

University of West Bohemia in Pilsen
Faculty of Applied Sciences
Department of Computer Science and Engineering

MVE - 2

MMDoc handbook

Author: **Petr Dvorak**
Date: *December 15 2005*

1. Introduction

1.1. Abstract

This document describes functionality and design of the *MMDoc* application. It's an automatic generator of the documentation to the *MVE2* libraries.

Program creates the documentation from XML file built up by Microsoft Visual Studio .NET and compiled library of MVE2 modules and data structures. The resulting documentation should be saved into one file (chm, mht or pdf).

1.2. Used short cuts and terms

XML (eXtensible Markup Language) – Set of rules for creating tagged languages e.g. HTML.

MHT (Mime HTML) – This format is used for packing of whole HTML page including images. It is based on format of email.

CHM – Help in format HTML Help 1.x

HTML Help Compiler – Program developed by *Microsoft*, which creates chm help.

CSS (Cascading style sheets) – This format defines an appearance of web pages.

XSLT (eXtensible Style Language Transformation) – It's used to convert of *XML* to different file types such as *XHTML*, *Tex*, etc.

ADO (Microsoft ActiveX Data Objects) – Libraries used for simplify access to a data.

CDO (Collaboration Data Objects) – This library is used for creation of Mime HTML in *MMDoc*.

2. Implementation

2.1. Description of input

Program reads information from the comments of source code stored in *XML* generated by *Visual Studio 2003* and metadata compiled with *MVE2* library. It actually means that XML file and *MVE2* library of modules and data structures are inputs. Both must have a same name (name of XML file generated by *Visual Studio* must be set in menu *Project->Properties->Configuration Properties->Build->XML Documentation File*).

2.2. Description of output

Program is mainly designed to generate a documentation of *XHTML*, *CHM*, *MHT* (Mime Html) and *DOC* types. There can be chosen output into XML (merged information from input XML and library), input files for *HTML Help Compiler* and *XHTML* in one file. But they are rather intermediate formats used in the program (see bellow). On the other hand they're useable somewhere else (e.g. for a converting into *HTML Help 2.x*).

2.3. Architecture

At begin the `mmdoc` application was only program for command line. But possibilities of `mmdoc` are still extending and it would be for user more difficult using the application. So the one project `mmdoc` has been divided in three projects.

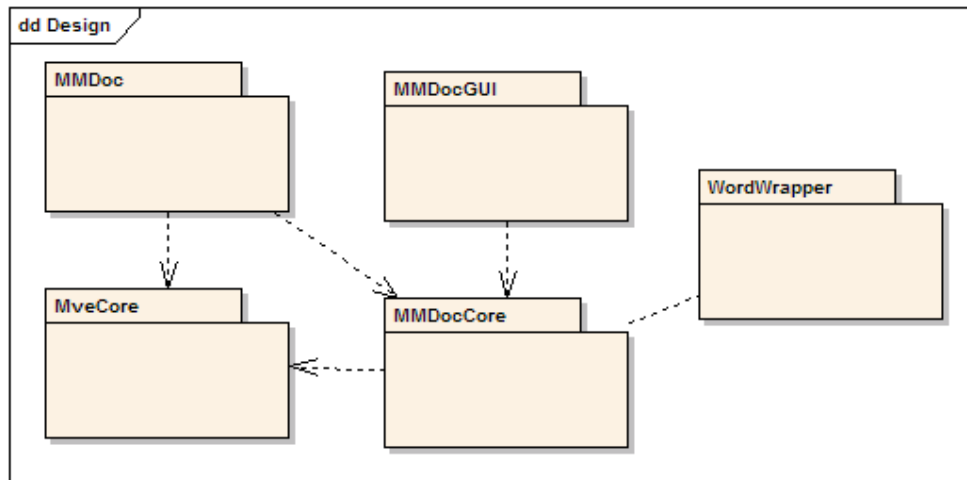
`MMDoc` now contains only statements for communicating with user from command line. The majority functionality, such as reading *XML* and libraries, parsing *XML*, converting etc., has moved in `MMDocCore`. And last it's *MMDoc Graphical User Interface*, which simplifies generating of documentation.

Some goals have been found out during the analysis:

1. Simple interface of `MMDocCore` for using in both *command line mmdoc* and *MMDoc Graphical User Interface*. On the other hand scalability has to be ensured (e.g. adding further parameters for generating).
2. At least same functionality of `MMDocGUI` and `mmdoc` has to be provided.
3. Saving and loading parameters for generating entered by user.
4. Unified reporting of results and errors.

2.4. Design

Assembly dependences:



Dependency between `MMDocCore` and `MveCore` is important. `MveCore` defines *MVE2* attributes used in documentation. Next `MveCore` contains `Module` class and `IDataObject` interface, which are necessary for finding of modules and data structures. Finally there is definition of `AbstractConfig` used for configuration files processing.

Interface of `MMDocCore` is mainly composed by classes `LibraryDoc`, which provides all functionality, and `Options` for passing parameters of compilation. `Options` can be extended easily on demand.

MMDocGUI version of `Options` is named `OptionsGUI`. It's the same class that has some *.NET* attributes assigned to the properties. It has no methods with serialization handling (see below).

MMDocGUI has more features than *command line mmdoc*. There's better user input check and ability to save and load compilation parameters. To do that *XML* serialization is used.

Reporting results and errors are retained by `MMDocCore`. It writes on the console. So it had to be redirected. Own class `RedirectStream`, which is subclass of `StringWriter`, was defined. This type can be passed to `Console.SetOut`, which redirects console output.

`WordWrapper` is assembly, which allow access to the *Word* component. It's detached from core to leave out dependency on *Word* installation.

2.5. Core analysis

During the analysis it has been found out the several criteria, which had to be satisfied:

1. For one library are two data inputs (i.e. `xml` and `dll` library)
2. Several libraries can be input of the program
3. It's desirable interconnect libraries by links
4. Easy to edit the source code (At the beginning it isn't known, what items will documentation involve), preservation of clarity
5. Extendable for other output formats

Therefore there was separated process of gathering information and their presentation i.e. processing into *XML* and their subsequent converting through *XSLT* and *CSS*. When we want to generate another *XHTML*, then it is only matter of writing *XSLT* template (without editing of source code).

Because of interconnection of libraries it had to be read all information from *XML* and *DLL*, then compare it to get interconnections of libraries by links, control of resources, etc. and finally create *XML*. It can be an independent process.

This method has also disadvantages:

- for bigger *XML* files can be rather slower
- *XSLT* templates programming isn't trivial

2.6. Algorithm

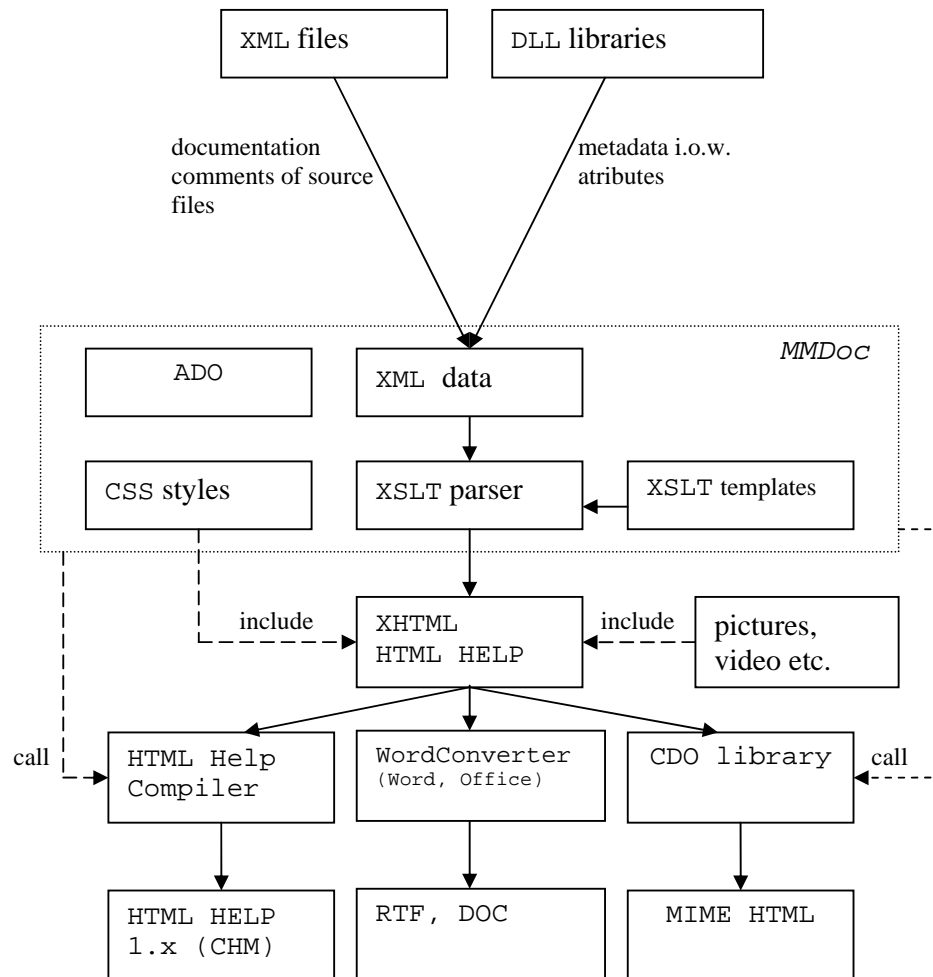
At first the program saves input information from libraries and *XML* into one *XML* data structure.

In next step *XML* data is passed to the *XSLT* parser together with *XSLT* templates. The parser generates several *XHTML* files according to number of modules and data structures. Alternatively project files for *HTML Help Compiler* are generated yet. The compiler is executed at the last step, so `chm` file is created. Similarly it can be chosen a creating of *MHT* using *ADO* and *CDO* and *DOC* (*RTF*) using office wrapper named `WordConverter`.

The structure of output files is assigned by XSLT templates and appearance by CSS styles.

In documentation comments can be used also *HTML* tags. It permits among other things to insert into documentation also images through the tag `img` (``). These images will become a part of the final documentation.

2.7. Diagram of activity



2.8. Core classes

LibraryDoc

The class contains information in member variables and methods for creating documentation of all entered libraries. It calls XSLT transformations of XML (about 12 templates).

Option

It contains parameters passed from user interface to the mmdoc core.

XmlResourceResolver

It's `XmlResolver`, which reads included XSLT templates as a resource (standard way is as a file).

ChmCompiler

It executes *HTML Help Compiler*.

Library

It contains information in member variables and methods for one particular library.

XmlBaseEntity

The abstract class with methods common for both modules and data structures to create its XML representation.

XmlDataObject

The class creates XML representation of data structure.

XmlModule

The class creates XML representation of module.

CustomMembers

It processes information about method and constructors for simple insertion into XML.

CustomPropertyInfo

It processes information about properties for simple insertion into XML.

Speed

It measures running time of the bounded program part. Mainly it serves for display of time of documentation compilation. But it can be used for debugging.

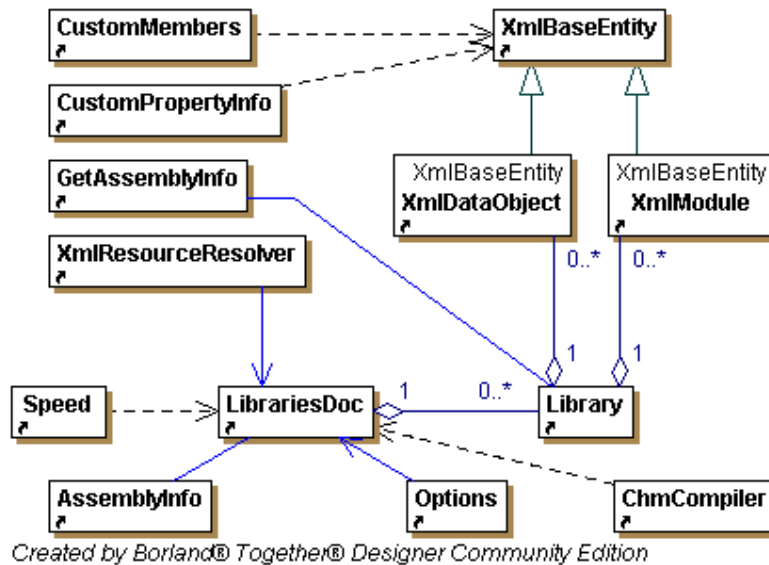
AssemblyInfo

It contains information about mmdoc represented by attributes.

GetAssemblyInfo

The class contains methods for obtaining the assembly attributes.

2.9. Core UML



3. User manual

3.1. Preconditions

The convention of commenting (described in document *Commenting of MVE2 library*) has to be complied to create high-quality documentation during programming of MVE2 library.

Further rules for creation of the modules and data structure must be kept. Especially, that the modules must be subclasses (direct or indirect) of `Zcu.Mve.Core.Module` class and data structure or its parent class (e.g. `Zcu.Mve.Core.DataObject`) must implement interface `Zcu.Mve.Core.IDataObject`.

3.2. Program description

Program creates the documentation to MVE2 library (or libraries).

The XML file and the library with modules and structures are an input. Both must have same name (name of XML file generated by Visual Studio is set in menu item - *Project->Properties->Configuration Properties->Build->XML Documentation File*).

Program is mainly designed to generate a documentation of XHTML, CHM, MHT (Mime Html) and DOC types. There can be chosen output into XML (merged information from input XML and library), input files for HTML Help Compiler and XHTML in one file.

There are two versions of mmdoc for:

- command line
- graphical user interface

3.3. Command line MMDoc

3.3.1. Execution

Sample of output:



```
C:\WINDOWS\system32\cmd.exe
D:\Projekty\mve\svn\Mve2-src-2005\MMDoc\bin\Debug>mmdoc.exe -l Examples -dir ./i
mg -name lib-doc
Start of documentation generating.
Start of loading Examples types.
End of loading Examples types.
Start of joining information about libraries into one XML file.
Start of converting Examples types into XML.
End converting Examples types.
End of joining information about librarians.
Saving XML.
Start of moving files from D:\Projekty\mve\svn\Mve2-src-2005\MMDoc\bin\Debug\ing
into D:\Projekty\mve\svn\Mve2-src-2005\MMDoc\bin\Debug\lib-doc\.
End of moving files.
Start of HTML files generating.
  index-Examples.html
  index.html
  Examples.TetraMeshSlicer.html
  Examples.TetraMeshSlicer-dialog.html
  Examples.TetraMeshSlicer-members.html
  Examples.PointToTetrahedronDistanceSampler.html
  Examples.PointToTetrahedronDistanceSampler-dialog.html
  Examples.PointToTetrahedronDistanceSampler-members.html
  Examples.ObjLoader.html
  Examples.ObjLoader-dialog.html
  Examples.ObjLoader-members.html
  Examples.NormalComputer.html
  Examples.NormalComputer-dialog.html
  Examples.NormalComputer-members.html
  Examples.VertexEdgeDetector.html
  Examples.VertexEdgeDetector-dialog.html
  Examples.VertexEdgeDetector-members.html
  Examples.PointToTetrahedronDistanceEvaluator.html
  Examples.PointToTetrahedronDistanceEvaluator-dialog.html
  Examples.PointToTetrahedronDistanceEvaluator-members.html
  Examples.TetraMeshComposer.html
  Examples.TetraMeshComposer-dialog.html
  Examples.TetraMeshComposer-members.html
  Examples.Sumator.html
  Examples.Sumator-dialog.html
  Examples.Sumator-members.html
  Examples.PromennePorty.html
  Examples.PromennePorty-dialog.html
  Examples.PromennePorty-members.html
  Examples.FrameExtractor.html
  Examples.FrameExtractor-dialog.html
  Examples.FrameExtractor-members.html
  Examples.Sinus.html
  Examples.Sinus-dialog.html
  Examples.Sinus-members.html
  Examples.GenerateGraph.html
  Examples.GenerateGraph-dialog.html
  Examples.GenerateGraph-members.html
  Examples.NumberSource.html
  Examples.NumberSource-dialog.html
  Examples.NumberSource-members.html
  Examples.Convolution.html
  Examples.Convolution-dialog.html
  Examples.Convolution-members.html
  Examples.AllEventSumator.html
  Examples.AllEventSumator-dialog.html
  Examples.AllEventSumator-members.html
  Examples.ScalarNumber.html
  Examples.ScalarNumber-members.html
End of generating.

Documentation was successfully generated.
Time: 2.42 s
```

Command line MMDoc is intended for batch processing. It is executed by command:


```
mmdoc.exe -l module
```

E.g.:

```
mmdoc.exe -l MveCore
```

- It creates the documentation to `MVECore.dll` in the directory `MVECore`.

```
mmdoc.exe -l MveCore -l Visualization
```

- It creates the documentation to `MveCore.dll` and `Visualization.dll` in the directory `MveCore`.
- It's usefull (but not a condition) that the both libraries are in the same directory.

```
mmdoc.exe -l MveCore -name Core
```

- It creates the documentation to `MveCore.dll` in the directory `Core`.

```
mmdoc.exe -l MveCore -title „Title of the documentation“
```

- It can define title used in documentation.

```
mmdoc.exe -l MveCore -w
```

- If entered, so warnings about missing comments and attributes are shown.

```
mmdoc.exe -l MveCore -noprotected
```

- If entered, so protected members won't be included.

3.3.2. Setting of output format:

```
mmdoc.exe -l MveCore -html
```

- It creates html documentation to `MVECore.dll` in the directory `MVECore`.

```
mmdoc.exe -l MveCore -chm
```

- It creates chm documentation to `MveCore.dll`.
- *Microsoft Html Help Workshop* must be installed on the computer to proper function of this feature or *Html Help Compiler* (part of *Html Help Workshop*) has to be copied into the same directory with `mmdoc.exe`.

```
mmdoc.exe -l MveCore -mht
```

- It creates the documentation in `mht` format.

```
mmdoc.exe -l MveCore -hhc
```

- It creates the documentation and the project files for `Html Help Compiler` to `MVECore.dll`

```
mmdoc.exe -l MveCore -xml
```

- It creates `xml` file with information from `MveCore.xml` and attributes in `MveCore.dll`.

```
mmdoc.exe -l MveCore -lhtml
```

- It creates html documentation with one `html` file.

```
mmdoc.exe -l MveCore -doc
```

- It creates documentation for opening in *Microsoft Word*.

```
mmdoc.exe -l MveCore -rtf
```

- It creates documentation for opening in *Microsoft Word*.

These switches can be omitted. Last used one is the default switch.

3.3.3. Pictures insertion

Author of library can use tag `` in documentation comments, where defines relative path to the picture. The switch `-dir` specifies an absolute directory for these relative paths. Naturally it can be used an absolute path right in the `` tag.

```
mmdoc.exe -l MveCore -dir "D:\\img"
```

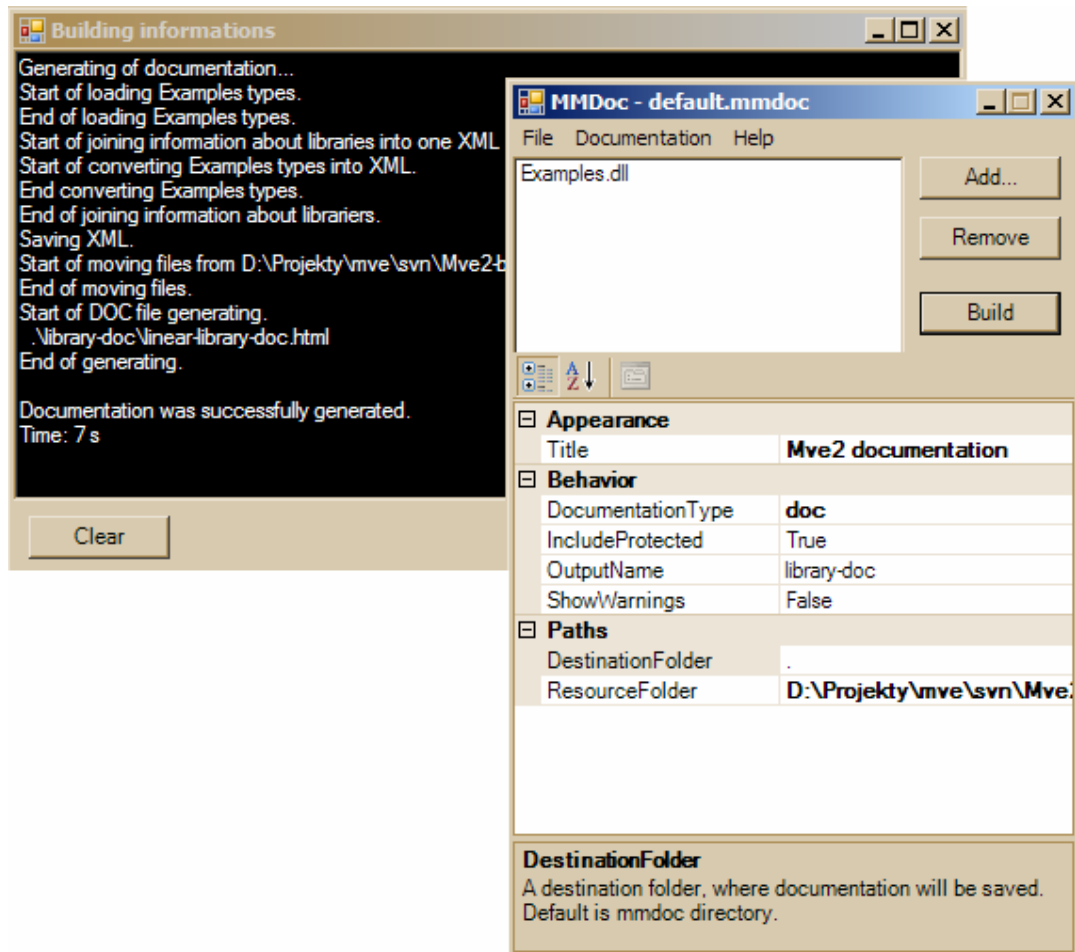
- It creates the documentation to `MveCore.dll` in the directory `MveCore` and copies into it a content of `D:\img`.

Example: There are tags `` and `` in the documentation comments to `MveCore`. There must be the file `delay.jpg` and the directory `Examples` with file `sumator.jpg` to correctly process of the example above.

Note: Other option is to omit the switch `-dir` and simply copy the pictures into the output directory.

3.4. Graphical user interface for MMDoc

Graphical user interface simplifies using of `mmdoc` and entering the parameters.



In menu *File* can be chosen opening, saving and creating of new project. Project contains compilation information such as type of output documentation, path of input files etc. These information user sets in bottom part of main window (in `PropertyGrid` known e.g. from *Visual Studio 2003*).

Notice: For creating *chm* help, *HTML Help Workshop* must be installed (see link at the end of document).

List of libraries, for which documentation is built, can be edited in upper part by buttons *Add...* and *Remove*.

Compiling is started by menu item *Documentation->Build* or key `F5`. Window with progress of compilation and results will be showed.

If any problem occurs, visit *MVE2* homepage by clicking *Help->MVE2 Online*.

4. Conclusion

Many good features are added. The third version is well usable program (e.g. styles and templates are compiled into `mmdoc.exe`). Next versions extend generated documentation with further information (list of the class members and their description, list of interfaces, changes for reaching quicker running etc.).

5. Links

Html Help Workshop: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/hwMicrosoftHTMLHelpDownloads.asp>

6. Appendix

6.1. Examples of final documentation

Overview of library in the generated documentation:



MVE2 Module Documentation

Examples

Title: *Modular Virtual Environment 2 - Example*

Description: *Example of MVE2 library.*

Company: *University of West Bohemia in Pilsen, Czech Republic*

Copyright: © 2004 - 2005 *University of West Bohemia, Czech Republic*

Version: *1.0.1936.19787*

Runtime version: *1.1.4322*

Modules

Examples.AllEventSumator	Sample module. It sums two input numbers. Reaction to all events is implemented.
Examples.Convolution	Module performs a convolution of a image.
Examples.GenerateGraph	Module creates 100 samples of connected function in $\langle 0, 10 \rangle$ interval.
Examples.PromennePorty	Sample module. Example of varying ports.
Examples.NumberSource	Sample module. Source of one random scalar number.
Examples.Sinus	Module calculates sinus of given number.
Examples.Sumator	Sample module. It sums two input numbers.

Data structures

Examples.ScalarNumber	One scalar number (double)
---------------------------------------	----------------------------

Overview of data structure in the generated documentation:



MVE2 Module Documentation

Examples.ScalarNumber

[Index](#) | [Overview](#) | [Members](#)

Sample data class. It represents one scalar number (accuracy as double type in .NET).
Can be shared by modules.

Inheritance:


```
System.Object
  Zcu.Mve.Core.DataObject
    Examples.ScalarNumber
```

Implements:


```
Zcu.Mve.Core.IDataObject
```

Created by MMDoc 20.4.2005 12:00:08

Module settings (*InvokeSetup* comment) in the generated documentation:



MVE2 Module Documentation

Examples.NumberSource 

[Index](#) | [Overview](#) | [Settings](#) | [Events](#)

You can choose from two modes of function of module:

1. Generating of random number:



2. Generating of defined number (2.0 in this case):



Created by MMDoc 12/12/2005 8:59:58 PM