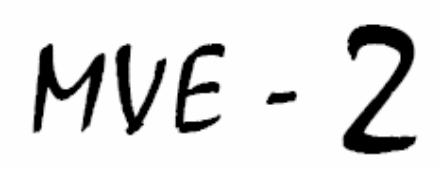
Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky



Komentování MVE2 knihovny

Vypracoval: **Petr Dvořák**
Datum: *15. prosince 2005*

1. Úvod

Tento dokument obsahuje návod k tomu, jakým způsobem psát komentáře a atributy ke knihovnám modulů a datových struktur *MVE2 projektu* tak, aby vznikla sebepopisující se jednotka.

2. Základní pojmy

2.1. Atributy

Jazyk C# umožňuje vkládat do zkompilovaného souboru další informace o třídách, metodách, návratových hodnotách apod. K vkládání těchto *metadat* se používá tzv. atributů. Pojmem atributy se ovšem zpravidla označují členské proměnné tříd. V tomto textu však budou oba pojmy rozlišovány. Atributy budou označovány příkazy pro vložení metadat.

Atribut se vkládá příkazem:

```
[Jméno_atributu(Parametr1, Parametr2, Nepovinný_parametr = hodnota)]
element_kódu //definice třídy, metody, členské proměnné…
…
```

např.:

```
[ModuleInfo("Miroslav F.", "Module is useful for cycles", IconName = „Delay.ico“)]
public class DelayModule : Module
{
  ...
}
```

kde `ModuleInfo` je jméno atributu, `IconName` je nepovinný parametr a `DelayModule` je element kódu, k němuž atribut náleží.

3. Atributy knihovny

Při založení nové knihovny modulů se k projektu automaticky přidá soubor `AssemblyInfo.cs`, který obsahuje atributy, jež se vztahují k této knihovně. Je vhodné nastavit minimálně:

- `AssemblyTitleAttribute(string)` – název knihovny
- `AssemblyDescriptionAttribute(string)` – k čemu slouží
- `AssemblyCompanyAttribute(string)` – společnost, která je vlastníkem autorských práv
- `AssemblyCopyrightAttribute(string)` – autorská práva

Například v jádře MVE2 (`MVECore.dll`) jsou v `AssemblyInfo.cs` doplněny tyto řádky:

```
[assembly: AssemblyTitle("Modular Visualization Environment 2 - Core")]
[assembly: AssemblyDescription("Assembly contains pivotal classes of MVE2.")]
[assembly: AssemblyCompany("University of West Bohemia in Pilsen, Czech Republic")]
[assembly: AssemblyCopyright("Milan Frank 2003 - 2004")]
```

4. Atributy modulů

4.1 Jak psát atributy

Veškeré informace o modulu jsou uloženy ve zdrojovém kódu modulu dvěma způsoby:

- atributy
- komentáře kódu

Atributy jsou nepovinné, ale je opravdu vhodné je používat. Používají se jednak k vygenerování uživatelské dokumentace (programem `mmdoc.exe`) a jednak za běhu programu (tooltipy, výpis seznamu modulů, detailní informace o modulu atd.). Proto by měli být stručné a výstižné. K jednoduchému formátování řetězců (např. u popisu funkce modulu či portu, které mohou být delší) lze požít použít znakové konstanty (nejčastěji `\n` – nová řádka, ale samozřejmě i ostatní znakové konstanty).

K modulům lze přidat 2 typy atributů:

- `ModuleInfoAttribute`
- `PortInfoAttribute`

K property modulu lze připsat standardní .NET atributy:

- `DescriptionAttribute`
- `BrowsableAttribute`

Tyto atributy jsou alternativou ke komentáři u metody `InvokeSetup` (viz kapitola *Komentáře modulů*). Popisují jednotlivé nastavitelné položky modulu v dialogovém okně s použitím `PropertyGrid`u.

Pozn.: Slovo Attribute lze vynechat, překladač si ho automaticky přidá.

4.2 ModuleInfoAttribute

`ModuleInfoAttribute` obsahuje informace o modulu. Atribut lze vložit pouze jeden před definici třídy.

Má 2 povinné parametry (řetězce):

- *autor* modulu
- stručný *popis* funkce modulu

3 nepovinné parametry (řetězce):

`IconName`

- Jméno ikony ve zkompilované knihovně, která reprezentuje modul (přidání viz dále).
- Pokud není zadána použije se výchozí ikona z jádra.

`Assemblied`

- Datum verze modulu (zadává se ve formátu `RRRR-MM-DD` … interně uložen jako typ `DateTime`).
- Pokud není zadán použije se datum kompilace.

`Category`
- Udává do jaké skupiny patří modul.
- Např. načítací modul (`XmlLoader`), systémový modul (`DelayModule`), rendrovací modul (`Renderer`).
- Podle tohoto jména se zařadí do skupin v seznamu modulů *(výhledově – zatím se to zobrazuje podle namespace).*
- Výchozí hodnota je `unspecified`.

Např.:

```
[ModuleInfo("Autor modulu", "Module is used for cycles", IconName =
"Delay.ico", Assemblied = "2004-06-20", Category = "System")]
```

4.2.1. Přidání vlastní ikony

Pro přiřazení vlastní ikony modulu je nejdříve třeba tuto ikonu přidat do knihovny. Dále nastavit její vlastnost *Build Properties* na hodnotu *Embedded Resource* (pro zobrazení stačí v *Solution Exploreru* kliknout na přidanou ikonu pravým tlačítkem a zvolit *Properties*). Tím se zajistí, že se ikona přikompiluje ke knihovně. A nakonec definovat v `ModuleInfo` atribut `IconName` na jméno přidané ikony.

4.3 PortInfoAttribute

`PortInfoAttribute` obsahuje informace o jednom portu modulu. Před definicí smí být uvedeno více těchto atributů podle počtu portů.

Má 2 povinné parametry:
- *jméno* portu – v rámci modulu je jméno portu unikátní, musí být stejné jako jméno použité při přidání portu v kódu modulu
- funkce portu – stručný *popis* funkce portu zadaného jména

1 nepovinný parametr:

`IconName`
- Jméno ikony ve zkompilované knihovně, která reprezentuje port (přidání viz odstavec 4.2.1.).
- Pokud není zadána použije se výchozí ikona z jádra.

Např.:

```
[PortInfo("Input", "Výchozí hodnota")]
[PortInfo("Output", "Zpožděná data")]
[PortInfo("Initial", "Inicializační vstup", IconName="init.ico")]
```

5. Atributy datových struktur

5.1 Jak psát atributy

Platí stejné zásady jako u modulů (viz 4.1).

5.2 DataObjectAttribute

`DataObjectAttribute` obsahuje informace o MVE2 datové struktuře. Smí být použit pouze jeden před definicí třídy.

Má 1 povinný parametr:

- *popis* – stručný popis datové struktury

Např.:

```
[DataObject("Reprezentace vektoru ve 2D")]
```

6. Komentáře modulů a datových struktur

6.1. Jak psát komentáře

Komentáře by měli poskytovat podrobný a vyčerpávající popis částí zdrojového kódu modulu (to co se do atributů nevešlo) a datových struktur z hlediska uživatele. Jsou to standardní dokumentační komentáře (`///<summary>Komentář</summary>`).

Lze používat i `HTML` tagy. Nejčastěji používané budou:

`<p>`Odstavec`</p>`
`<br />` ... odřádkování
`<b>`**Tučný text**`</b>`
`<i>`*Kurzíva*`</i>`
`<a href="adresa">`Odkaz`</a>`
`<img src="relativní cesta k obrázku" />`

Pozor: Tagy by měli dodržovat formát *XML* ale zároveň i *HTML*. Proto jsou na konci tagů `<br />` a `<img />` mezera a lomítko. Navíc komentáře smí obsahovat znaky „`<`" a „`>`" jen jako ohraničení tagů. V ostatních případech je nutné je nahradit řetězci `&lt;` a `&gt;`. Komentáře, v nichž jsou tyto nepovolené znaky uvedeny, jsou *Visual Studiem* vyhodnoceny jako chybné a vynechají se.

Pro generování dokumentace ke knihovně programem `mmdoc.exe` jsou klíčové pouze některé z komentářů. Jsou to:

- `<summary>` u tříd
- `<summary>` u datových struktur public a protected metod, konstruktorů, eventů, property a členských proměnných
- `<param>` u datových struktur public a protected metod a konstruktorů
- `<returns>` u datových struktur public a protected metod

6.2. Komentáře modulů

- je třeba okomentovat třídu modulu

  - k čemu modul slouží
  - možnosti použití
  - omezení daná výpočetními možnostmi
  - omezení daná znalostmi z aplikační oblasti
  - s jakými moduly spolupracuje
  - apod.

- komentář u metody `InvokeSetup` (vyvolává okno pro nastavení modulu)
  - měl by popsat okno pro nastavení modulu z hlediska jeho uživatele
  - vhodné by bylo např. připojit obrázky pomocí tagu `<img />`

- komentáře u property s atributem `Description` případně s atributem `Browsable` nastavenym na `True`

6.3. Komentáře datových struktur

- je třeba okomentovat třídu datové struktury
  - jaká data reprezentuje
  - popis
  - je vhodný nějaký obrázek
  - omezení datových typů
  - omezení známá znalostmi v aplikační oblasti
  - jaké definice či pravidla splňuje (např. v geometrickém modelování přidat definici Eulerovy věta)
  - souvislost s jinými datovými strukturami
  - apod.

- komentáře jednotlivých public metod, property a členských proměnných
  - jsou určeny hlavně pokročilejším uživatelům – programátorům modulů
  - ti si zde vyhledají funkce a možnosti datových struktur, které používají ve svých modulech

7. Příklad

7.1. Příklad `AssemblyInfo.cs`

```
using System.Reflection;
using System.Runtime.CompilerServices;

// General Information about an assembly
[assembly: AssemblyTitle("Modular Virtual Environment 2 - Example")]
[assembly: AssemblyDescription("Example of MVE2 library")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("University of West Bohemia in Pilsen, Czech Republic")]
[assembly: AssemblyProduct("Examples")]
[assembly: AssemblyCopyright("Milan Frank 2004")]
[assembly: AssemblyTrademark("ZCU-MVE2.NET")]
[assembly: AssemblyCulture("")]

// Version information for an assembly consists of the following four
[assembly: AssemblyVersion("1.0.*")]

// In order to sign your assembly you must specify a key to use.
[assembly: AssemblyDelaySign(false)]
```

```
[assembly: AssemblyKeyFile("")]
[assembly: AssemblyKeyName("")]
```

Tučně jsou zvýrazněny informace používané `mmdoc`em a zobrazené ve výsledné dokumentaci.
Popis knihovny ve výsledné dokumentaci:

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 <0, 10> 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)

7.2. Příklad modulu

```
using System;
using Zcu.Mve.Core;

namespace Examples
{
  /// <summary>
  /// Source of one random scalar number.
  /// It's double-precision floating point number in the interval &lt;0.0, 1.0).
  /// Possibly you can define own output number.
  /// </summary>

  [ModuleInfo("Milan Frank", "Sample module. Source of one random scalar number.",
    IconName = "numbersource.ico", Assemblied = "2004-06-13", Category = "Example")]

  [PortInfo("Output", "Random or user defined scalar number.")]

  public class NumberSource : Zcu.Mve.Core.Module
  {
```

```
    /// <summary>
    /// Create ports.
    /// </summary>
    public NumberSource()
    {
        AddOutPort("Output", typeof (Examples.ScalarNumber));
    }

    ...
    //class body
    ...

    /// <summary>
    /// You can choose from two modes of function of module:
    /// <ol>
    /// <li><p>Generating of random number:</p>
    /// <p><img src="numbersource1.jpg" /></p></li>
    /// <li><p>Generating of defined number (2.0 in this case):</p>
    /// <p><img src="numbersource2.jpg" /></p></li>
    /// </ol>
    /// </summary>
    /// <returns>User control.</returns>
    public override ModuleSetup InvokeSetup()
    {
      //some code
    }

  } // NumberSource
} // namespace
```

Důležité části jsou zvýrazněny tučně.

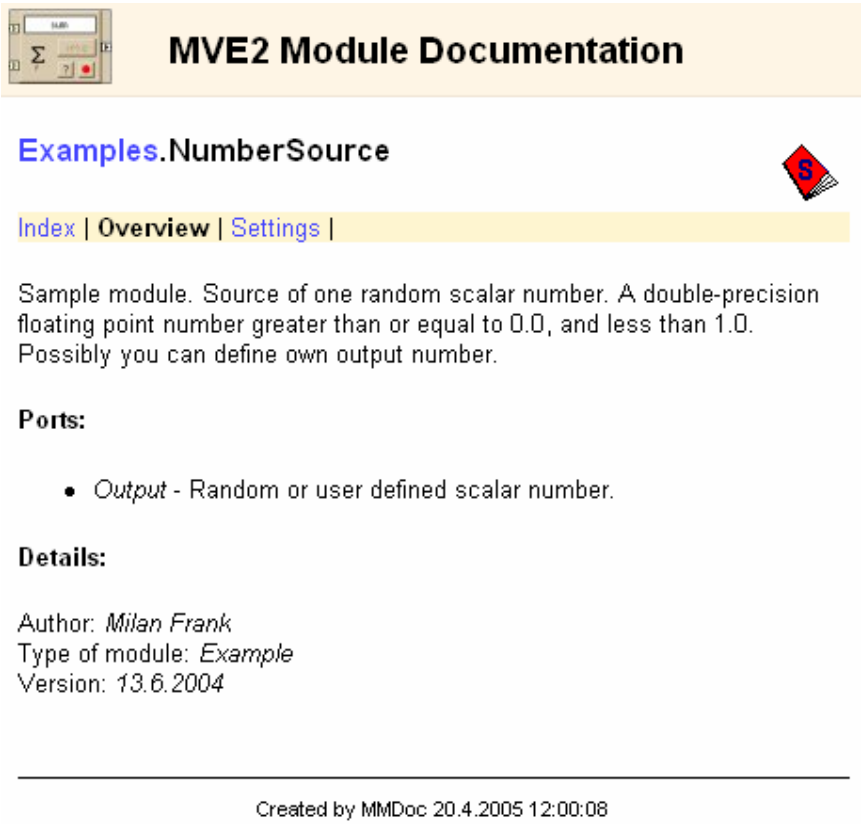
Na začátku je mezi tagy `<summary>` detailní popis modulu. Všimněte si jakým způsobem je zapsán interval <0.0, 1.0).

Pod ním je atribut `ModuleInfoAttribute`. Jsou v něm definovány 3 nepovinné parametry `IconName, Assemblied a Category`. Dále je atribut `PortInfoAttribute`. Za povšimnutí stojí, že jméno portu (první parametr) je stejné jako jméno portu zadané při přidávání v konstruktoru.

Poté je definice třídy modulu. Ta je odděděná od `Zcu.Mve.Core.Module`. Pak následuje kód, který definuje funkci modulu.

Nakonec je uvedena metoda `InvokeSetup`. Komentář u ní mezi tagy `<summary>` popisuje možnosti konfigurace modulu přes konfiguračníokno modulu. Je zde použito `html` a vkládání obrázků z aktuálního adresáře. Všimněte si ukončení tagu `<img src="numbersource2.jpg" />` mezerou a lomítkem.

Popis modulu ve výsledné dokumentaci:



MVE2 Module Documentation

Examples.NumberSource

Index | **Overview** | Settings |

Sample module. Source of one random scalar number. A double-precision floating point number greater than or equal to 0.0, and less than 1.0. Possibly you can define own output number.

Ports:

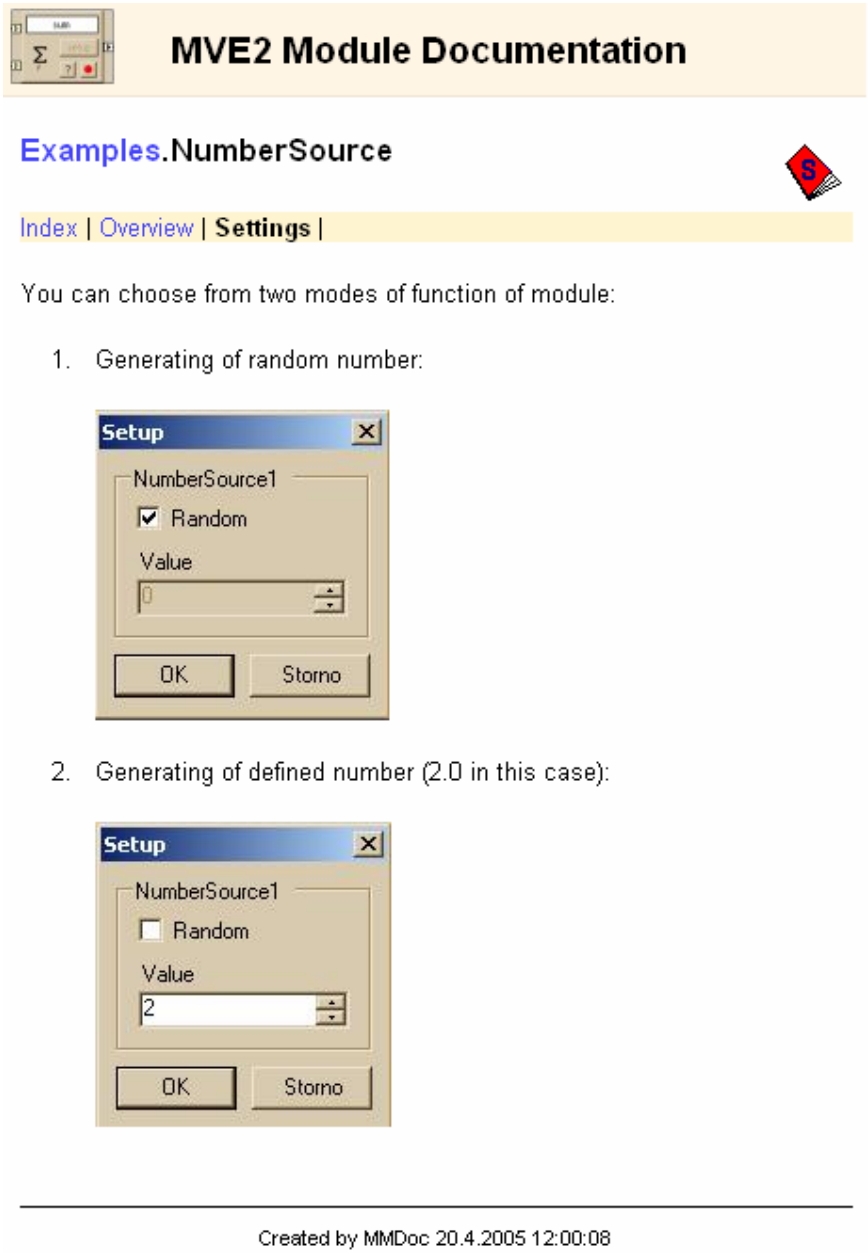
- *Output* - Random or user defined scalar number.

Details:

Author: *Milan Frank*
Type of module: *Example*
Version: *13.6.2004*

Created by MMDoc 20.4.2005 12:00:08

Nápověda ke konfiguračnímu oknu modulu (komentář k InvokeSetup):



MVE2 Module Documentation

Examples.NumberSource

Index | Overview | **Settings** |

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 20.4.2005 12:00:08

7.3. Příklad datové struktury

```
using System;
using Zcu.Mve.Core;

namespace Examples
{
  /// <summary>
  /// It represents one scalar number (accuracy as double type in .NET).
  /// Can be shared by MVE2 modules.
  /// </summary>
```

```
  [DataObject("One scalar number (double)")]

  public class ScalarNumber : DataObject
  {
    /// <summary>
    /// Value carried by this data object.
    /// </summary>
    private double val;

    /// <summary>
    /// Constructor with default settings.
    /// </summary>
    public ScalarNumber()
    {
      val = 0.0f;
    }

    ...
    //class body
    ...

    /// <summary>
    /// Gets/Sets scalar vaule of this object.
    /// </summary>
    public double Val
    {
      get
        {
          return val;
        }
      set
        {
          val = value;
        }
    }


    /// <summary>
    /// User defined deep copy of this data object.
    /// </summary>
    /// <returns></returns>
    public override IDataObject DeepCopy()
    {
      ScalarNumber ret = new ScalarNumber();
      ret.Val = this.Val;

      return ret;
    }

    /// <summary>
    /// Writes data object into XML file.
    /// </summary>
    /// <param name="xmlTextWriter">Xml stream.</param>
    public override void WriteData(System.Xml.XmlTextWriter xmlTextWriter)
    {
      xmlTextWriter.WriteString("" + this.Val);
    } // WriteData()

    /// <summary>
    /// Report about data structure. It's used for example by ConsolePrinter.
    /// </summary>
    /// <returns></returns>
    public override string ToString()
    {
      return this.val.ToString(Globals.nfi);
    }

  } // ScalarNumber
} // namespace
```

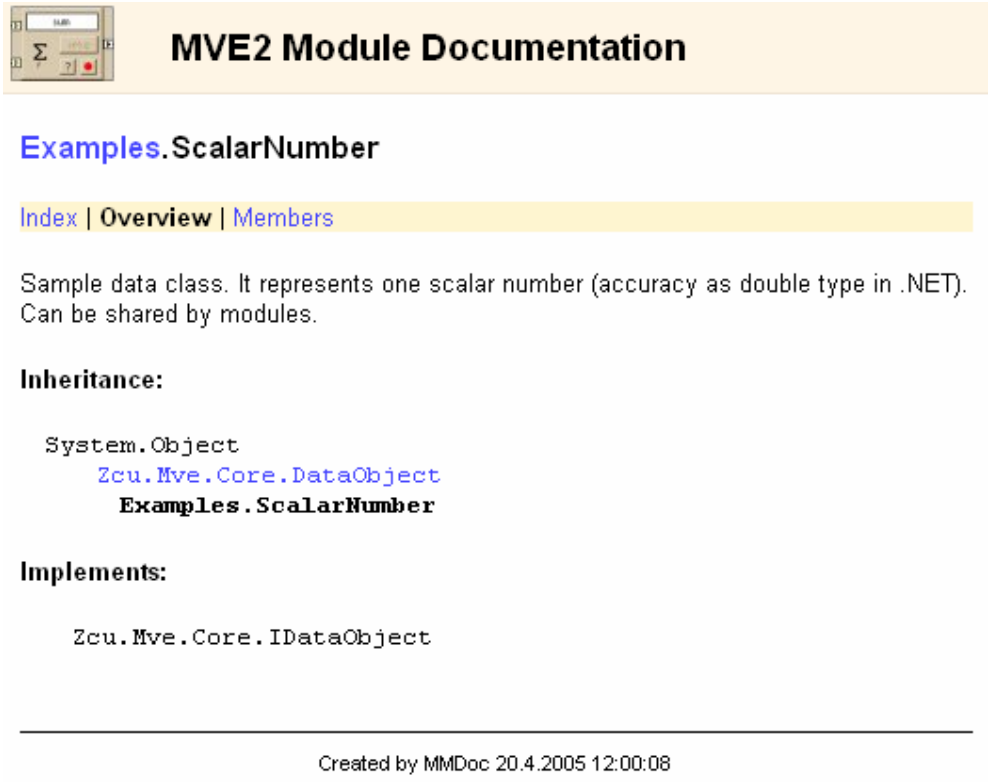
Důležité části jsou zvýrazněny tučně.

Na začátku je uveden podrobný popis *MVE2* datové struktury. Dále následuje o něco stručnější atribut `DataObjectAttribute`.

Pak je definice třídy. V tomto případě dědí od datového typu `DataObject`, který implementuje nutné rozhraní `IDataObject`.

Potom je definice členských proměnných, konstruktor, veřejné metody (`DeepCopy()`, `ToString()`) a property (`Val`).

Popis datové struktury ve vygenerované dokumentaci:



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

Komentáře konstruktorů, metod, property a členských proměnných datové struktury ve vygenerované dokumentaci:

MVE2 Module Documentation

Examples.ScalarNumber

Index | Overview | **Members**

Constructors

```
ScalarNumber ( )
```
Constructor wit default setting.

Public methods

```
virtual System.Void WriteData ( System.Xml.XmlTextWriter
xmlTextWriter ) ;
```
Writes data object into XML file.

- `xmlTextWriter`
  Datovy tok pro zapis dat do Xml souboru.

```
virtual System.String ToString ( ) ;
```
Report of the content of data structure. It's used by ConsolePrinter for example.

```
virtual System.Boolean CheckConsistence ( ) ;
```
User defined check of consistence of this data object. Warnings can be written to Console as well as reasons of inconsistency.

Returns:
True if it's consistent, false otherwise.

```
virtual System.Void ReadData ( System.Xml.XmlTextReader
xmlTextReader ) ;
```
Reads data from XML file into data object.

- `xmlTextReader`
  Datovy tok pro cteni dat z Xml souboru.

```
virtual Zcu.Mve.Core.IDataObject DeepCopy ( ) ;
```
User defined deep copy of this data object.

Public properties

```
System.Double Val { get;set; }
```
Gets/Sets scalar value of this object.

Created by MMDoc 12/12/2005 9:35:05 PM