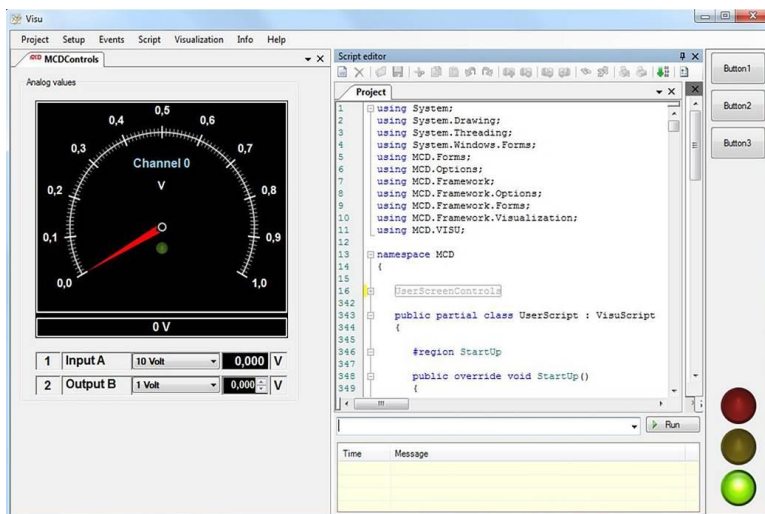


# Manual

## Toolmonitor Visu



GET IN **touch**  
WITH SENSITIVE TESTING

Softline

Modline

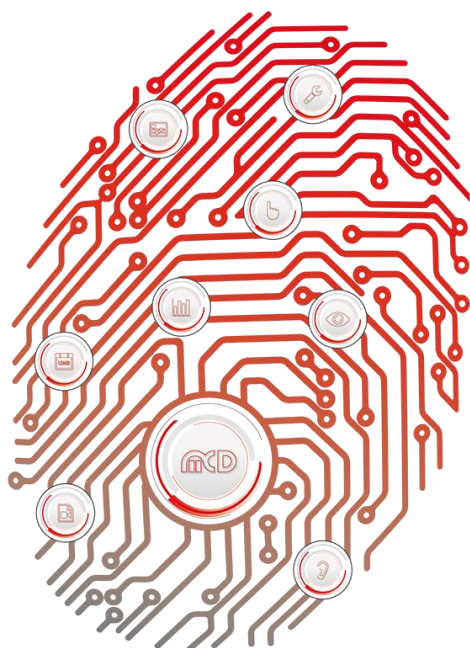
Conline

Boardline

Avidline

Pixline

Application



### MCD Elektronik GmbH

Hoheneichstr. 52

75217 Birkenfeld

Tel. +49 (0) 72 31/78 405-0

Fax +49 (0) 72 31/78 405-10

info@mcd-elektronik.de

www.mcd-elektronik.com

HQ: Birkenfeld

Managing CEO: Bruno Hörter

Register Court Mannheim

HRB 505692

## Table of Contents

<b>1. GENERAL</b>	<b>4</b>
<b>2. SOFTWARE AND DRIVER INSTALLATION</b>	<b>4</b>
2.1. REQUIREMENTS	4
2.2. STARTING THE INSTALLED SOFTWARE	4
<b>3. INTRODUCTION INTO OPERATION</b>	<b>5</b>
3.1. BRIEF INTRODUCTION	5
<b>4. SOFTWARE MANUAL</b>	<b>7</b>
4.1. DESIGNER	7
4.1.1. <i>Opening Designer</i>	7
4.1.2. <i>Creating a New Form</i>	7
4.1.3. <i>Settings and Parameters</i>	8
4.1.4. <i>Adjusting the Menu Structure</i>	9
4.1.5. <i>Properties and Events</i>	10
4.1.6. <i>Controls</i>	11
4.2. SCRIPT	13
4.2.1. <i>Script Editor</i>	13
4.2.1.1. <i>Base Script</i>	14
4.2.1.2. <i>Explorer</i>	15
4.2.1.3. <i>Context Menu</i>	16
4.2.1.4. <i>Status Display</i>	18
4.2.1.5. <i>Text Editor</i>	19
4.2.2. <i>Run Command</i>	23
4.2.3. <i>Stop Command</i>	23
4.2.4. <i>Integrating DLLs</i>	24
4.2.5. <i>Global Variables</i>	25
4.2.6. <i>Virtual Interface</i>	25
4.3. REPORT	29
<b>5. EVENTS</b>	<b>34</b>
5.1. LOGGING	34
5.2. TRACE	34
<b>6. DEBUGGER</b>	<b>35</b>

---

6.1. DEBUGGER INSTALLATION .....	35
6.2. STARTING THE DEBUGGER .....	36
6.3. DEBUGGING THE SCRIPT.....	39
6.4. USING BREAKPOINTS .....	39
6.5. MONITORING VARIABLES .....	40

## 1. General

The “Visualization Toolmonitor“, also referred to “Toolmonitor Visu“, offers an easy and fast way to generate operating and user interfaces in connection with the MCD TestManager. No external software is required. The interfaces created using the *Designer* can be linked to own procedures and functions via the script editor.

**Order number:** # 122427

## 2. Software and Driver Installation

### 2.1. Requirements

- Windows (Windows XP® – Windows 8.1®, 32 or 64 bit)
- .Net – Framework 3.0

Copy the *VisuMonitor.exe* into a user - defined directory on the target system to install the MCD Toolmonitor Visu.

### 2.2. Starting the Installed Software

The software can be started by executing *VisuMonitor.exe*.

### 3. Introduction into Operation

#### 3.1. Brief Introduction

On first starting of the Toolmonitor Visu, the still empty interface of the monitor appears.

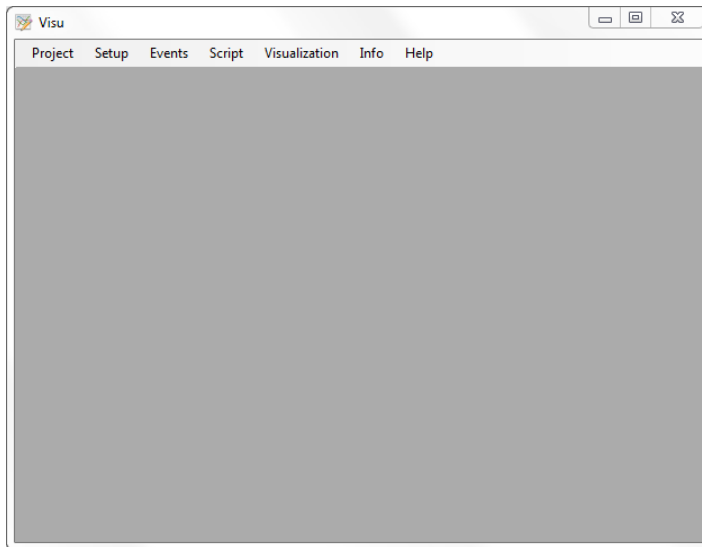


Figure 1: Default Interface

To obtain an overview of the functions and possibilities of Toolmonitor Visu, a sample preset can be loaded via **Project → Presets → Sample**, which offers an overview of all monitor functions.

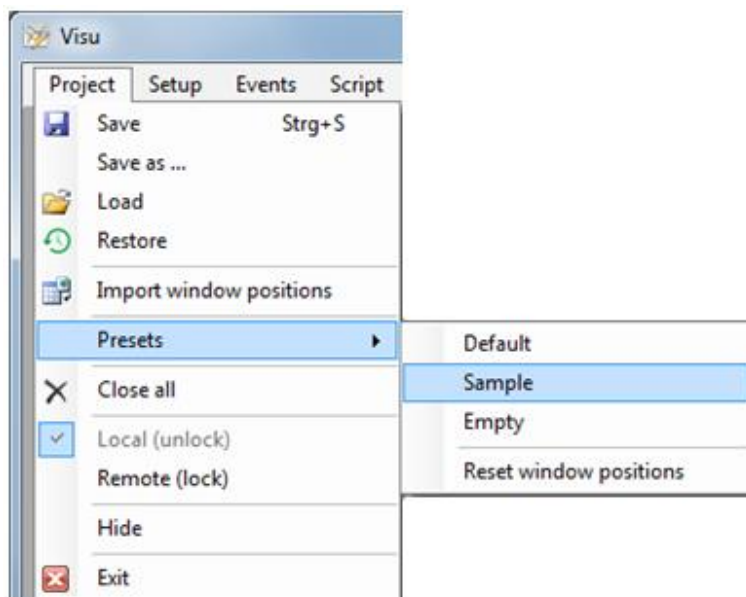


Figure 2: Loading a Sample Preset

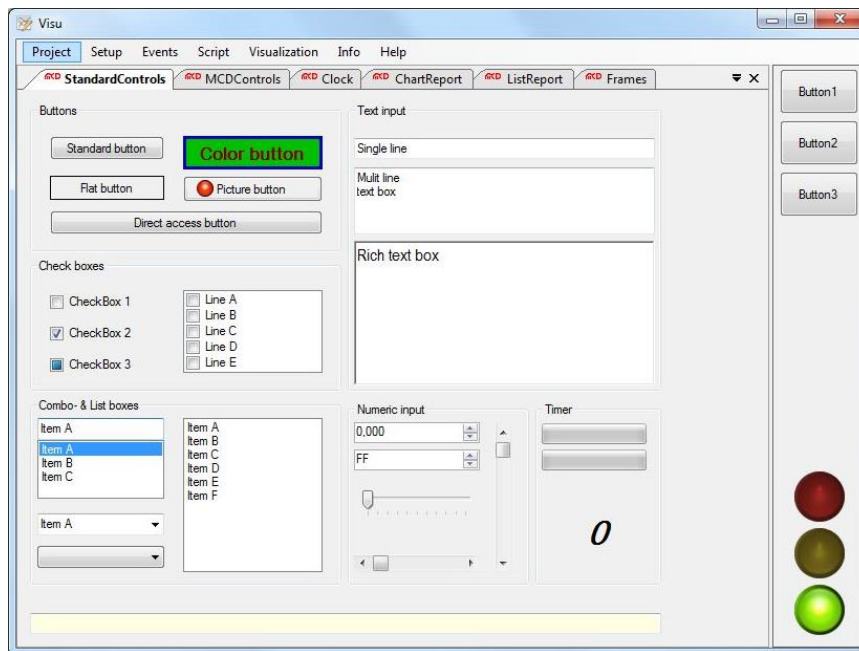


Figure 3: Sample Preset Interface

The **Script Editor** for the generation of own functions and procedures can be reached via the **Script → Script Editor** menu item.

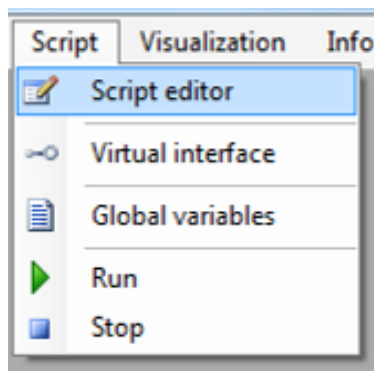


Figure 4: Open Script Editor

## 4. Software Manual

### 4.1. Designer

Custom operating and user interfaces can be created in the **Designer** of Toolmonitor Visu. Standard components and additional MCD controls are available for this purpose.

#### 4.1.1. Opening Designer

To open the *Designer* of the previously created project, the desired project (here "Sample") is selected under the **Visualization → Designer** menu item.

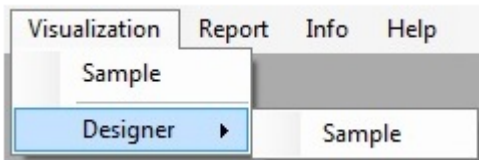


Figure 5: Opening Designer

#### 4.1.2. Creating a New Form

Operate the **New** button in the **Form** menu to create a new form in the *Designer*. It is also possible to import an already existing interface or to export the newly created interface using this menu. The button **Clear** deletes the current interface.

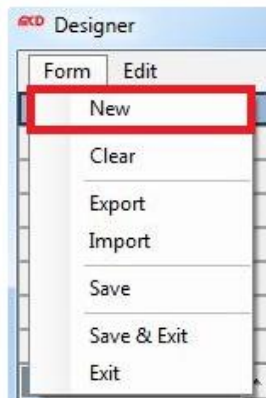


Figure 6: Creating a New Form

After a new form is created, a name can be assigned. This **Form Title** is inquired automatically and required for creating the new form.



Figure 7: Naming the form

### 4.1.3. Settings and Parameters

A list with the settings and parameters of the selected object can be found on the right side in the *Designer*. These configurable parameters are also referred to as **Properties** and are different for every form and control. The exact size and position can be defined for most objects. In the case of analog displays, the minimum and maximum value and the display scaling can be configured as well.

To change the *Properties* of a control / form, it must be marked in the design window (center window).

In the example below, Button1 (left red frame) was selected. Thus, the color (upper red frame) or the text of the button (lower red frame) can be changed in the right list.

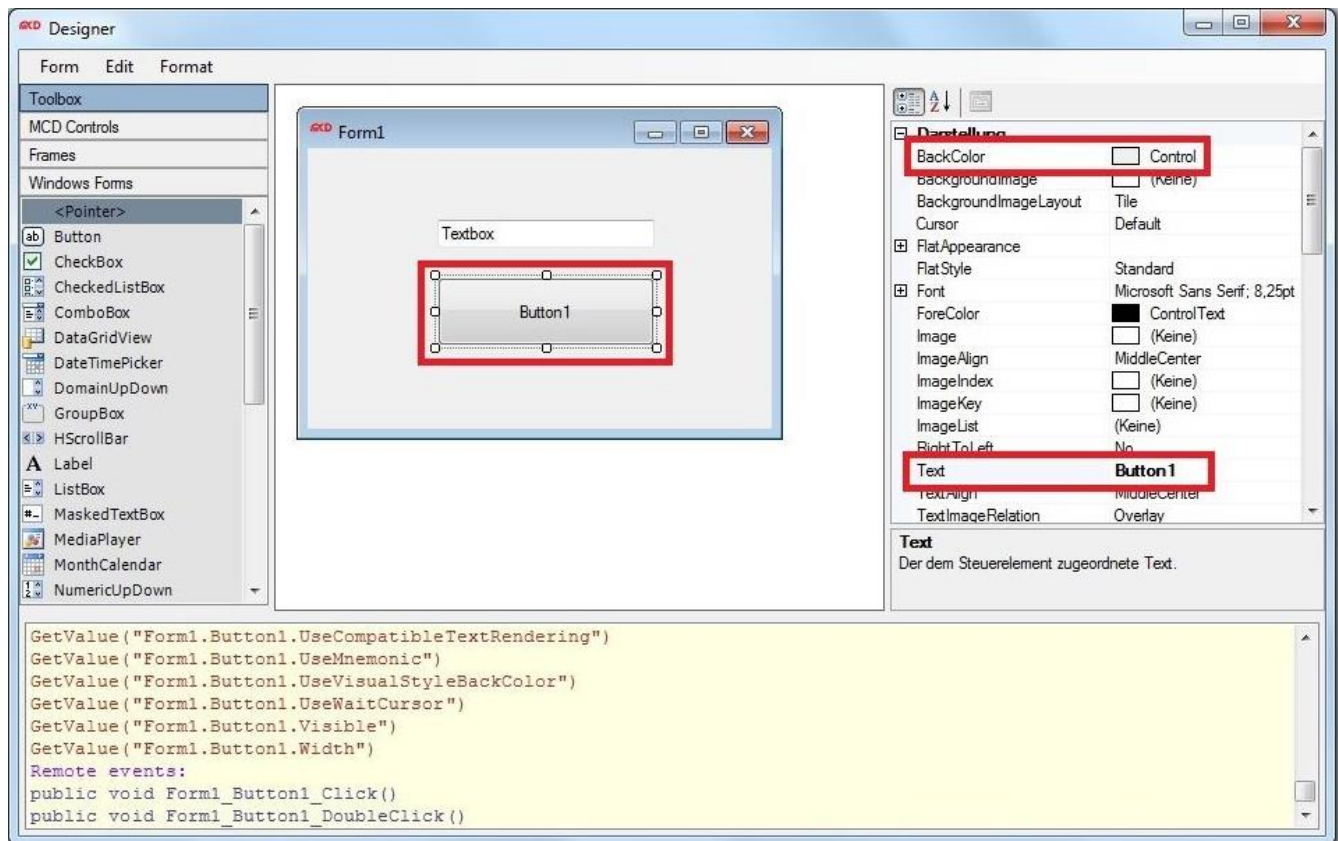


Figure 8: Properties of a Button



#### 4.1.4. Adjusting the Menu Structure

The structure of the **Menu Tree** can also be adjusted about the properties of the form. After the desired form was selected in the *Designer*, the *Menu Tree* property is available on the right side under *Properties*. Here it can be defined, under which sub - group the interface should be displayed in the Toolmonitor.

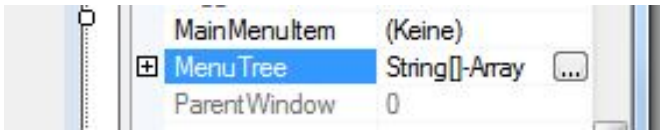


Figure 9: Menu Tree

In this example, the form is called "Test". It contains an analog voltage display. The structure of sub menu, in which the "Test" interface is displayed, can be seen in Figure 10. "Visualization" is at first position, and thus the menu item to be displayed in the top bar of the Toolmonitor. "Voltage" is a sub - item in the "Visualization" menu. The next point would then be a sub - item of "Voltage".



Figure 10: Structure of the Sub Menu

Any number of sub items can be created according to this scheme. However, this does not always ensure clarity. In our example, the “Test interface” can be seen under “Voltage”, as can be seen in Figure 11.

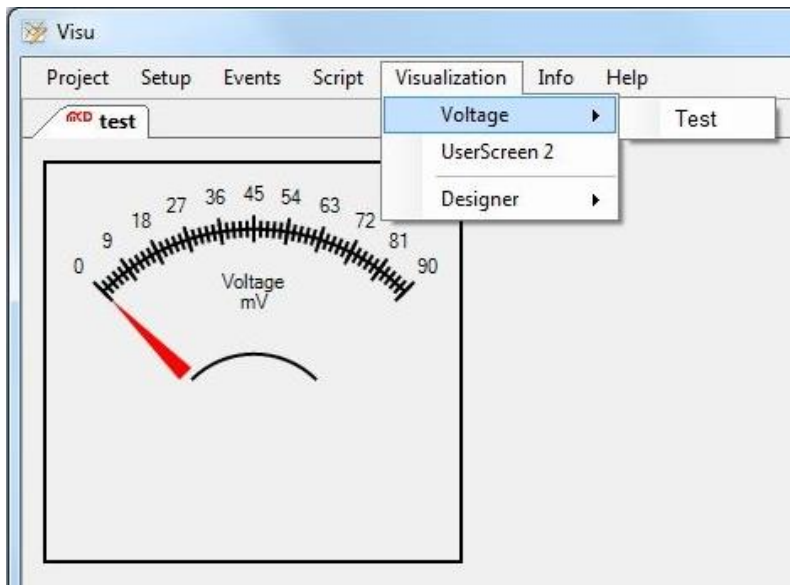


Figure 11: Sub Menus in the Toolmonitor

#### 4.1.5. Properties and Events

A list of the changeable *Properties* and **Events** of a control can be found in the lower *Designer* window. For example, the **Remote Events** of a button can be found on the first figure. The *Events* and changeable *Properties* are different for every control. The respective *Remote Events* are listed under **Controls**. *Properties* of a control can be read out using the **GetValue** function in the **Script Editor**. They can be changed using the **SetValue** function.

```
GetValue("Test.Form.Visible");  
SetValue("Test.Form.Size", "295; 305");
```

Figure 12: GetValue and SetValue Functions

Reactions to changes or interactions with a control or form can be designed using *Events*.

```
public void Test_Button1_Click(object sender, EventArgs e)  
{  
    ...  
}
```

Figure 13: Displaying Event

A double - click on the desired *Event* switches from *Designer* to **Script Editor**, where the reactions to the respective *Event* can be programmed.

```
GetValue ("example.Button1.UseCompatibleTextRendering")
GetValue ("example.Button1.UseMnemonic")
GetValue ("example.Button1.UseVisualStyleBackColor")
GetValue ("example.Button1.UseWaitCursor")
GetValue ("example.Button1.Visible")
GetValue ("example.Button1.Width")
Remote events:
public void example_Button1_Click()
public void example_Button1_DoubleClick()
public void example_Button1_KeyDown()
public void example_Button1_SizeChanged()
public void example_Button1_TextChanged()
```

Figure 14: Properties and Events

#### 4.1.6. Controls

The list of usable **Controls** can be found on the left side of *Designer*.

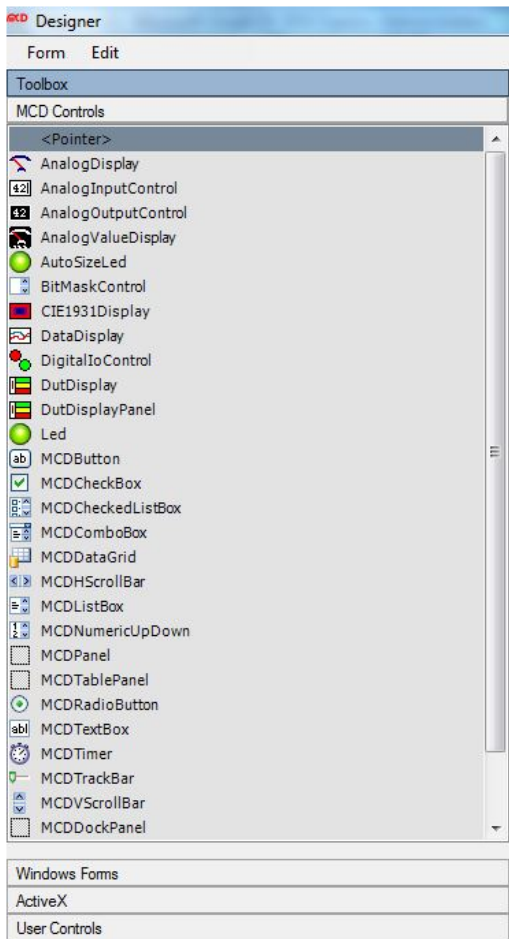


Figure 15: List of available Controls

In addition to the standard **Windows Controls**, additional MCD related controls are available under the **MCD Controls** tab. These *MCD Controls* contain controls, such as the **CurveAnalysis Control**, which are not included in the Toolmonitor by default, but can be activated via license. The difference between a standard *Windows Control* and a *MCD Control* can be seen in the following example.

For example, if a form with a MCD button is controlled via the TestManager, hence **Remote**, the button is automatically locked on the interface and cannot be manually operated. However, a Windows button cannot be locked and can be further manually operated (figure on the right). Thus, the desired type can be selected for every application case. To test this behavior in the Toolmonitor, the view can be switched between **Local and Remote** via the Project → **Local / Remote** menu item (figure on the left).

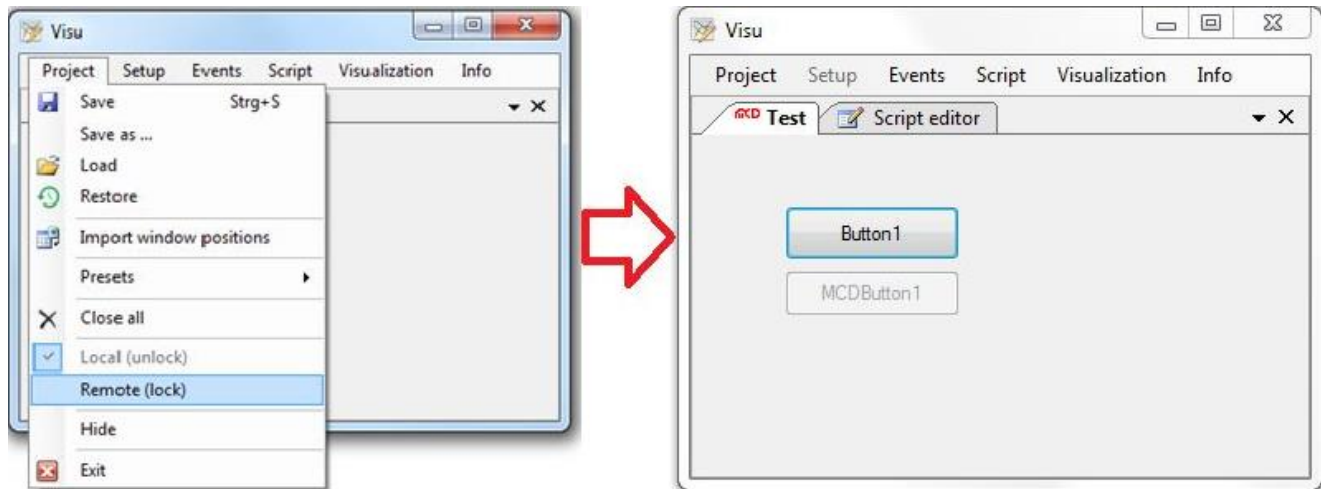


Figure 16: Windows Controls and MCD Controls

## 4.2. Script

### 4.2.1. Script Editor

It is possible to create custom procedures and functions in the **Script Editor** to expand the Toolmonitor. Programming uses the C# programming language of the .Net framework. Available .Net assemblies can be used in the scripts.

The *Script Editor* mainly exists of three areas. The **Text Editor**, the **Explorer** and the **Status and Debug Display**. Most settings and additional functions can be reached via the context menu of the *Script Editor* and are described in the following.

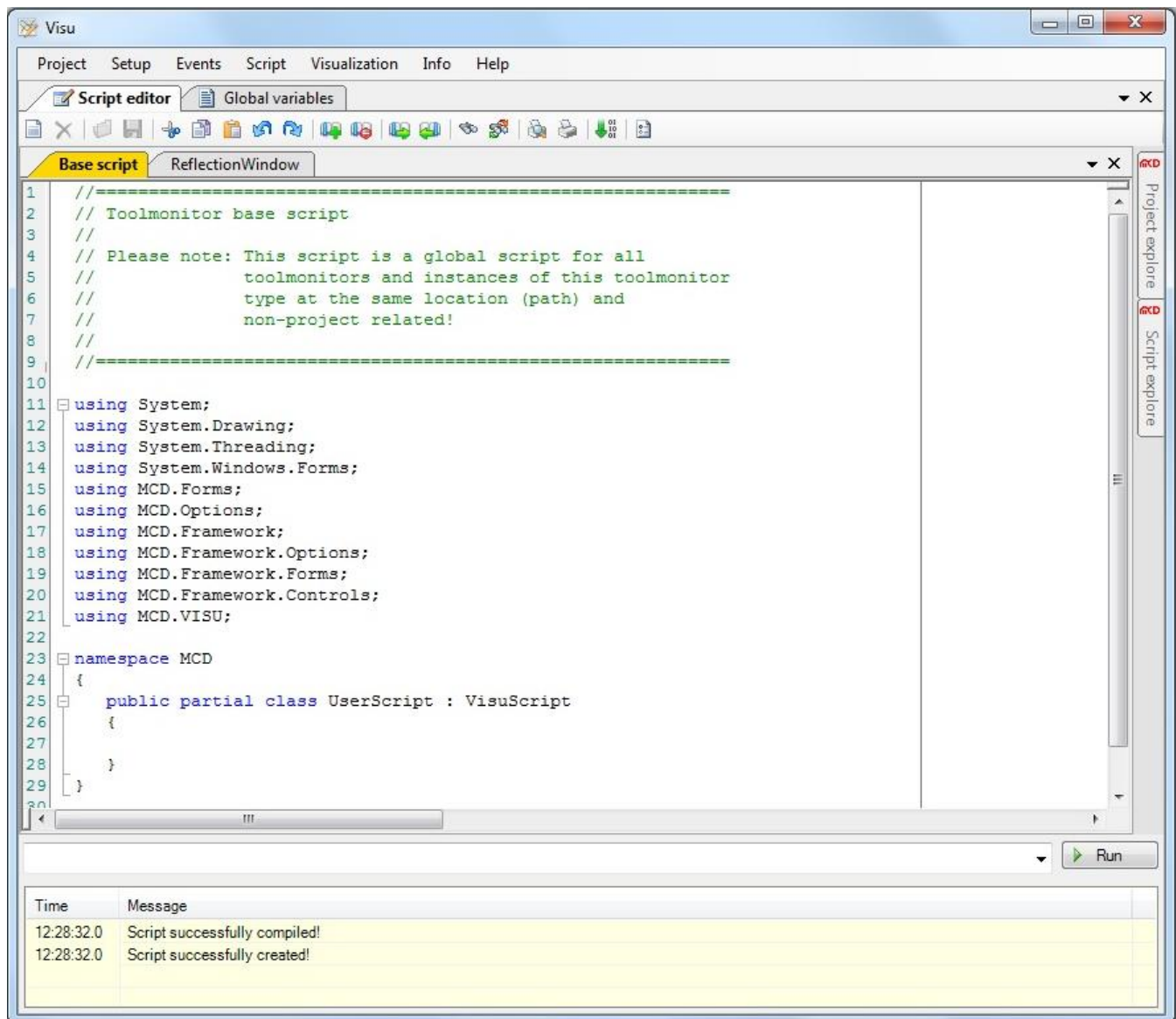


Figure 17: Script Editor Interface

#### 4.2.1.1. Base Script

The so - called **Base Script** is a global script for all Toolmonitors and their instances. The *Base Script* is the standard script that opens on first opening of *Script Editor*. Any other created scripts and / or tabs can access the *Base Script*.

In the lower example, a “Testfunktion()” function was created in the *Base Script*, which is used to call a “Testfunktiontab()” function in the “tab1” script. In the “tab1” script, it is possible to access the variables (here “Testwert”) of the “Base Script” as can be seen in the example below.

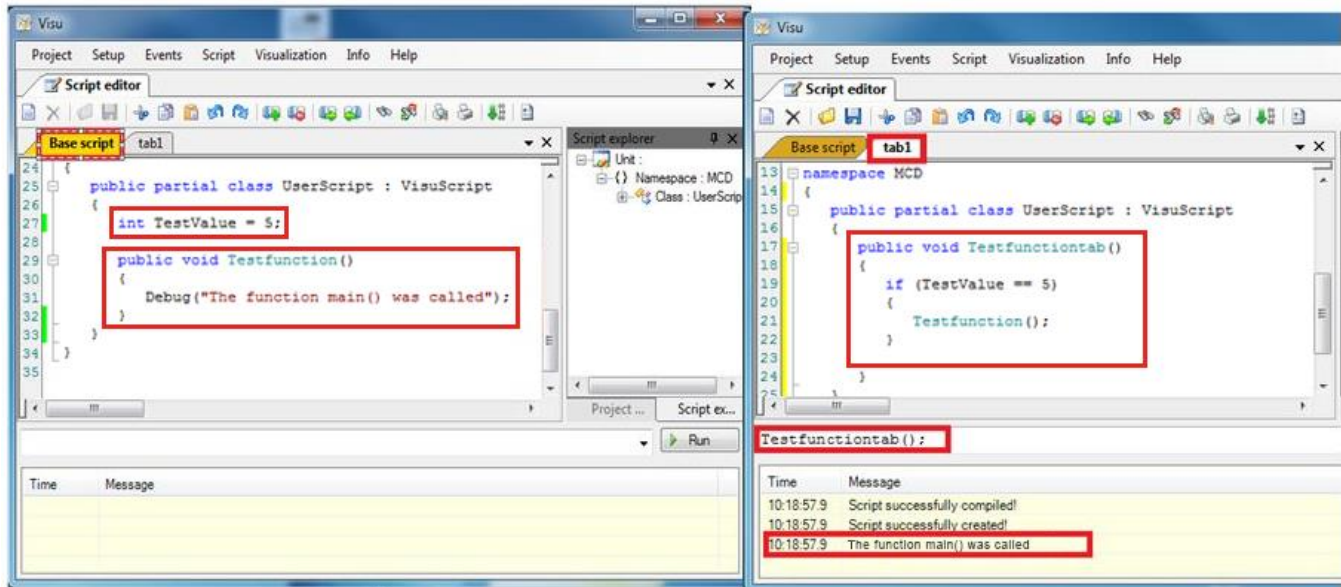


Figure 18: Function Call in the Base Script

The *Base Script* of a Toolmonitor is the only script that is saved as separate C# file (.cs). All other scripts, settings, and contents of the project are saved as .xml in the project file.

Thus, if a project with several scripts is moved to a different PC, the project file and the corresponding *Base Script* must be moved. The advantage of this structure is that several scripts and several different Toolmonitors can access the content / functions of the *Base Script* at the same time.

#### 4.2.1.2. Explorer

The **Script Explorer** and / or **Project Explorer** can be found on the right side of the Toolmonitor interface. Here, the inserted **Usings**, as well as the existing variables and functions are displayed hierarchically (**Script Explorer**) and / or the different scripts are listed (**Project Explorer**). When double - clicking on an element, the cursor in the text editor jumps automatically to the respective position in the source text.

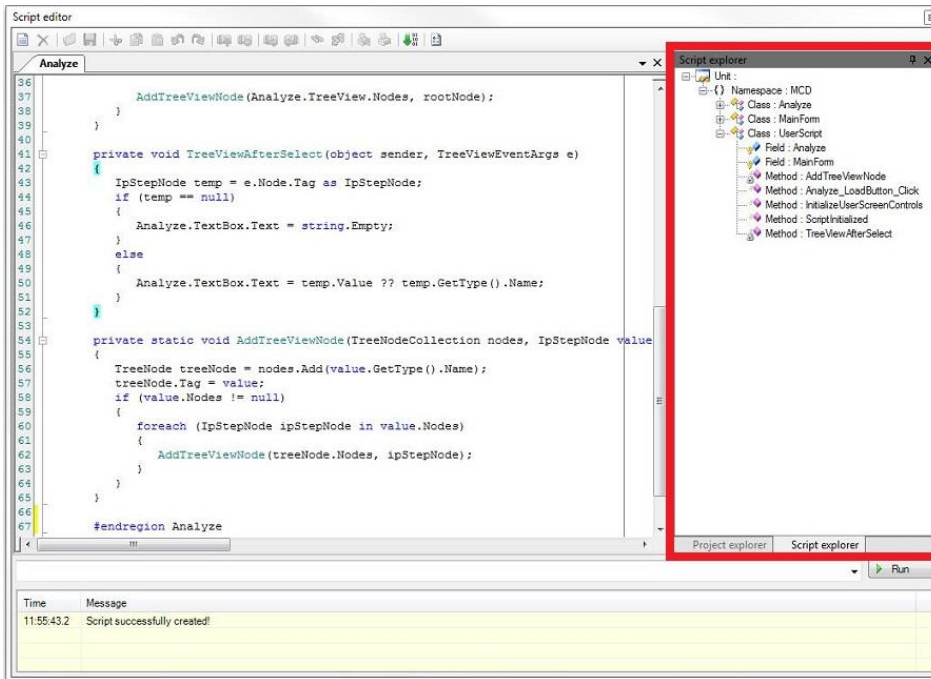


Figure 19: Script Explorer

The so - called **Script Explorer** displays all namespaces with their respective classes, all functions, and all created variables hierarchically. When double - clicking on one of the displayed elements, the cursor in the text editor jumps automatically to the respective position in the source code.

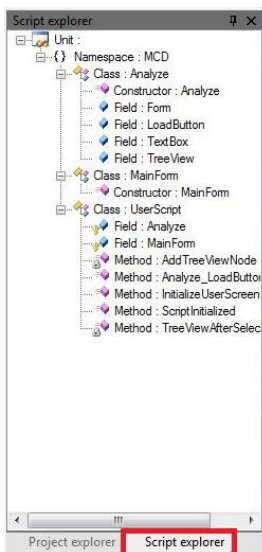


Figure 20: Displaying Script Explorer



The so - called **Project Explorer** contains an overview of all available scripts. Double - clicking on a script opens the respective script automatically.

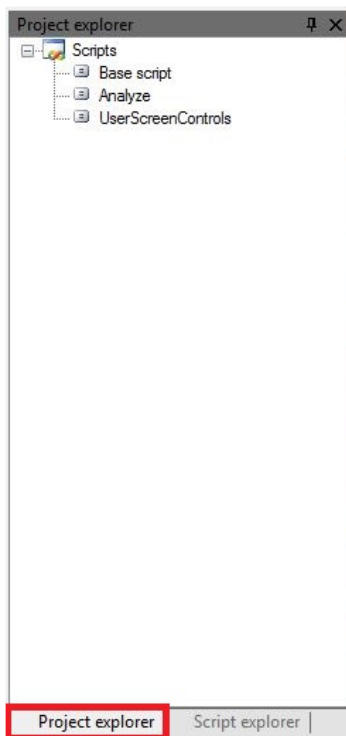


Figure 21: Displaying the Project Explorer

#### 4.2.1.3. Context Menu

The different options and settings of the *Script Editor* can be reached via the **Context Menu**. For this purpose, the right mouse button is clicked within the editor. After clicking, the *Context Menu*, shown below, opens. An overview of the individual functions of the *Context Menu items* can be found in the table below. Some of these functions can also be reached via the icons in the top menu bar of the text editor.

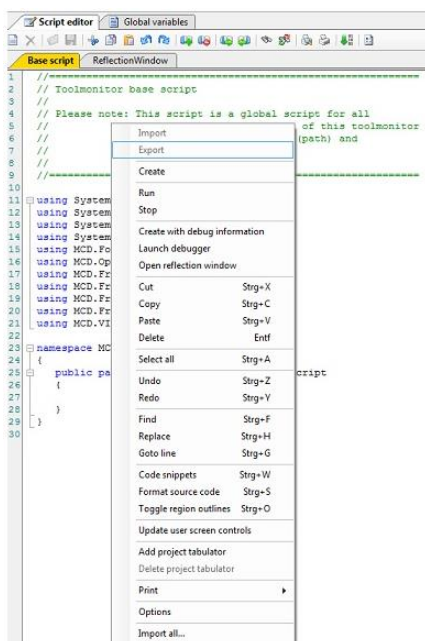


Figure 22: Context Menu of the Script Editor



The following table shows the Context Menu items, the corresponding functions, and, if available, the corresponding icons in the toolbar.












	Context menu	Description
	Import	Imports an existing script
	Export	The created script can be stored externally
	Create	Creates the script
	Run	Runs the script
	Stop	Stops a running script
	Create with debug information	The script is created with debug information
	Launch Debugger	Launches the debugger
	Open reflection window	The overview window for the project opens. Right click for refresh
	Cut	The selected text/code is cut out
	Copy	The selected text/code is copied to the clipboard
	Paste	Text/code is inserted from the clipboard
	Delete	The selected text/code is deleted
	Select all	The entire content of the script is selected
	Undo	Undo the last step
	Redo	Repeat the last undone action
	Find	Opens the search window
	Replace	Open the search and replace window
	Goto line	Jumps to the entered line
	Code snippet	Here you will find often needed, ready-made pieces of code
	Format source code	The code is automatically formatted
	Toggle region outlines	#regions are collapsed / expanded
	Update user screen controls	Updates the created user interfaces
	Add project tabulator	Adds a new tab (tab) to the project
	Delete project tabulator	The currently opened tab will be deleted
	Print	Prints the script
	Options	Opens the syntax setting of the Script Editor
	Import all...	Imports all scripts from an MCD Toolmonitor XML file

Figure 23: Function Overview of the Context Menu

#### 4.2.1.4. Status Display

The third area with the **Status and Debug Display** is located at the bottom in the *Script Editor* window.

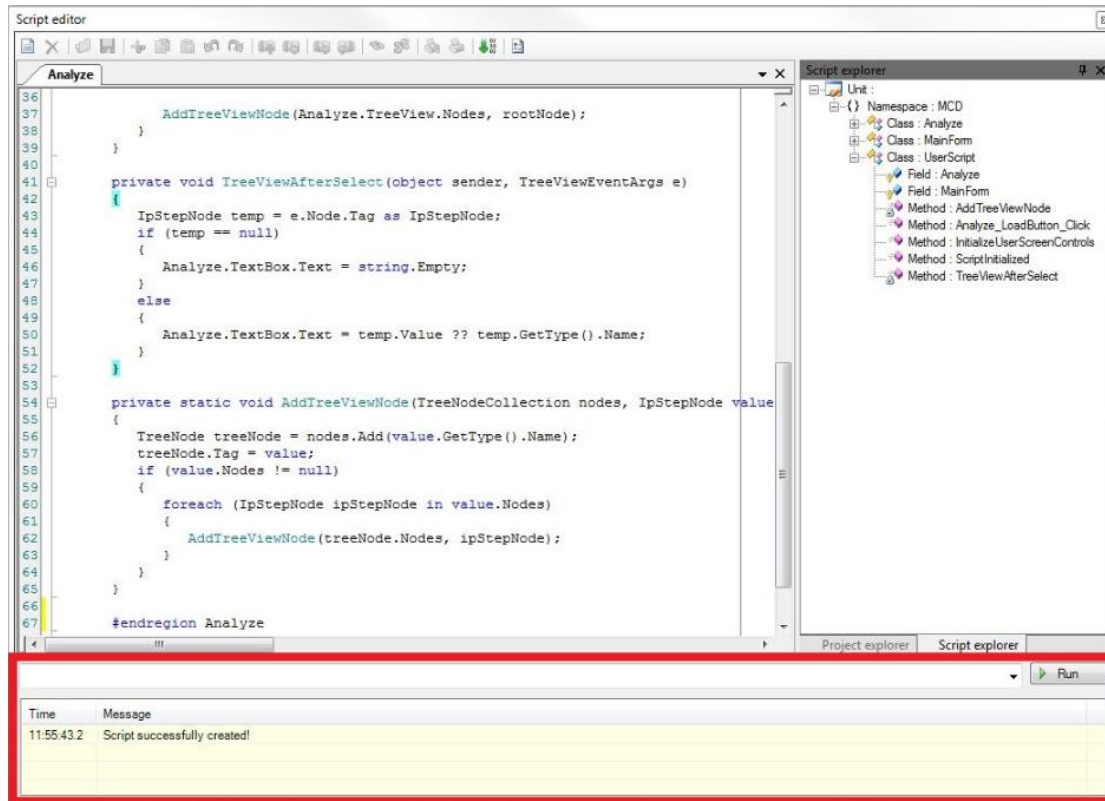


Figure 24: Status Display

This area allows debugging individual functions and commands via the *Debug line* and button. Furthermore, the bottom window also displays status messages and *Debug outputs*.

To call a desired function, the name of the function is entered into the *Debug line* and debugging is started with *Debug button* on the right. A *Debug command* is available to display variable values for test and verification purposes. In the example below, the **"Main() method"** is called and the values of variables Testvalue 0 to Testvalue 5 are outputted in the *Debug window*.

The entered *Debug commands* are saved in the **Registry** and can be selected again for another call.

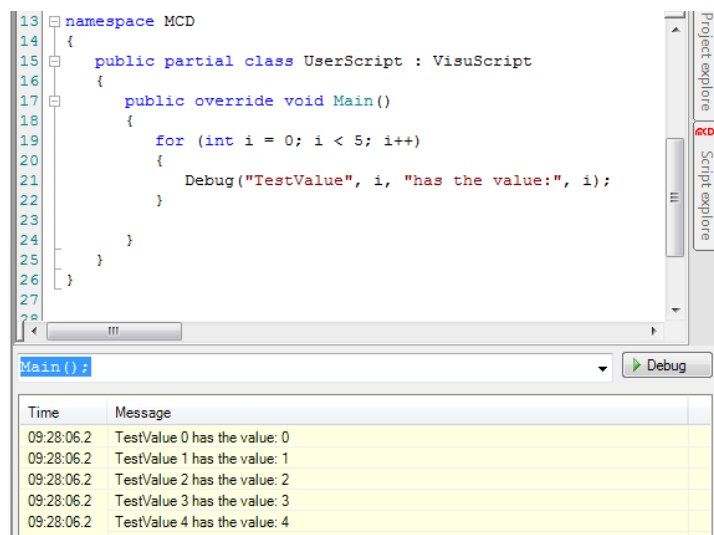


Figure 25: Debug Output of the Main() Method

### 4.2.1.5. Text Editor

The actual **Text Editor** for source text creation can be found in the left window of the *Script Editor*.

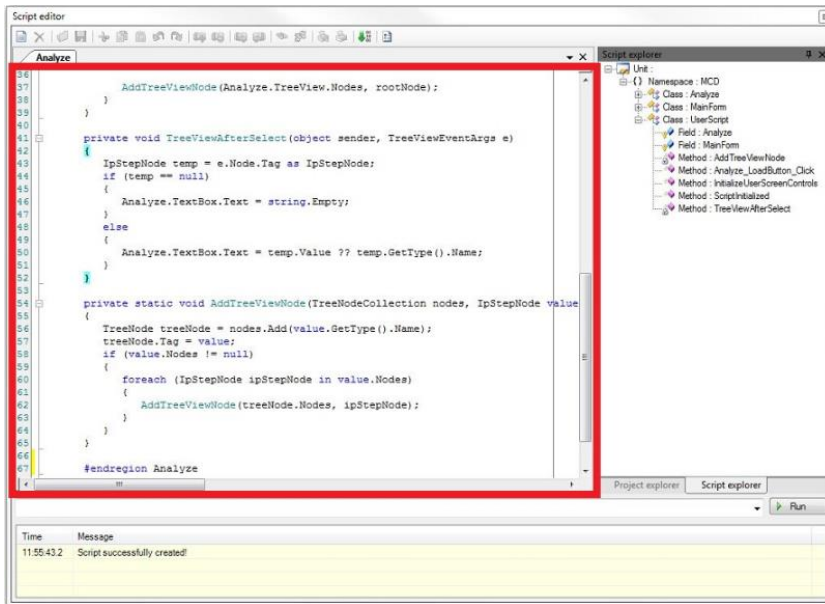


Figure 26: Text Editor Interface

If you start with an empty script and do not want to enter the required *Usings* manually, any character can be written into the empty text field. After clicking on the **Create** command of the *Context Menu*, the required usings are inserted automatically.

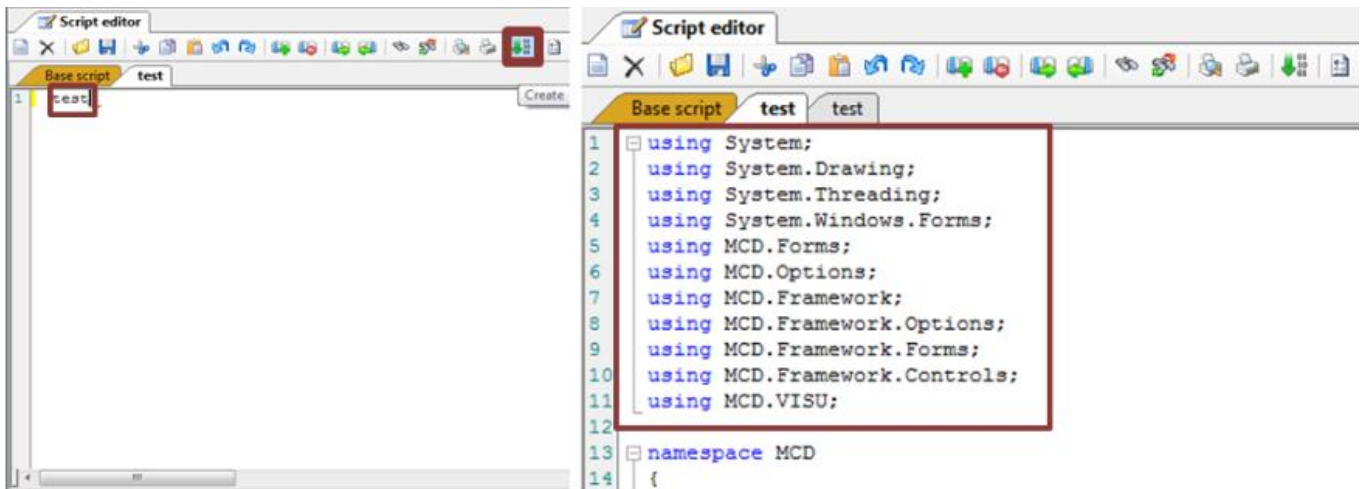


Figure 27: Entering Usings into the Empty Script

The *Text Editor* offers different possibilities for automatic code completion (*Intelli - Sense*). This way, properties and functions can be displayed for existing variables. If you enter a period after a variable, the available functions and properties are displayed automatically. In the figure below, the properties of a *double variable* are called to convert it into a *string* (the `ToString()` method is used for this purpose).

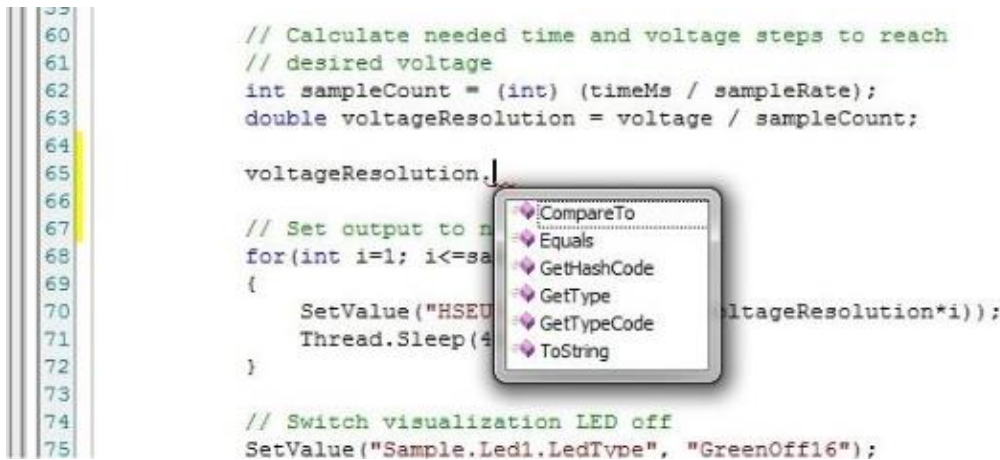


Figure 28: Code Completion (variable. / function.)

Furthermore, **operator this.** and a *Context Menu* can be used to call the available functions. The example shows the call of the `SetEvent()` function. For this purpose, simply enter text "this.". After the period is entered, a window opens automatically, displaying the functions and their parameters. Select the desired function via double - click or enter.

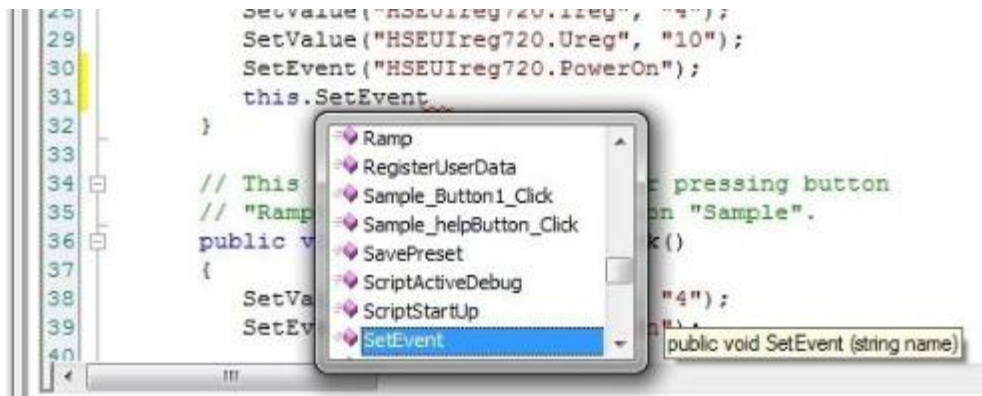


Figure 29: Code Completion (.this)

If you have entered the correct function name, the possible parameters are displayed when you enter the open parenthesis. If the used function is overloaded, you can select using the arrows to display the different parameters.

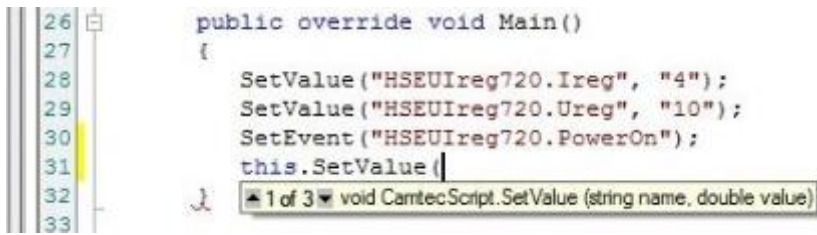


Figure 30: Displaying Parameters

Detailed help regarding programming - specific questions is available via the integrated Microsoft online offering (MSDN). Keyboard shortcut "Ctrl + F1" opens "MSDN" automatically in your internet browser. You can mark a keyword and / or place the cursor into the word to call up the "MSDN help" for this keyword via "Ctrl + F1". In the figure, the document for keyword "double" has been called up.

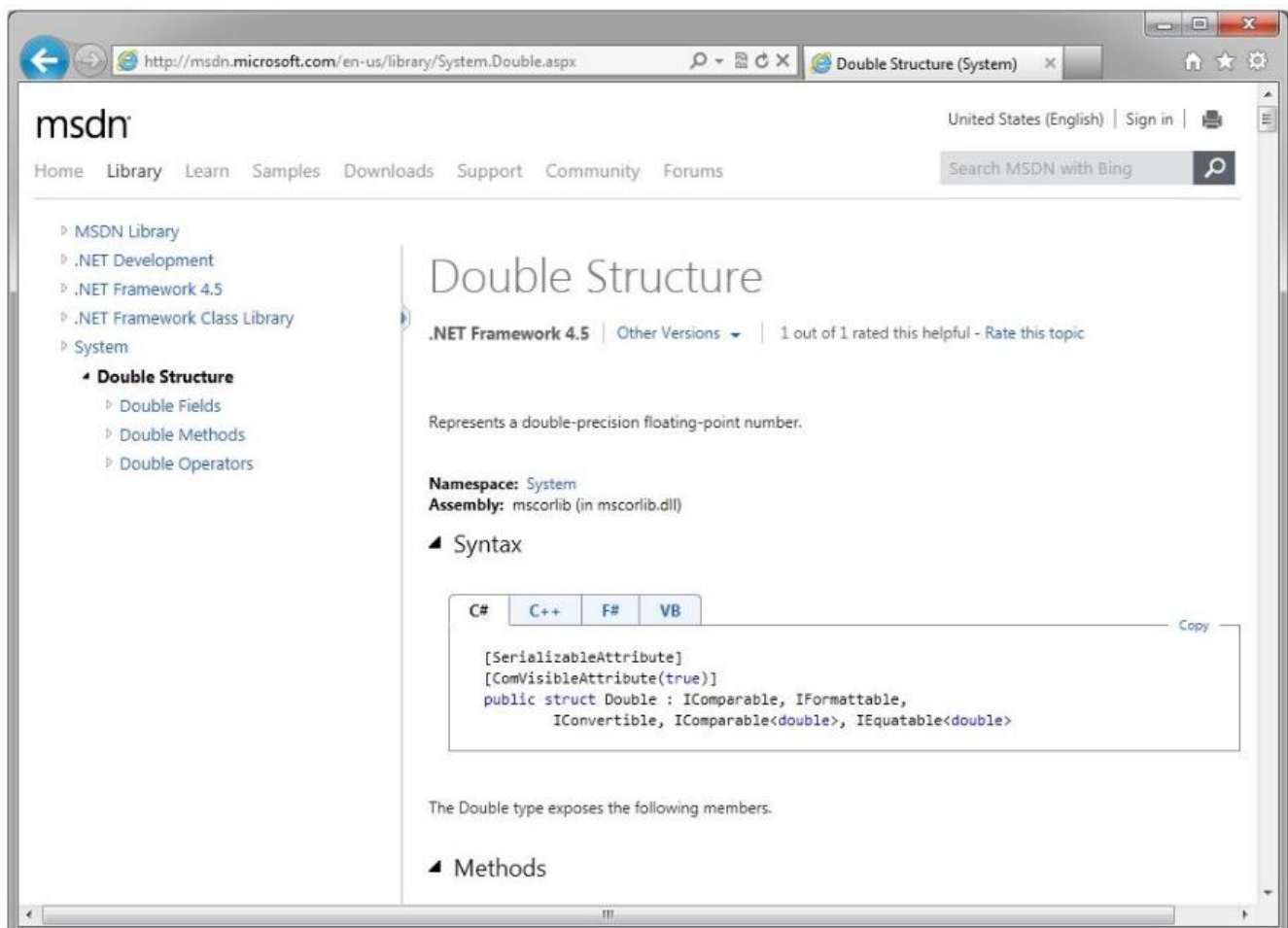


Figure 31: MSDN Online Help (Ctrl + F1)



The source text presentation can be adjusted via the item **Options** in the *Context Menu*. For example, the colors for certain keywords, strings or numbers can be defined here. Furthermore, the number of spaces for a tab or the division of the editor window can be adjusted as well. The figure shows the main page of the *Options*.

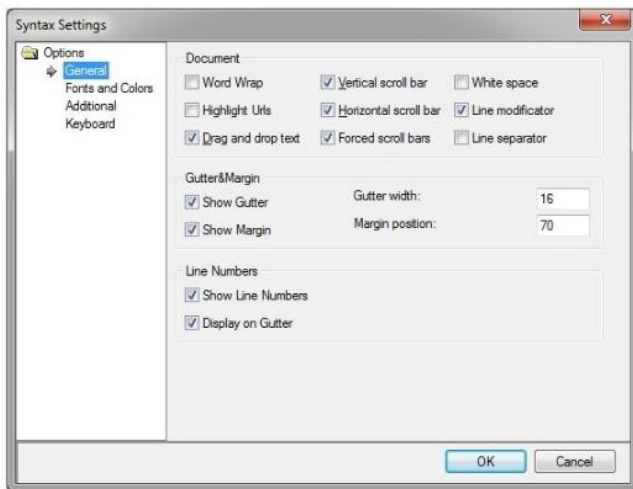


Figure 32: Script Editor Options (Context Menu)

If you would like to display and edit different areas of your script in parallel, a second window can be opened. This window can be opened via the rectangular top right button (via the scrollbar). Move the mouse over the button and pull open another window from the top.

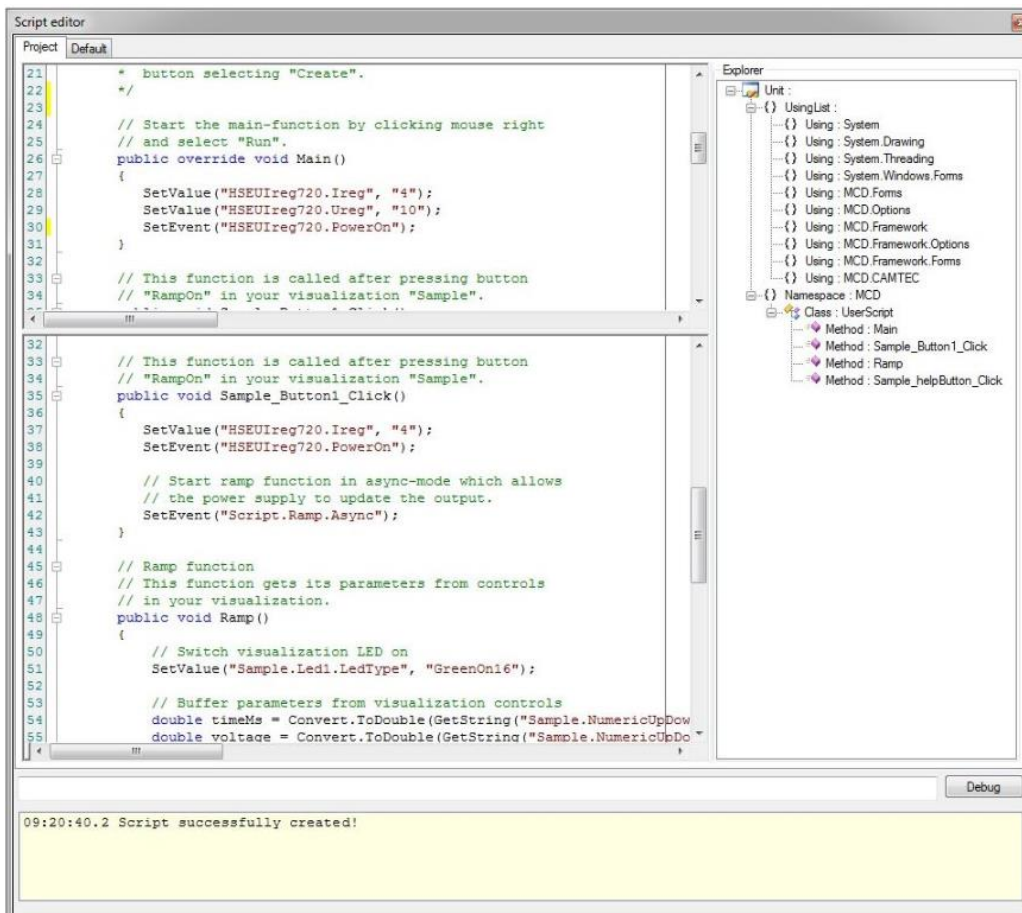


Figure 33: Second Window (Top Right Button)

### 4.2.2. Run Command

Using this command, the **Main() method** is exclusively called in the script asynchronously, assuming that it is available. There are two possibilities to execute the **Run command**:

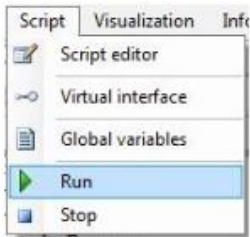


Figure 34: Possibility 1 (Script → Run)



Figure 35: Possibility 2 (Right click → Run)

### 4.2.3. Stop Command

A possibly running script will be stopped. There are two possibilities to stop the running script:

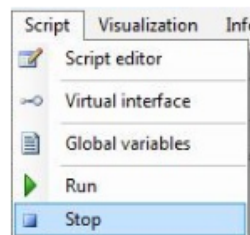


Figure 36: Possibility 1 (Script → Stop)



Figure 37: Possibility 2 (Right click → Stop)

#### 4.2.4. Integrating DLLs

External .Net assemblies can be integrated into the *script engine* of a Toolmonitor via the script settings. Their content will then be available in the Toolmonitor.

As can be shown in the figure below, three important .Net assemblies are already integrated by default for new projects. The *System.dll*, the *System.Drawing.dll* and the *System.Windows.Forms.dll*.

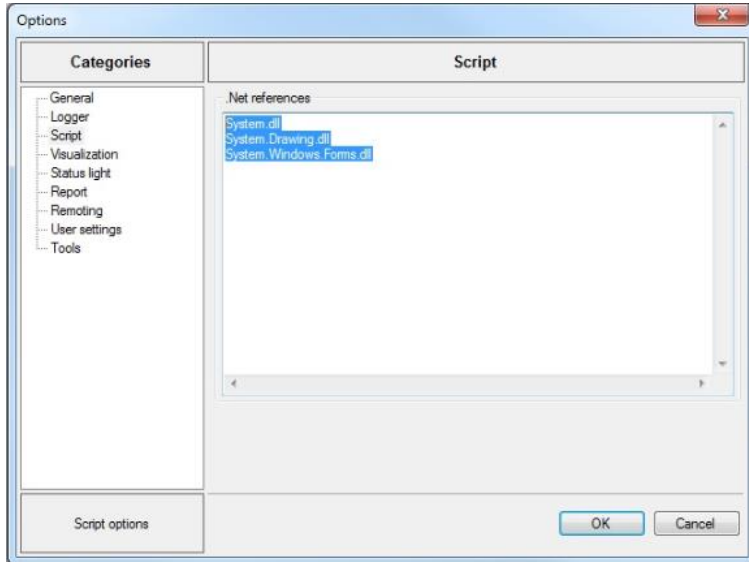


Figure 38: .DLLs in a New Project

In order to integrate further .Net assemblies, the name of the **.DLL** is entered into the list and confirmed with "OK" as shown below. Here, the *System.Xml.dll* and a self - created *ClassLibraryBeispiel.dll* were integrated for demonstration purposes. It is important that the **.DLLs** are in the same directory as the Toolmonitor. To jump one line lower in the input window, „Ctrl + Enter“ must be pressed.

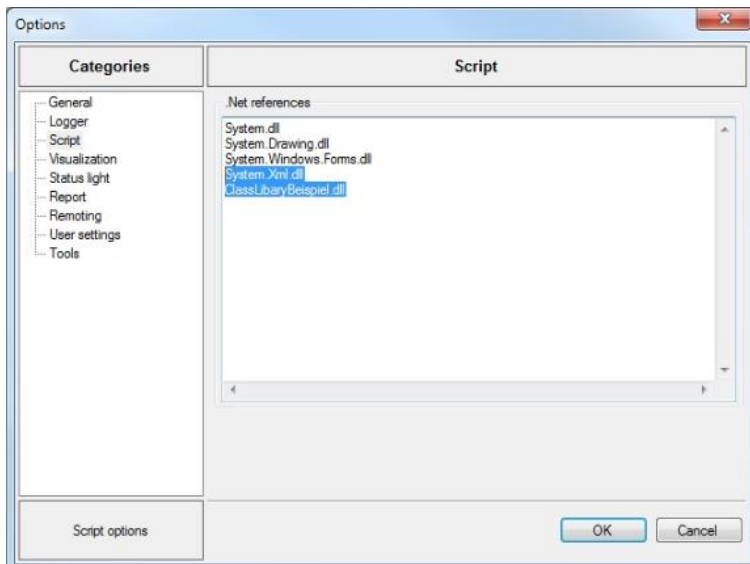


Figure 39: Newly Added .DLLs



#### 4.2.5. Global Variables

Using the **Global Variables** window, global variables can be created, deleted or changed.

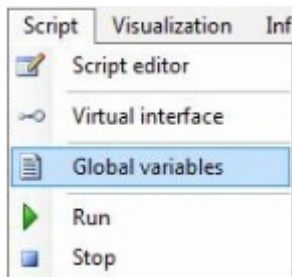


Figure 40: Opening Global Variables

New variables can be added ("Add") or deleted ("Delete") via right click. Four data types (*Real*, *String*, *Vector*, *StringVector*) are available. The syntax for the respective data types can be seen in the example below.

Type	Name	Value
Real	integer_test	20
String	string_test	"hello world"
StringVect	stringvector_test	["hello world"]
Vector	vector_test	[20]

Figure 41: Examples of Global Variables

#### 4.2.6. Virtual Interface

Using the form for the **Virtual Interface**, the Toolmonitor functions which are available there can be executed and tested. It presents the interface for communication via the COM interface.

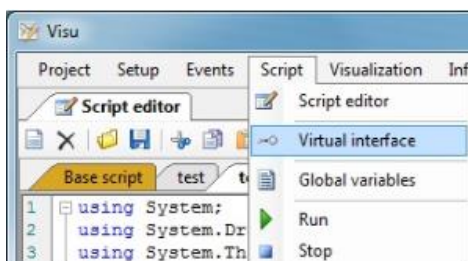


Figure 42: Opening the Virtual Interface

To control a Toolmonitor via this interface, the Toolmonitor must be registered as COM server first.

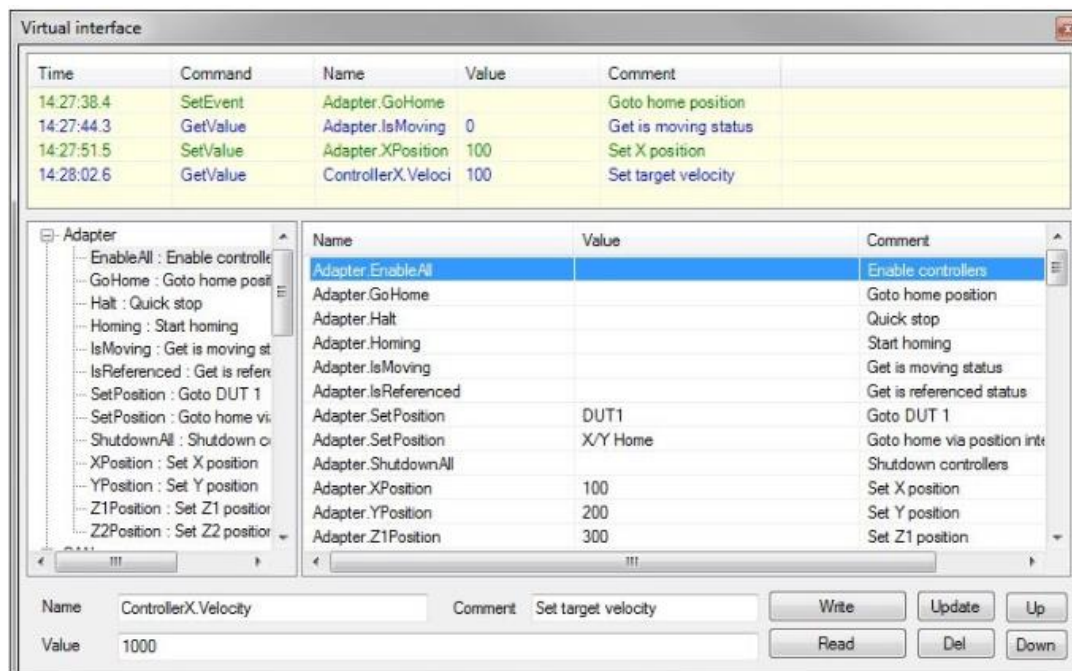


Figure 43: Virtual Interface

All possible commands for the *Virtual Interface* can be displayed by right - clicking "Add Defaults" in the virtual interface.

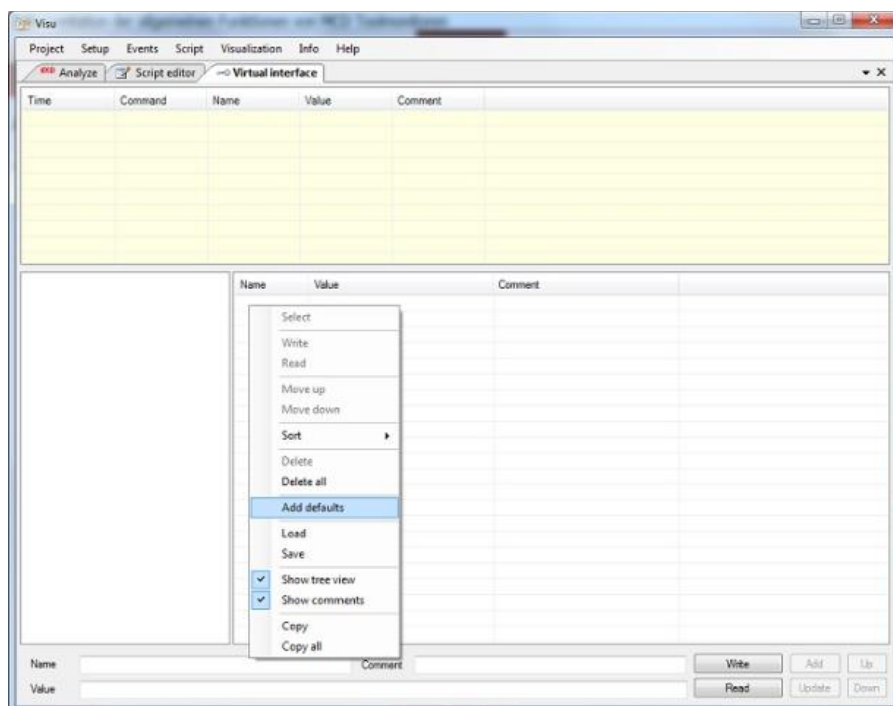


Figure 44: Displaying the Defaults

Before adding the commands a second time, they should be deleted by right - clicking "Delete All". Otherwise, every command will appear twice in the list.

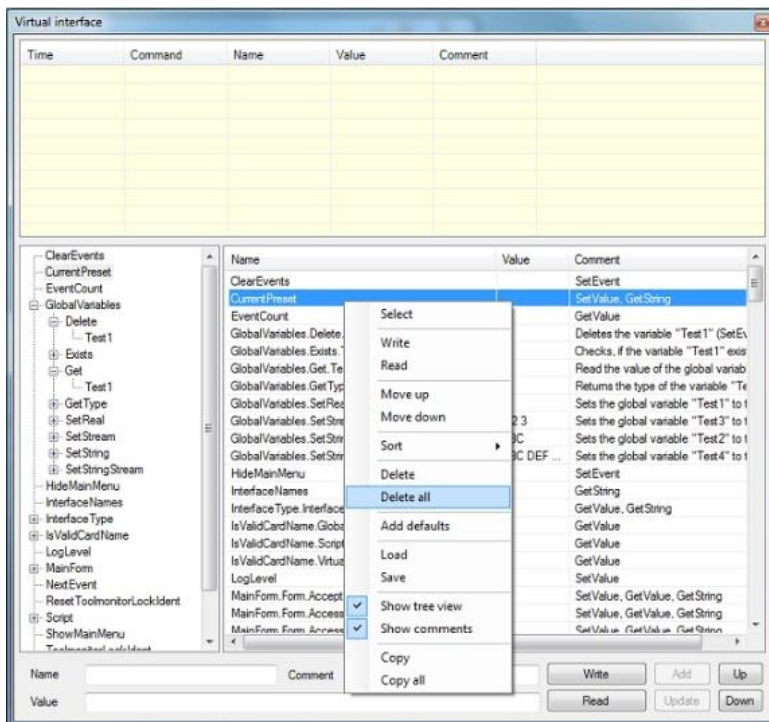


Figure 45: Deleting All Commands

A more detailed documentation of the available commands and parameters can be found in the class documentation of the **General Help** and in the specific help.

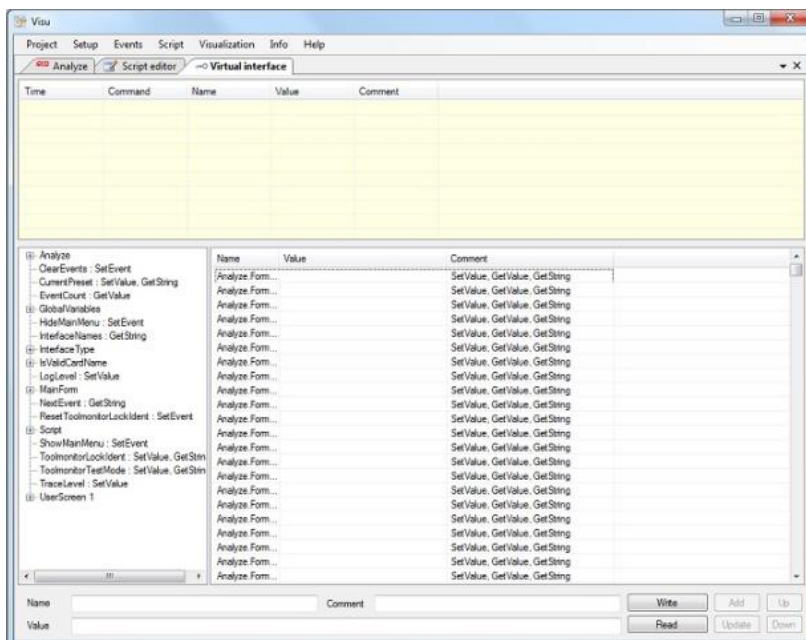


Figure 46: All available Commands for the Virtual Interface

Double - clicking on an element (here: *Global.Variables.SetReal.Test1*) applies its properties to the **Name**, **Value** and **Comment** fields (bottom). In the example here, the value of the "Test 1" variable is set to 10. The commands for reading and writing can be sent using the **Read** and **Write** buttons. Alternatively, the *Read* and *Write* commands can be reached via right mouse click. The return values and results of the instruction then appear in the window at the top of the form.

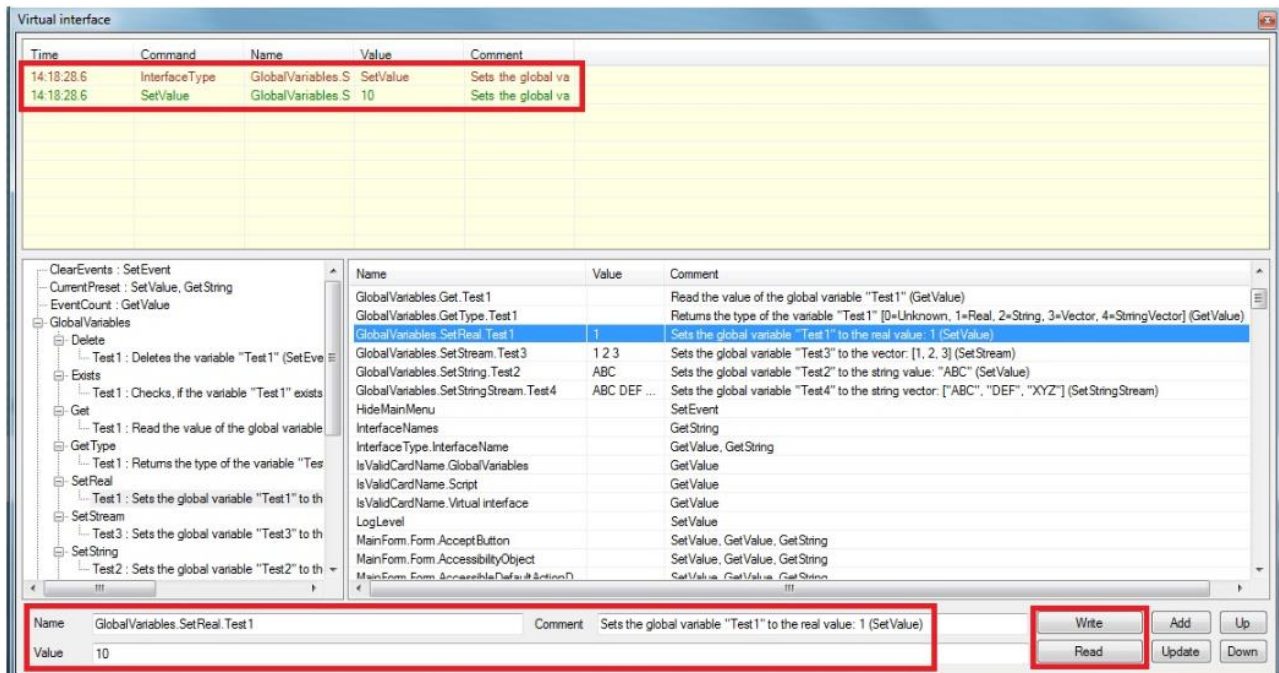


Figure 47: Read / Write Commands

### 4.3. Report

Using the menu **Report**, a user - defined report can be created depending on the Toolmonitor and the settings in the setup. To activate this menu item, the number of **Report Views** must be created in the options of the Toolmonitor. These are independent forms, in which different reports can be designed and created via **FastReport**.

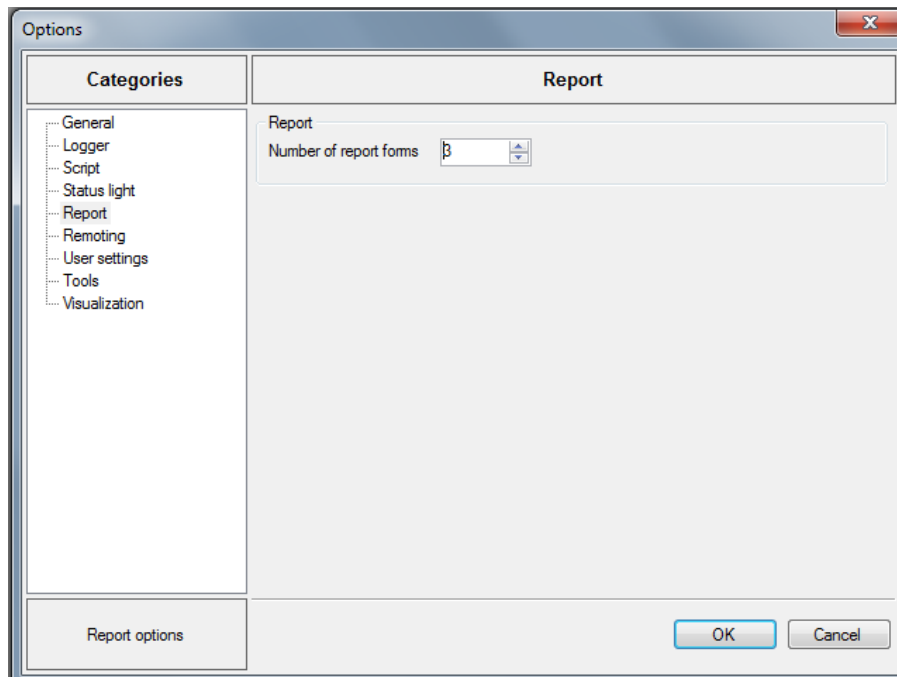


Figure 48: Changing the Number of Reports

To create a **new Report View** select the *Report tab* under **Setup → Options** in the menu bar as described here and adjust the desired number of forms in the selection box. In the figure below, three forms were created. After creation, the additional *Reports* option appears in the menu bar. Here, it can be selected between the created forms.

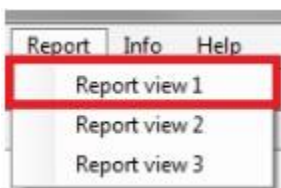


Figure 49: Opening the Report View

After the created **Report View** was opened as shown above, an empty interface appears with the menu bar shown below. Four buttons are available at the top of the menu: *Refresh*, *Load report*, *Designer* and *Export*.

The *Report preview* can be refreshed via the **Refresh** button. For example, if a time is displayed or a change was made in *Designer*, it can be refreshed here. For this purpose, the *Designer* must be exited. Otherwise, an error message appears at the bottom of the screen.

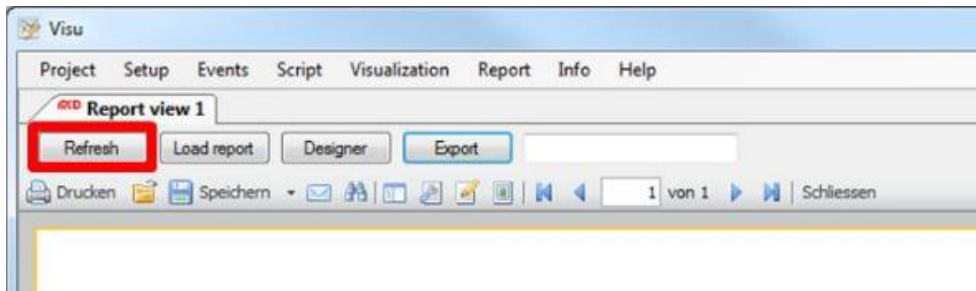


Figure 50: Refresh Button

An already available report can be loaded via the **Load report** button. During this process, the *Designer* must be exited. Otherwise, an error message appears at the bottom of the screen.

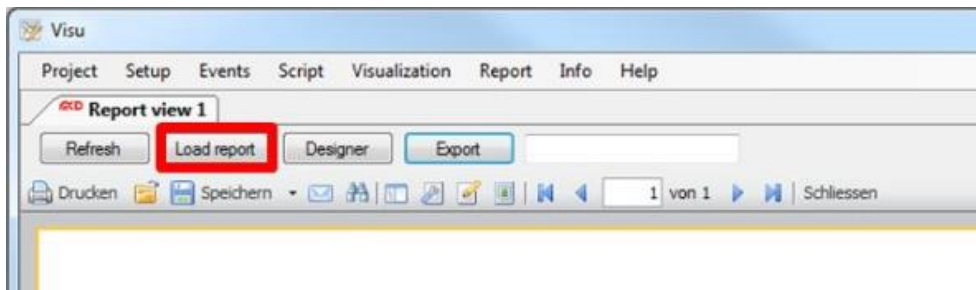


Figure 51: Load Report Button

The **FastReport Designer** opens via the **Designer** button. Using the *Designer*, layouts for new reports can be created as well as existing reports changed and / or edited. A detailed description of *Designer* and all available components can be found in the *FastReport User's Manual* under [https://www.fast-report.com/public\\_download/FRNetUserManual-en.pdf](https://www.fast-report.com/public_download/FRNetUserManual-en.pdf).

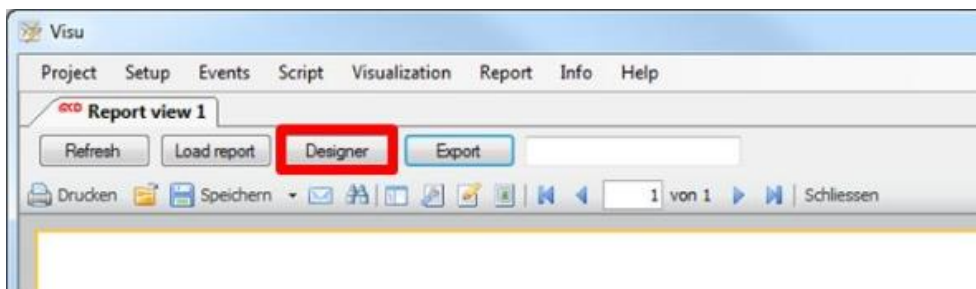


Figure 52: Designer Button



Using the **Export** function, the report can be directly exported (see example “Beispielbericht.html” below). If no specific path is indicated, but only the file name, the export is stored in the directory, where the Toolmonitor can be found.

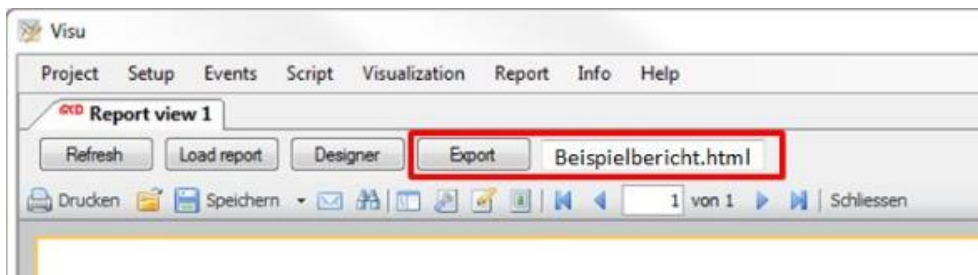


Figure 53: Export Button

In addition to the four described buttons, several icons are available in the toolbar. The report can be printed via the *Print* icon as in any Windows application.

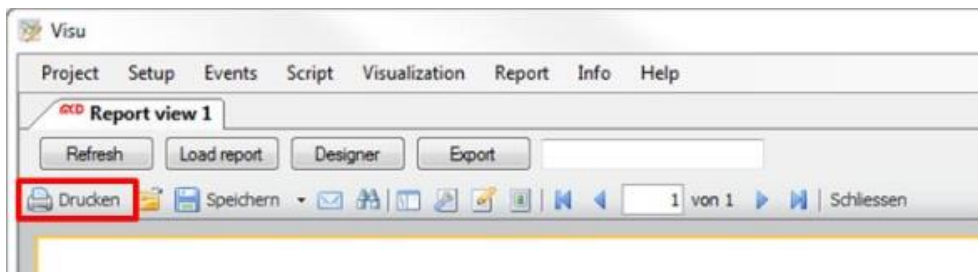


Figure 54: Printing the Report

Already existing reports can be opened via the *Open* icon.

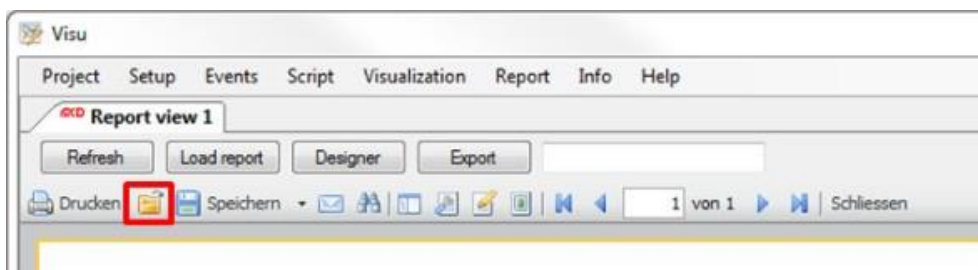


Figure 55: Opening the Report

The created report can be saved in different formats using the *Save* icon. The following selections are supported: PDF, ODS, ODT, Excel, XML, RTF, HTML, text, CSV, BMP, Jpeg, Tiff, and Gif.

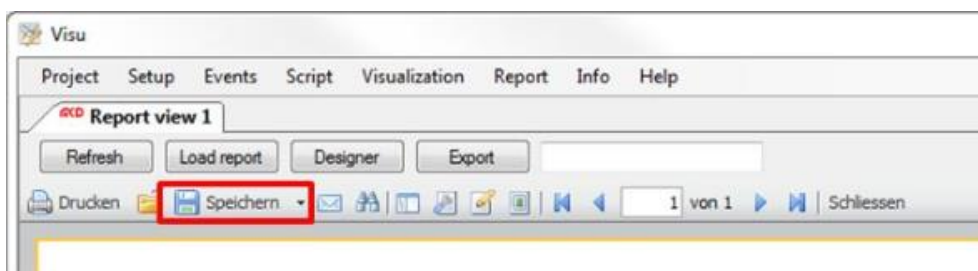


Figure 56: Saving the Report

Using the *Search* icon, the report can be searched for keywords.

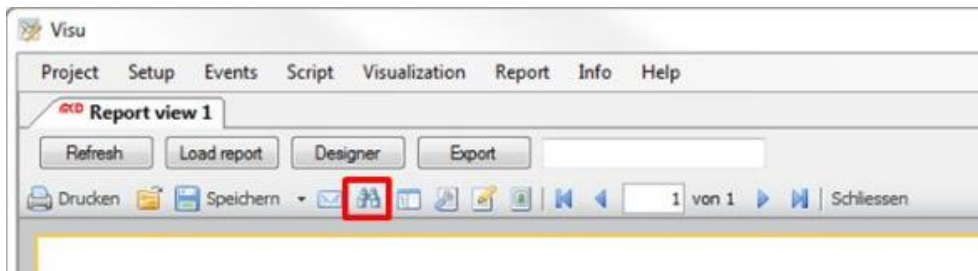


Figure 57: Searching for Keywords

The report can be displayed in a frame using the *Frame* icon.

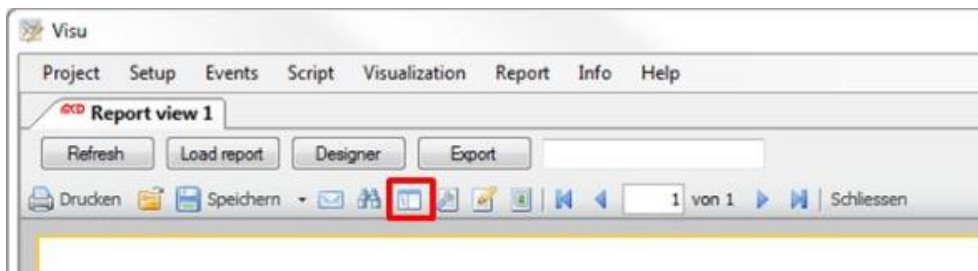


Figure 58: Showing / Hiding the Frame

The paper format or the report dimensions can be adjusted using the *Setup Page* icon.

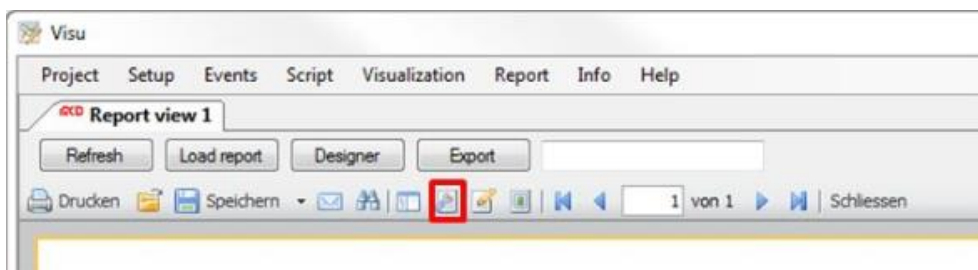


Figure 59: Editing Page Layout

The *Designer* can be directly accessed and the report edited via the *Edit Report* icon.

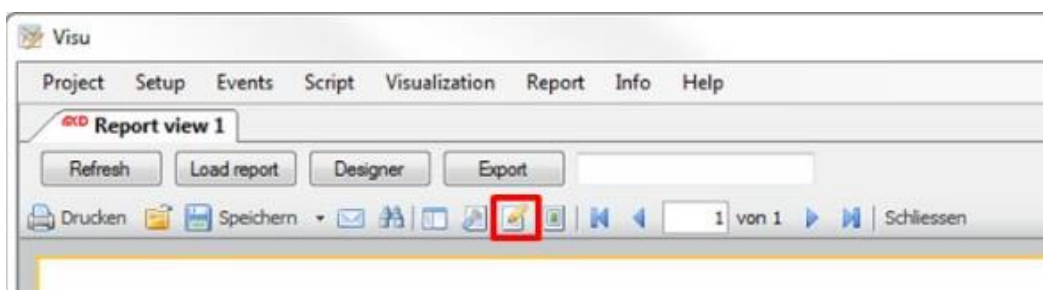


Figure 60: Editing Report in the Designer



Using the *Watermark* icon, an image, or text can be added to the report as watermark.

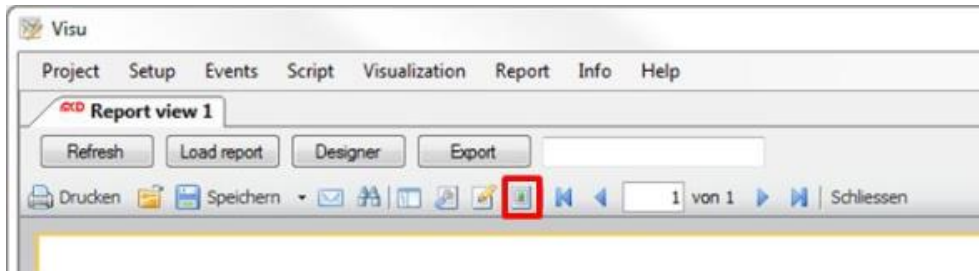


Figure 61: Inserting a Watermark

This toolbar can be used to switch between all available pages of the report.

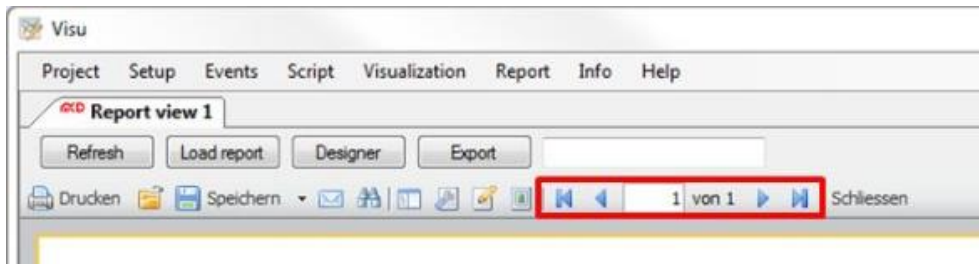


Figure 62: Displaying a different Page

The *Report view form* can be exited using the *Close* button.

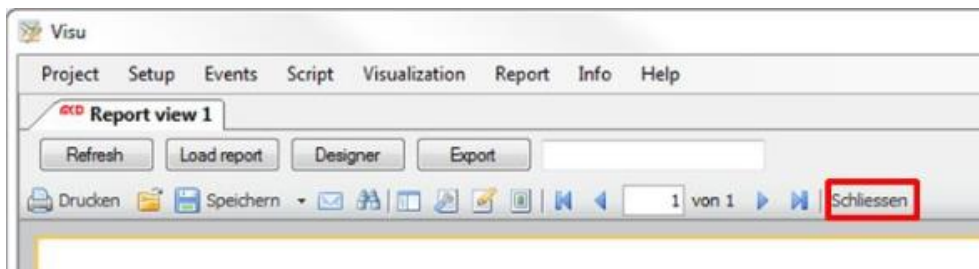


Figure 63: Closing the Report

## 5. Events

Here, the views for the log and trace messages can be accessed.



Figure 64: Events Menu

### 5.1. Logging

Using this menu, the log messages view for general events, warnings, errors, etc. is called up.

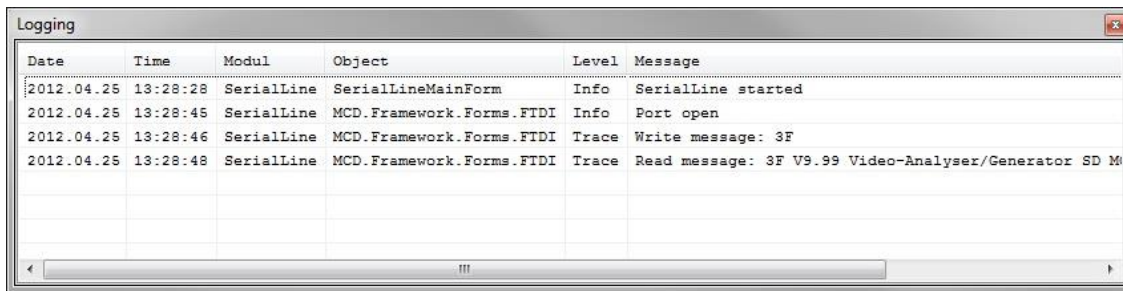


Figure 65: Log Monitor

### 5.2. Trace

Using this menu, the trace messages view (sent and received messages) is called up.

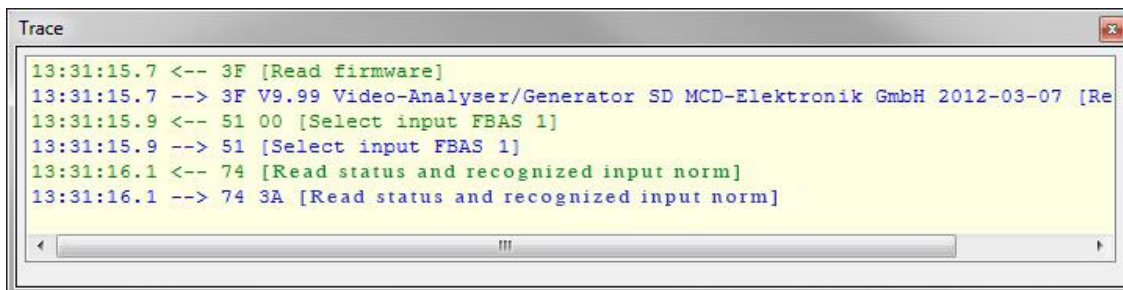


Figure 66: Trace Monitor

## 6. Debugger

Using the **Debugger**, you can debug scripts in the MCD Toolmonitor. The *Debugger* can be called directly from the script. This simplifies the creation of your C# functions. The user interface of the *Debugger* is shown in Figure 67.

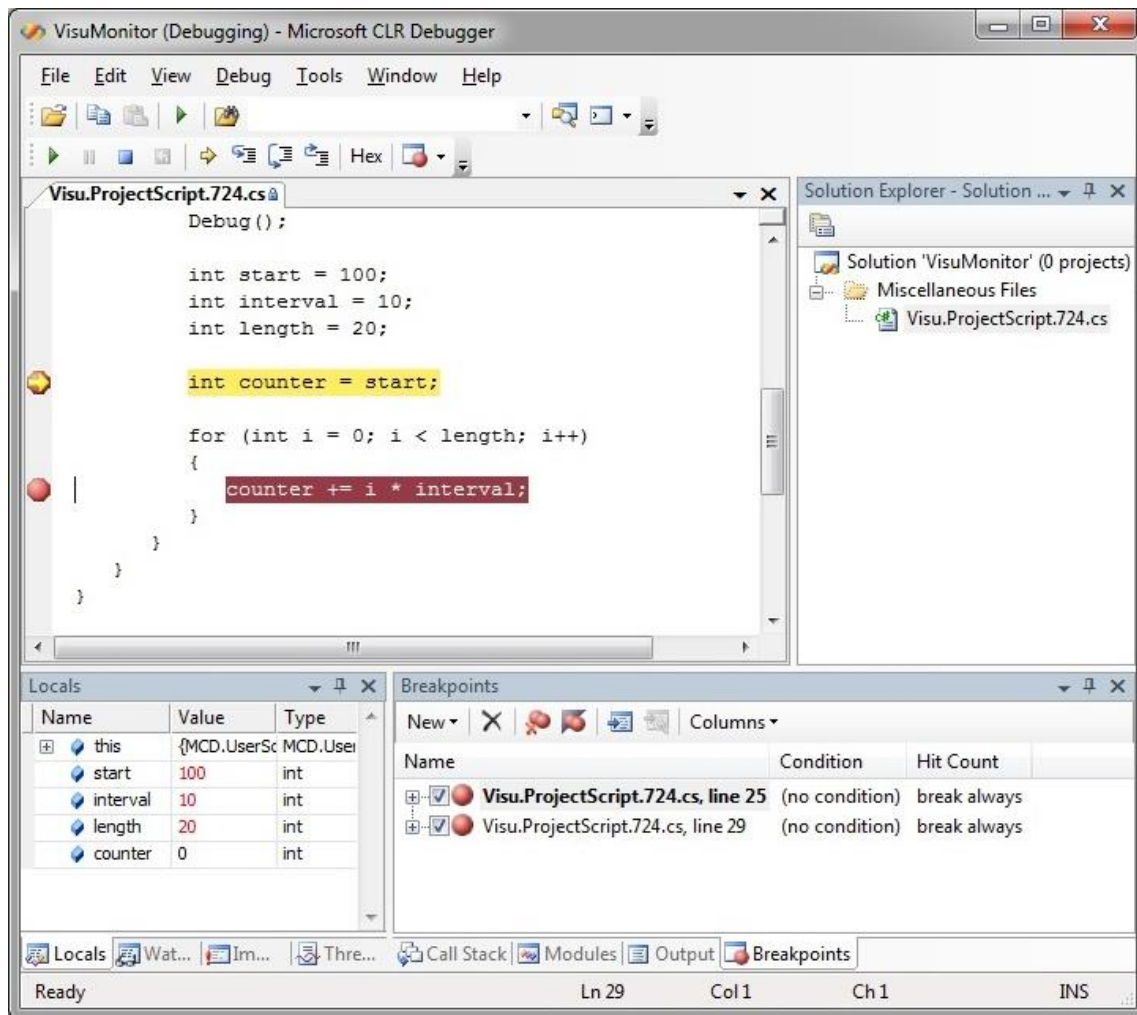


Figure 67: Debugger User Interface

### 6.1. Debugger Installation

The *Debugger* must be installed on your system. For this purpose, execute the provided installer *Microsoft.Net SDK.exe*.

You can also download the required installer from the Internet:

[Download Microsoft.Net SDK 2.0](#)

Search for "Microsoft.Net SDK 2.0". You can use the *Debugger* after the installation.

## 6.2. Starting the Debugger

Please open the *context menu* in the *Script Editor* via right click to start the *Debugger*. You can then open the *Debugger* via the **Launch debugger** entry.

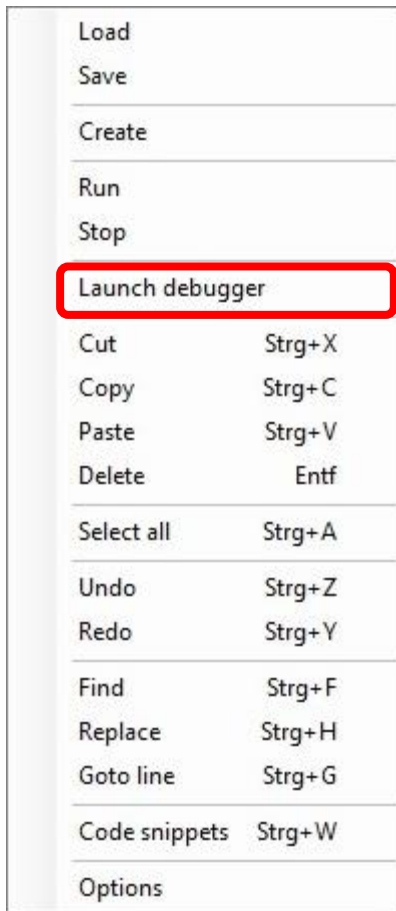


Figure 68: Starting the Debugger

There are two possibilities to stop running your script in the *Debugger*. You can set a **Breakpoint** in the *Debugger*. The use of *Breakpoints* is described in *chapter Using Breakpoints*.

It is also possible to define the start point for the *Debugger* in the script using the **Debug()** command (see figure 69, line 19). The *Debugger* starts at this position. This way, you can start the *Debugger* in certain situations, e.g., after inquiring a condition in an “if instruction”.

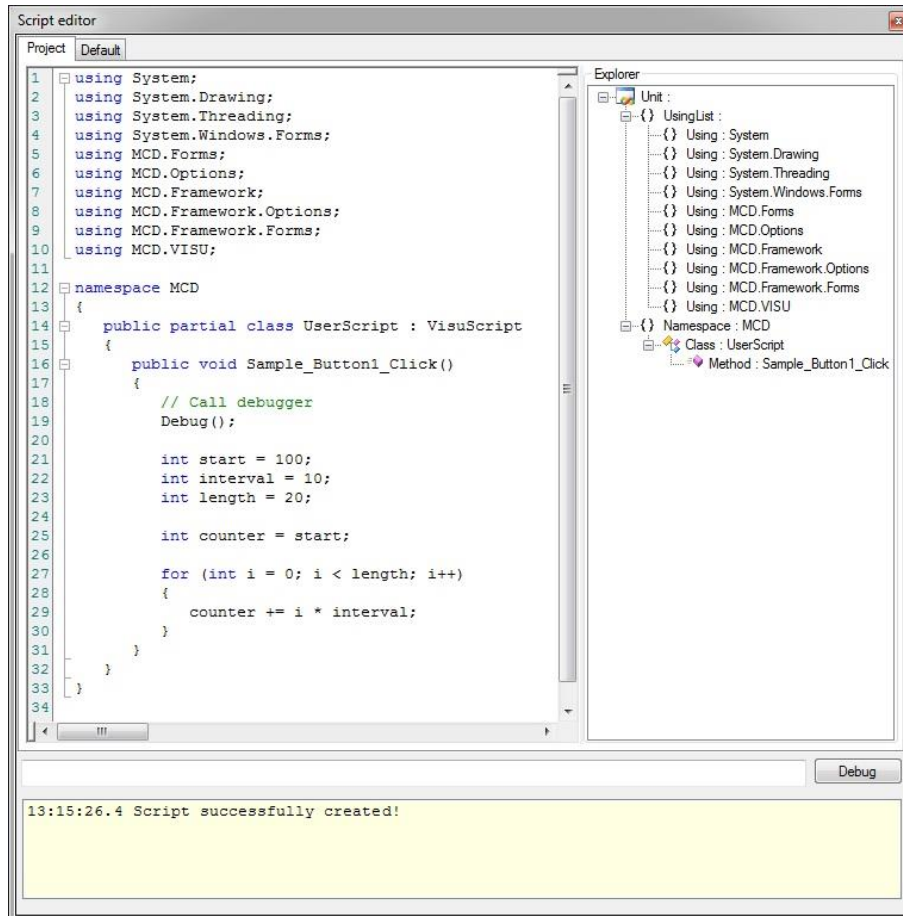


Figure 69: Debug Command in the Script Editor

You can now start actually debugging your source code. For this purpose, switch into the *Script Editor* and / or your *visualization interface* and start the desired code section by operating a control or starting the script via "run". Now, switch into the *Debugger*. If you inserted a *Breakpoint*, you can now see a yellow arrow at this position. It signals that program processing stopped here.

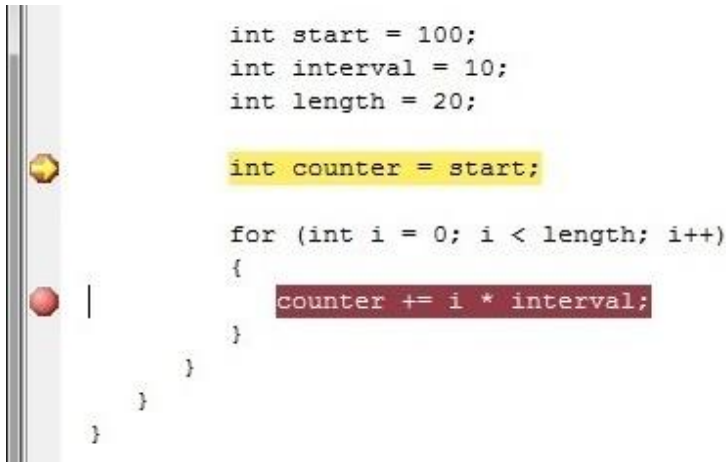


Figure 70: Stopping the Debugger with a Breakpoint

If you entered a *Debug()* command, the debugging is started at this position. This is marked by a green arrow.

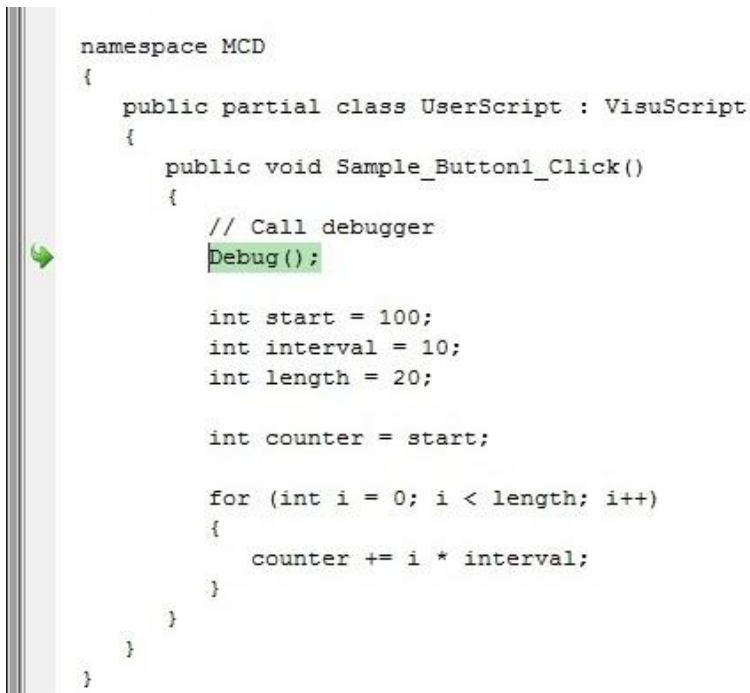


Figure 71: Starting Debugging at the Start Position

### 6.3. Debugging the Script



Figure 72: Debugger Toolbar

The easiest way of navigating the *Debugger* is via the **Debug toolbar** (see figure). The following table shows an overview of the most important functions:

	Run - Runs the program until the end or the next breakpoint
	Stop - Stops the debugger
	Single step - The debugger's instruction pointer is set to the next instruction
	Process step - The debugger's instruction pointer is set to the next instruction (Functions are processed in one step)
	Return - The instruction pointer returns to the calling function
Hex	Hexadecimal display - The representation of variables can be switched between decimal and hexadecimal

Figure 73: Function Overview of the Toolbar

### 6.4. Using Breakpoints

**Breakpoints** can be used for stopping script execution at a certain point. A *Breakpoint* can be activated via mouse click on the gray column on the left of the desired command. To deactivate the *Breakpoint* simply click it again.

Once you started the *Debugger* and set a *Breakpoint*, you can activate the execution of all commands up to the *Breakpoint* using the button *Next*. As can be seen in the figure, the *Debugger* stops with a yellow arrow at the desired command. You can now analyze the variables and, e.g. debug further commands in *step mode*.

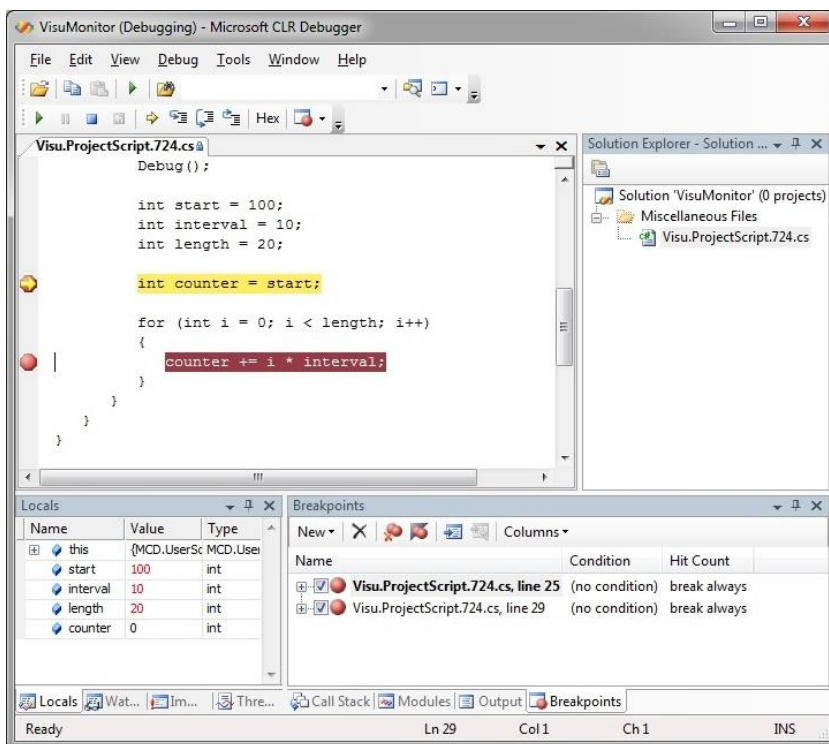


Figure 74: Stopping the Debugger with a Breakpoint



## 6.5. Monitoring Variables

Using the *Debugger* variable values can be displayed and changed. The easiest way of displaying a variable value is moving the mouse pointer over the desired variable name. The current value is then displayed automatically.

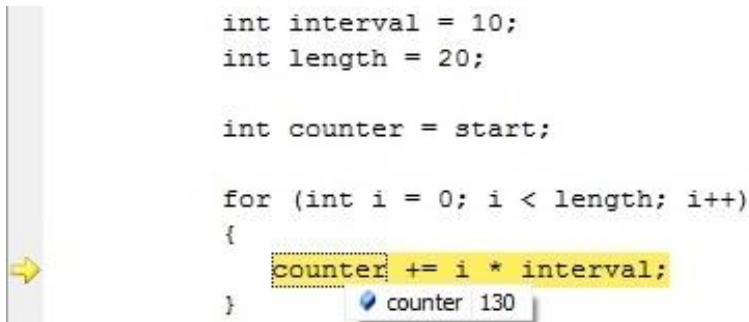


Figure 75: Displaying variable Values using the Mouse Pointer

Furthermore, you can display the currently applicable variables in the **window Local**. You can also add custom variables for monitoring in the **window Watch**. For this purpose, simply add the variable name to the **Name column**.

To change the value of a variable, you can adjust the entry in the **Value column** to the desired value. Please be aware that this is a manual intervention into the software sequence, which will not be available later on in the source code. Thus, this intervention should only be used for debugging purposes.

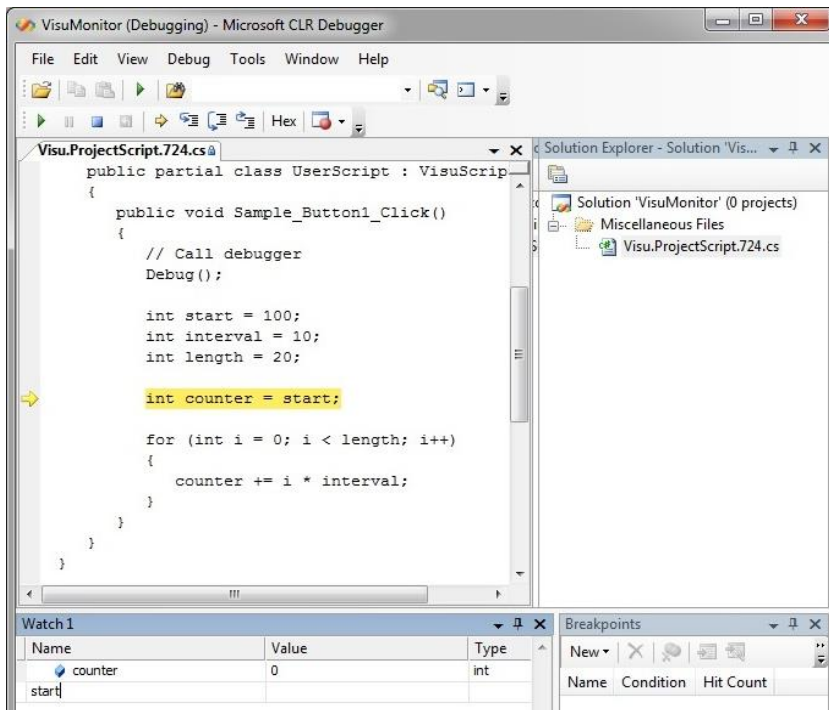


Figure 76: Monitoring Variables