0.0. [assembly: AssemblyVersion("2.0.0")].

9. Change Requests, Support Requests and Bug Reports

Change requests, support requests and bug reports specific to OpenMI should be recorded in the relevant forum on SourceForge - <https://sourceforge.net/projects/openmi/>. There are four forums available - Developers, Feature requests, Help and Open Discussion. All change requests and bug reports entered into the forums will be discussed at the technical meetings. All support requests will be dealt with by the technical team asap.

Change requests, support requests and bug reports specific to individual models should be reported to the model providers support desk.

10. Maintaining the .NET and JAVA versions

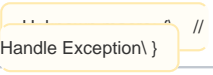
The aim is to keep both versions as synchronised as possible. This is an absolute must for the Standard, however other parts may divulge. Java developers are responsible for telling .NET developers when small extensions and bug fixes are made, and vice versa. The Developer Forum on SourceForge should be used as the means of communication. Major changes will be discussed at technical meetings. The synchronisation between the two versions will be discussed at each meeting.

11. C # style guide

The purpose of coding standards and coding guidelines is to improve productivity and quality. If everyone on a development team follows the same standards, the result is source code that is easier to read by all members of the team because it is written in a consistent style. Easy to read code is then easier to debug and maintain by all members of the development team.

Styles defined	Description	Example
Pascal case	The first letter in the identifier and the first letter of each subsequent concatenated word are capitalized	BackColor DataSet
Camel case	The first letter of an identifier is lowercase and the first letter of each subsequent concatenated word is capitalized	numberOfDays isValid
Hungarian type	Hungarian notation is any of a variety of standards for organizing a computer program by selecting a schema for naming your variables so that their type is readily available to someone familiar with the notation. It is in fact a commenting technique.	strFirstName iNumberOfDays

Naming		
	Description	Example
Fields	Prefix private or protected variables with a "_" and use Camel case for the name	// Field private OleDbConnection _connection; protected double _waterLevel;
Local variables	Use Camel case or Hungarian-style notations as you find most appropriate.	double lowFlow; string strRiverName; string riverName;

Class names	Use Pascal case Use a noun or noun phrase to name a class. Do not use a type prefix, such as C for class, on a class name. Do not use the underscore character (_). Classes may begin with an "I" only if the letter following the I is not capitalized, otherwise it looks like an Interface. Classes should not have the same name as the namespace in which they reside.	LinkableComponent Element Node TimeSpan
Collection classes	Follow class naming conventions, but add Collection, Set, or List to the end of the name. Alternatively add a plural s to the end of the name	NodeCollection NodeList NodeSet Nodes
Collection of collection classes	Follow the collection class conventions but add a plural s the end of the name	NodeCollections NodeLists NodeSets
Interfaces	Use Pascal case. Prefix interface names with the letter "I", to indicate that the type is an interface. Do not use the underscore character (_).	INodeCollection
Methods	Use Pascal case Use verbs or verb phrases to name methods.	GetValues(...)
Property	Use Pascal case Use a noun or noun phrase to name properties.	NodeList
Parameter	Use Camel case	myModel
Events	Use Pascal case Use and EventHandler suffix on event handler names. Specify two parameters named sender and e. The sender parameter represents the object that raised the event. The sender parameter is always of type object, even if it is possible to use a more specific type. The state associated with the event is encapsulated in an instance of an event class named "e". Use an appropriate and specific event class for the e parameter type. Name an event argument class with the EventArgs suffix.	public delegate void EventHandler(object sender, EventArgs e);
Exceptions	Event handlers in Visual Studio .NET tend to use an "e" parameter for the event parameter to the call. To ensure we avoid a conflict, we will use "ex" as a standard variable name for an Exception object.	catch (Exception ex)  Handle Exception }
Constants	The names of variables declared class constants should be all uppercase with words separated by underscores. It is recommended to use a grouping naming schema.	AP_WIN_MIN_WIDTH, AP_WIN_MAX_WIDTH, AP_WIN_MIN_HEIGHT, AP_WIN_MAX_HEIGHT
Namespaces	Must start with org.OpenMI. For the remaining part of the name use Pascal case.	org.OpenMI.System. Components org.OpenMI.Utilities.Buffer
Assembly names	Only one namespace for each assembly. Assembly names must be "namespace" + ".dll"	org.OpenMI.Utilities.Buffer.dll
C# file names	Put every class in a separate file. The file name must be "classname"+" .cs"	ElementSet.cs
XML comments file names	XML comments file name must be Assembly name + ".xml"	org.OpenMI.Utilities.Buffer.xml
Directory names	Create a directory for every namespace. (For org.OpenMI.Utilities.Buffer use C:\HarmonIT\SourceCode\DotNet\OpenMI\Utilities\Buffer as the path, do not use the namespace name with dots.) This makes it easier to map namespaces to the directory layout.	C: \HarmonIT\SourceCode\DotNet\OpenMI\Utilities\Buffer

Project file name	The project file name must be "assembly name" + ".csproj"	org.OpenMI.Utilities.Buffer.csproj
-------------------	---	------------------------------------

Lines wrapping, indentation, white space, etc...		
	Description	Example