

imc STUDIO 2026

Programming interface - Manual

Doc. Rev.: 9.1 - 2026-08-27



Disclaimer of liability

The contents of this documentation have been carefully checked for consistency with the hardware and software systems described. Nevertheless, it is impossible to completely rule out inconsistencies, so that we decline to offer any guarantee of total conformity.

We reserve the right to make technical modifications of the systems.

Copyright

© 2026 imc Test & Measurement GmbH, Germany

This documentation is the intellectual property of imc Test & Measurement GmbH. imc Test & Measurement GmbH reserves all rights to this documentation. The applicable provisions are stipulated in the "imc Software License Agreement".

The software described in this document may only be used in accordance with the provisions of the "imc Software License Agreement".

Open Source Software Licenses

Some components of imc products use software which is licensed under the GNU General Public License (GPL). Details are available in the About dialog.

A list of the open source software licenses for the imc measurement devices is located on the imc STUDIO/imc WAVE/imc STUDIO Monitor installation medium in the folder "*Products\imc DEVICES\OSS*" or "*Products\imc DEVICEcore\OSS*" or "*Products\imc STUDIO\OSS*". If you wish to receive a copy of the GPL sources used, please contact our tech support.

Notes regarding this document

This document is an excerpt from the manual of imc STUDIO.

The manual for imc STUDIO is available both as an e-book and in PDF format. In the menu bar, click on the symbol  to open the help.

The program "*imc Help and Documentation*" provides you with access to all formats and other documents such as the **device documentation** or the **technical data sheet**.

What is the best way to read the imc STUDIO documents?

Getting Started

Before installing the software, start by reading the document "*Getting Started*". This contains important tips for achieving problem-free installation, for updating the software, as well as for operation and integration of devices in measurement setups.

All information presented in the document "*Getting Started*" is also presented in the imc STUDIO manual.

Manual - complete documentation

The manual serves as reference material. This document describes how to operate the software and how to configure parameters for the devices. In some case, individual device groups have special functions. The documentation for these usually appear exclusively in the device manuals.

Tutorials

Some chapters come with supplemental tutorials. We recommend practicing these tutorials as an "accompanying" introduction to the respective topic.

If you are working for the first time with imc STUDIO, we recommend that you work through the guided first steps before beginning any tutorials.

Training programs for introduction to the systems, and advanced workshops

We recommend that before you begin working with imc STUDIO you participate in an extensive training session. Such training will enable you to get started working efficiently much faster. Additionally, you will obtain valuable tips and information on how to use the software more effectively. More information is available on our homepage under "*Service & Training*" > "*imc ACADEMY*".

Special notes



Warning

Warnings contain information that must be observed to protect the user from harm or to prevent damage to property.



Note

Notes denote useful additional information on a particular topic.



Reference

A reference in this document is a reference in the text to another text passage.

Table of contents

1 General introduction	6
1.1 Before you start	6
1.2 Tech support	6
1.3 Service and maintenance	7
1.4 Legal notices	7
2 Programming interface	8
3 Scripting	9
3.1 Menu bar and context menu	10
3.2 Access to functions	22
3.3 Script types	84
3.4 EClassID - Parameters	97
3.5 Script options	113
3.6 Tutorial	115
3.7 Important information	124
4 Third Party Device Interface	126
4.1 Getting started	126
4.2 Serial interface	128
4.3 Third party device script	129
4.4 3rd Party Device Management	137
4.5 Complete third party devices	139
4.6 Device script templates	146
4.7 FAQ	157
4.8 Tutorial	158
5 API	162
5.1 Licensing	162
5.2 Installation	163
5.3 Collaboration with the imc DATA API	163
5.4 Specifying the imc STUDIO version	164
5.5 Base interface	165
5.6 Sample projects	169
5.7 User-created application	173
5.8 Distribution	182
5.9 Notes	182
6 ME'Scope	183
6.1 Overview	183
6.2 Supported imc hardware	183
6.3 Licensing	183
6.4 Installation	183
6.5 Configuration	184
6.6 Notes	184

Index 185

1 General introduction

1.1 Before you start

Dear user.

1. The software you have obtained, as well as the associated manual are directed toward competent and instructed users. If you notice any discrepancies, we request that you contact our [tech support](#) .
2. Updates during software development can cause portions of the manual to become outdated. If you notice any discrepancies, we request that you contact our tech support.
3. Please contact our tech support if you find descriptions in the manual which you believe could be misunderstood and thereby lead to personal injury.
4. Read the license agreement. By using the software, you agree to the terms and conditions of the license agreement.



Note

Notes on the descriptions and the screenshots

- The help may also contain parts **shared imc software components**. These parts may differ from the rest of the help in terms of style and structure. All help files are equipped with a full text search functionality and have an index.
- The screen shots appearing in this documentation were created with a **variety of Windows versions** and their appearance may thus differ from that of your installed program.
- imc STUDIO works with user groups and access rights. The **user group** has wide-reaching influence on the visibility and/or operability of menus, icons, etc. Note that in all imc STUDIO documents the administrator roll is assumed to be filled. All descriptions are therefore related to the full utilization of the user interface.
- The following descriptions and screenshots always pertain to the **view with the full scope of functions** (*Complete*). Many of these functions are also found in the minimized view at another location.

1.2 Tech support

If you have problems or questions, please contact our tech support:

Phone: (Germany): **+49 30 467090-26** (Mon-Thu 09:00-12:00 and 13:00-16:00; Fri 09:00-12:00)

E-Mail: hotline@imc-tm.de

Internet: <https://www.imc-tm.com/service-training/>

Tip for ensuring quick processing of your questions:

If you contact us **you would help us**, if you know the **serial number of your devices** and the **version info of the software**. This documentation should also be on hand.

- The device's serial number appears on the nameplate.
- The program version designation is available in the About-Dialog (click on the symbol  in the menu bar).

Product Improvement and change requests

Please help us to improve our documentation and products:

- Have you found any errors in the software, or would you suggest any changes?
- Would any change to the mechanical structure improve the operation of the device?
- Are there any terms or explanations in the manual or the technical data which are confusing?
- What amendments or enhancements would you suggest?

Our [Tech support](#)  will be happy to receive your feedback.

1.3 Service and maintenance

Our service team is at your disposal for service and maintenance inquiries:

Phone: (Germany): **+49 30 629396-333** (Mon-Fri 09:00-12:00 and 13:00-17:00)

E-Mail: imc-service@axiometrixsolutions.com

Internet: <https://www.imc-tm.com/service>

Service and maintenance activities include, for example calibration and adjustment, service check, repairs.

1.4 Legal notices

Quality Management



imc Test & Measurement GmbH holds DIN EN ISO 9001 certification since May 1995 and DIN EN ISO 14001 certification since November 2023. You can download the CE Certification, current certificates and information about the imc quality system on our website:

<https://www.imc-tm.com/quality-assurance/>.

imc Warranty

Subject to the general terms and conditions of imc Test & Measurement GmbH.

Liability restrictions

This documentation has been prepared with the highest possible degree of care. It takes into account the standards and regulations in effect at the time of preparation, as well as the current state of the art. The functions and features described have been tested using the corresponding hardware and software. Nevertheless, discrepancies or errors cannot be completely ruled out. For this reason, the manufacturer does not guarantee that the documentation fully corresponds to the product. We reserve the right to make technical changes.

Liability exclusion

The manufacturer is not liable for damages resulting from:

- failure to follow this documentation or the "Getting Started" guide,
- or improper use of the product.

2 Programming interface

Section	Description
Scripting  9	Interface for using Scripts in imc STUDIO.
Third Party Device Interface  126	Interface for using third party devices in imc STUDIO.
API  162	Interface for developing custom application with access to imc STUDIO functions.

3 Scripting

The **imc STUDIO Scripting** is an **imc STUDIO** plug-in providing a programming interface based on C# and .NET to imc STUDIO. Various development environments (script editors) are supported for editing code; you must install them separately. We recommend Visual Studio Code or Microsoft Visual Studio. This chapter of the user manual describes basic entry points to Scripting and outlines the interfaces to imc STUDIO.



In imc STUDIO versions up to 2025 R5, the SharpDevelop development environment was included; however, it is no longer supported as of version 2026 R1. Descriptions of SharpDevelop are only included in the manuals for those versions. Existing scripts from those versions can still be used; see the section: [Important notes](#) ¹²⁴.

Besides user-defined programming, **fundamental functions** of imc STUDIO are available via Scripting:

- read and write device and channel settings/parameters (Setup),
- access to the Panel and the Widgets,
- access to the Data Browser: create, read and write variables,
- application of imc FAMOS-functions to channel data,
- execution of menu actions (ribbon),
- execution of Sequencer commands,
- respond to events.

The following **script types** are available:

- [Script](#) ⁸⁴
- [Panel Script](#) ⁸⁵
- [Context script](#) ⁸⁹
- [Type library script](#) ⁹³
- [Event script](#) ⁹⁵
- [Third Party Device script.](#) ⁹⁶

It is possible to **export** and **import** the **scripts**, also as DLL.

To **run a script**, the following mechanism are available depending on the script type:

- as command
 - in the Sequencer
 - at Widget
 - on events
- in the background
 - linked to a Panel page
 - linked to the experiment or project

Note

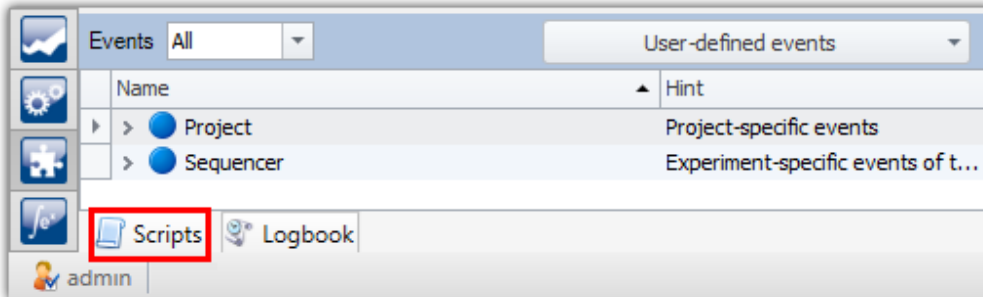
- For **creating** and **editing** a script an imc STUDIO *Developer* license is needed.
- Scripts¹ can be **run** in all editions without a special license.
- The programming language is C#.

¹ with the exception of Third-Party-Device scripts

3.1 Menu bar and context menu

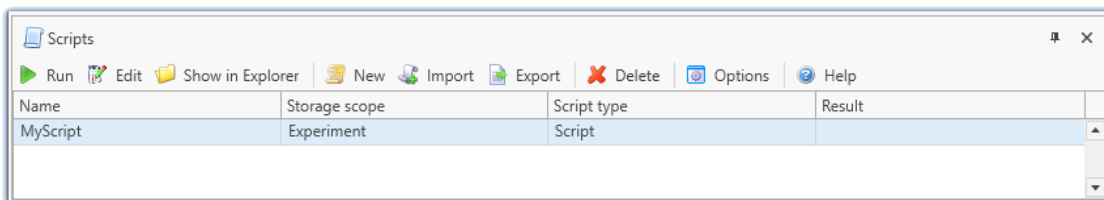
Tool window

To use the Scripting Plug-in, go to the Sequencer. There is a tab on the bottom called **Scripts**.



Sequencer: Tool window "Scripts"

In the tool window the scripts can be sorted by clicking on the column names.



Tool window "Scripts"

Note

If the *imc STUDIO Scripting Editor* (Developer-Edition) is activated and the tool window *Scripts* is not available, please check in the ribbon *View > Choose Tool Windows* if the tool window *Scripts* is activated.

Menu

Menu action		Description
Run		Runs ¹⁶ the <u>selected</u> script.
Edit		Opens ¹⁵ the <u>selected</u> script in the editor.
Show in Explorer		Show the <u>selected</u> script in the Explorer.
New		Opens the dialog for creating ¹² a script.
Import		Opens the Import-Dialog ¹⁸ .
Export		Opens the Export-Dialog ¹⁷ .
Delete		Deletes the <u>selected</u> script.
Options		Opens ¹¹³ the script options in the imc STUDIO options dialog.

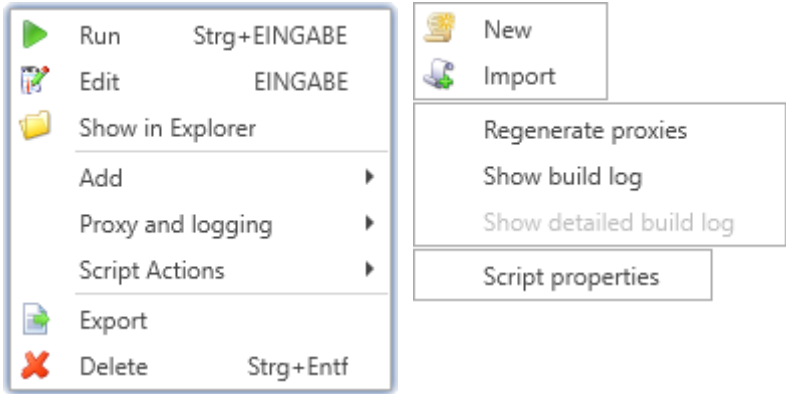
You can edit a script in two ways:

- by double-clicking the script, or
- by selecting the script and then clicking Edit in the toolbar.

The Edit function is also available in the context menu.

Only one script can be edited at a time. For this reason, the script editor must be closed before editing another script.

Context menu

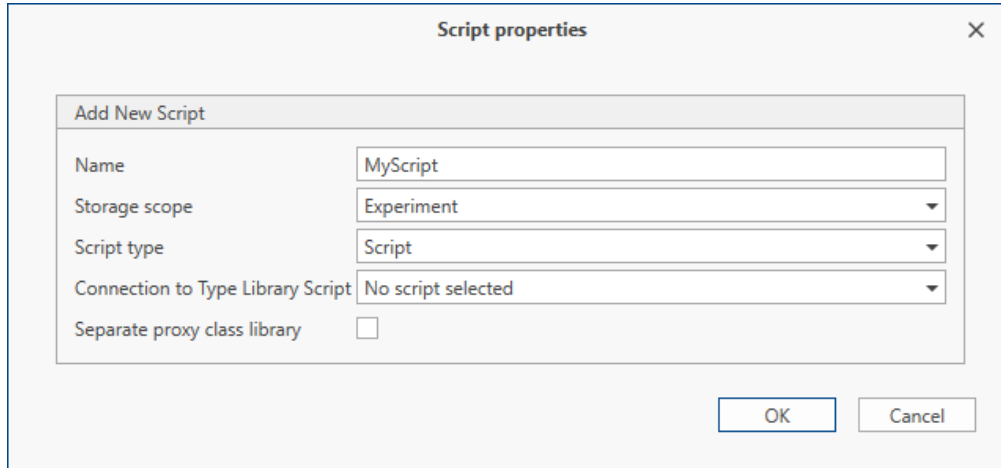


Menu actions	Action/Sub-menu
Run	Runs ¹⁶ the <u>selected</u> script.
Edit	Opens ¹⁵ the <u>selected</u> script in the editor.
Show in Explorer	Shows the <u>selected</u> script in the editor.
Add	• New¹² • Import¹⁸
Proxy and logging	• Regenerate proxies¹⁹ • Show build log • Show detailed build log
Script Actions	Script Properties: New Storage scope ¹⁴
Export	Opens the Export-Dialog ¹⁷ .
Delete	Deletes the <u>selected</u> script.

3.1.1 Create scripts

To create a new script (C#), switch to the Sequencer and open the tool window **Scripts**.

Click on **New**  **New**. To add new **scripts**, no script may be currently being edited (i.e., the editor or IDE must not be open). The following dialog box will open:



The image shows a dialog box titled "Script properties" with a close button (X) in the top right corner. Inside the dialog, there is a section titled "Add New Script". Below this title, there are five fields: "Name" with the text "MyScript", "Storage scope" with a dropdown menu showing "Experiment", "Script type" with a dropdown menu showing "Script", "Connection to Type Library Script" with a dropdown menu showing "No script selected", and "Separate proxy class library" with an unchecked checkbox. At the bottom right of the dialog, there are two buttons: "OK" and "Cancel".

Dialog: Script properties

For the other [script types](#)^[13], this dialog and the editor template looks different. See also the corresponding sections: [Panel-script](#)^[85], [Context-script](#)^[89], [Type-Library-script](#)^[93], [Event-script](#)^[95].

If the dialog is exited with **OK**, the **Script Editor** opens.

- If **no** program is associated with *.sln files on the computer:
 - A dialog box that allows you to select a program.
- If a program is associated with *.sln files on your computer:
 - The selected program.
- If a program has been specified via the options: That program.
(In the case of VSCode, this ensures that the directory is opened as a workspace.)

The programming language is C#.

Helpful examples are provided in the *Examples* portion of the source text.

3.1.1.1 Script type

The following **script types** are available. The **script type** should be selected which accords with the intended application.

Script Type	Description
Script 	A Script is called and run by means of the <i>run</i> command.
Panel Script 	- When creating a Panel Script an according <i>Panel page</i> has to be selected. - A Panel Script is linked to the corresponding Panel page. - The <code>Run()</code> method in the script is called when browsing to the corresponding page or when the page is already opened and the <i>Design mode</i> is deactivated. This is also the case when loading an experiment. - The <code>Stop()</code>-method is called when browsing to other pages or when the experiment is closed, or the <i>Design mode</i> is activated.
Context Script 	- A Context Script is linked to a <i>scope</i> (Experiment, Project, ...), - The <code>Run()</code>-method is called when opening an experiment (<i>Scope Experiment</i>). - The <code>Stop()</code>-method is called when exiting the experiments or when imc STUDIO is closed or another experiment is opened. (<i>Scope Experiment</i>). - Here, some of the possible actions are to register ("<i>hook</i>") actions with <i>events</i> and to de-register ("<i>unhook</i>") them from the events.
Type Library Script 	The Type Library Script is a way to provide one's <i>own classes</i> and <i>methods</i> for the other script types as a <i>DLL</i>.
Event Script 	- The Event Script is appended to the associated events as a <i>command</i>. - In the script, there is the possibility to access <i>arguments</i> of the events triggered, e.g. the measurement's name or its path for interval storage.
Third Party Device Script 	With the help of the Third Party Device Script, it is possible to integrate and use Third Party Devices in imc STUDIO.

When registering actions to, or de-registering them from events, it is preferable to use a **Context Script** (or for Widget-actions, a **Panel-script**) instead of a **normal script**.

For further information, see the corresponding sections: [Panel-script](#) , [Context-script](#) , [Type-Library-script](#) , [Event-script](#)  or rather [Third Party Device Script](#) 

3.1.1.2 Script properties

Script Properties	Description
Name	The script's name <u>must not</u> start with a <i>number</i> . <u>No spaces</u> or <i>special characters</i> are allowed, except for the underline. It also may not be the designation of any C# command (for, foreach, if, else, class, private, public, int, double, string ...).
Storage Scope	The script's <i>availability</i> , e.g. Experiment, Project, Application or Sequencer. To change this setting, click on <i>Script actions</i> > <i>Script properties</i> > New Storage Scope ¹⁴ .
Type	The selected script type ¹³ .
Connected Type-Library-Script	Selection from a list of created Type-Library-Scripts ⁹³ . The Type-Library-Script is available within the script as a library.
Separate proxy class library	The Proxy class is added as separate reference (DLL) and can be also used in other scripts, e.g. for global use of objects from the script's namespace, using the Script clipboard ⁸⁰ .
Activity Scope	<i>for Context scripts only:</i> Scope (Experiment, Project, Application), in which the script is active.



Note

Name assignment

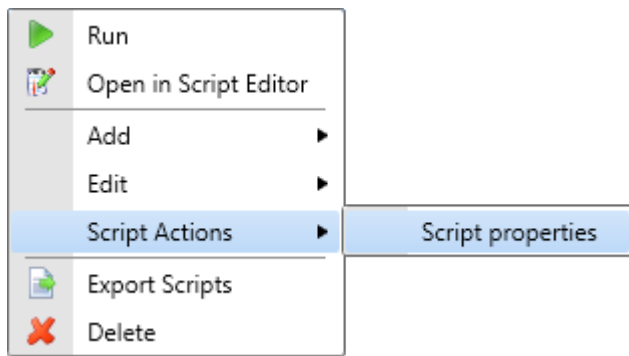
- Note that the same script **name** may not be used multiple times, even when using different **Script Storage Scopes**.
- **Example:**
 - In the **Experiment** *Exp_001*, a **script** "Test" is created with the **Storage Scope** *Experiment*.
 - In another **experiment**, *Exp_002*, a **script** "Test" is created with the **Storage Scope** *Project*.
 - The **experiment** *Exp_001* is loaded, and a warning appears in the logbook, indicating that the **script** "Test" cannot be loaded since it already exists (in the project).

3.1.1.3 Storage scope

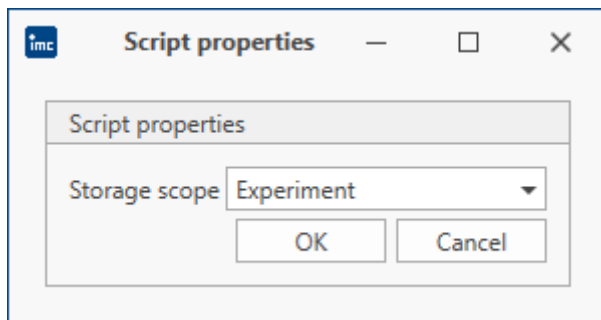
The option **Storage Scope** (*Save With / Script storage*) specifies the script's location in memory. In order to use scripts in other experiments belonging to the same project, the script is created with the setting *Save with* > *Project*.

Storage Scope	Description
Experiment	The script is saved in the <i>experiment folder</i> and is also only available within the experiment.
Project	The script is saved to the <i>project folder</i> and is listed and available to all experiments belonging to the project. If you wish to transfer the experiment to another system, the project-script must also be exported and transferred along with it.
Application	The script is saved to the <i>applications-folder</i> (e.g. C:\ProgramData\imc\imc STUDIO\Applications_1\Scripts) and is available for all projects and all experiments. Here, too, for the purpose of transferring the experiment to a different system, the script must be exported and transferred along with it.
Sequencer	Like Experiment, but with stated special rule.

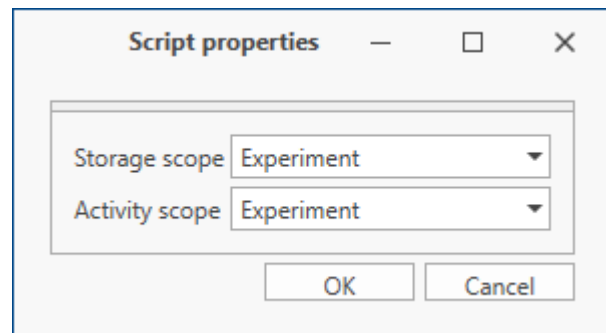
This option can be changed in the **context menu** *Script Actions > Script properties*.



Script Actions: Script properties



Dialog: Script properties



Dialog: Script properties (for Context-Scripts)



Warning

Save Project and Experiment

Please **save** your **project** and **experiment** after **changing** the *Storage Scope*, otherwise changes might be lost.

3.1.2 Edit scripts

To edit a script, please open the tool window **Scripts** and click on *Edit*  **Edit**. You can also double-click on a script to edit the script.

Only one script can be edited at a time. For this reason, no script should be open for editing (in an editor or IDE).



Note

The scripts are deposited beyond imc STUDIO.

The save state of the scripts is only transferred, if the corresponding *storage scope* (Experiment, Project) is **saved**.

3.1.3 Run scripts

To run a script, the following mechanism are available depending on the script type:

Script type	Mechanism
Script 	• Command in the Sequencer • Command on an event. • Tool window scripts ◦ Menu ◦ Context menu
Panel script 	• Browsing to a page (<code>Run</code> method) • Leaving the page (<code>Stop</code> method)
Context script 	• Loading/opening the corresponding context (<code>Run</code> method) • Leaving/closing the context (<code>Dispose</code> method)
Type library script 	• Is used in other scripts.
Event script 	• Command on an event.
Third Party Device Script 	Refresh of the Third Party Device after closing the editor. The Third Party Device Script is automatically updated under the following conditions: • During a build or rebuild An update occurs automatically during the build or rebuild process, provided that the editor or development environment being used supports post-build events. • When executing after changes If the "Execute" function is started, an update occurs only if changes have been made to the external device script since the last execution. • When restarting imc STUDIO When the imc STUDIO software is restarted, the external device script is automatically updated.

After running the script using **Run**, you can display the Build-Log by selecting *Edit > Show Build log* in the context menu.

If the script runs successfully, it will be indicated by a green dot symbol  in the tool window.

Otherwise, if there is any error, a red dot symbol  will be displayed.

All script types can be executed.

Execution:

- Compiles if changes are detected.
- Updates if changes are committed (the `Dispose()` / `Stop()` method is called).
- Initializes if not already done.
- Executes the `Run()` method (only with Script, Context, and Event).



The action that occurred is displayed next to the green dot.

3.1.4 Export and import

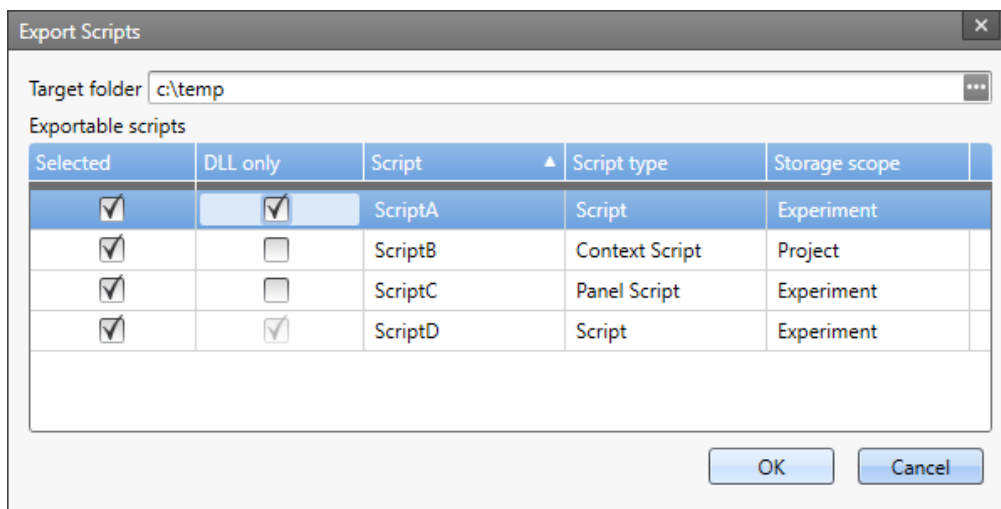
Note

If **Resources** are used in the *C#-Project*, please get sure, that they the property "Build action" is set to "Resource" or "Embedded Resource" and/or "Copy to output directory" is set to "true".

With these settings the Resources folder is considered when using **DLL-Export**.

Export

To export scripts, click in the **Scripts** tool window on *Export* . Subsequently, a dialog showing all scripts opens.



Dialog: Export scripts

Here, select the scripts to export and enter a target path.

if only the library (DLL) of a script is to be exported, then make the corresponding selection in the column "only DLL".

The scripts are exported as an archive (*.zip). Along with the script-files, some information is also recorded on the type and storage type of the ZIP-file. This also applies to the DLL-exports.

Warning

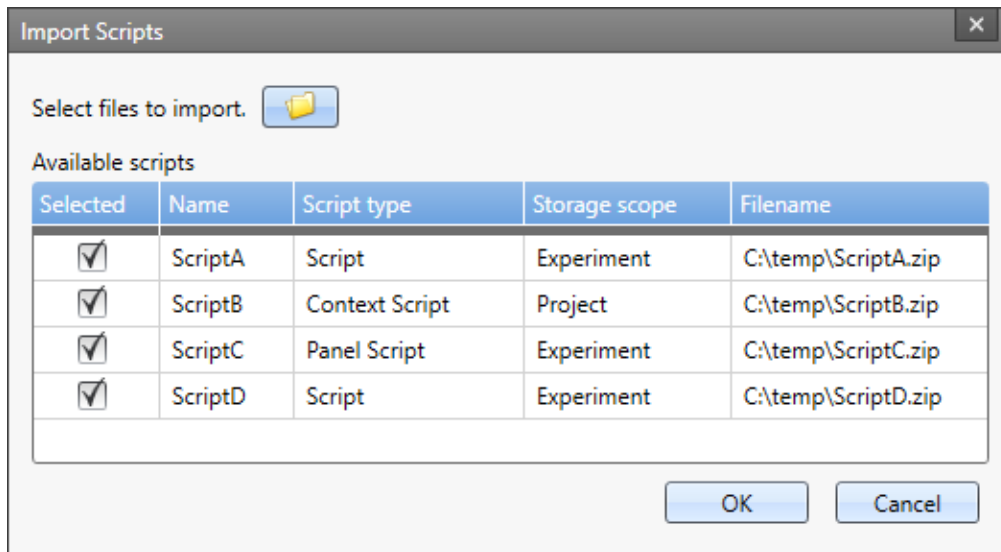
If a script is exported as DLL, there might be a dependency to assemblies of the current version.

If an experiment with DLL-scripts is used in a newer version, there might be an `FileNotFoundException` error.

The developer of the script must recompile it in the newer version. See also, [Important Information](#) ¹²⁴.

Import

Click in the tool window **Scripts** on *Import*  *Import* . The following dialog will appear:



Dialog: Add existing scripts

Choose the file at *Name*.

Depending on the script type, the dialog will change and further settings can be changed, e.g. Name and Storage Scope. Click on *OK* to add the script to the list. When scripts are imported which contain only a library (DLL), then of course these can not be edited but only executed.

Also, single script files (*.cs) can be added in the **ScriptEditor** to the C# project via *Add > Existing item*. Former script files with the extension *.csscript has to be renamed with the extension *.cs.



Warning

If you import a Panel script, you have to link the script to the corresponding Panel page in the page properties.



Note

Add "Import Scripts" to the ribbon

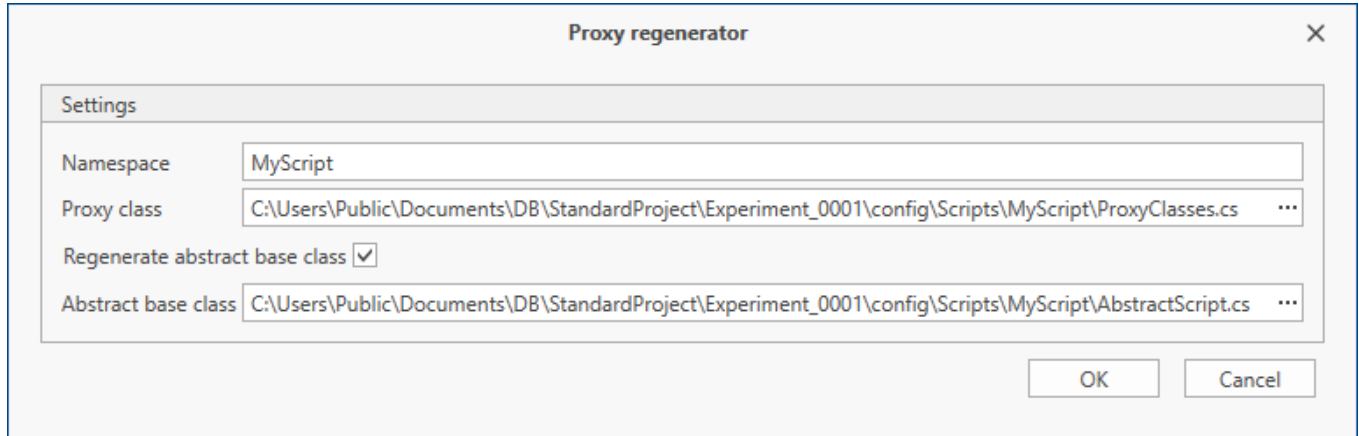
You can customize the ribbon and add the menu action "Import Scripts".
After adding the menu action the view needs to be saved.

3.1.5 Regenerate proxies

If a script of an old imc STUDIO version is imported and new functions are needed in this script, the proxy classes have to be regenerated.

Select the script in the tool window **Scripts** and click in the **context menu** on *Edit > Regenerate proxies*.

The following dialog opens.

The image shows a dialog box titled "Proxy regenerator" with a close button (X) in the top right corner. Inside the dialog, there is a "Settings" section. It contains four fields: "Namespace" with the value "MyScript", "Proxy class" with a file path "C:\Users\Public\Documents\DB\StandardProject\Experiment_0001\config\Scripts\MyScript\ProxyClasses.cs" and a browse button "...", "Regenerate abstract base class" with a checked checkbox, and "Abstract base class" with a file path "C:\Users\Public\Documents\DB\StandardProject\Experiment_0001\config\Scripts\MyScript\AbstractScript.cs" and a browse button "...". At the bottom right of the dialog are "OK" and "Cancel" buttons.

Dialog: Proxy regenerator



Warning

Due to new functions, internal structures may have changed. Therefore after regenerating proxies, you have to check and change the source code to fix the compiler errors.

You will find hints in the chapter [Important Information](#) ¹²⁴.

3.1.6 Debugging

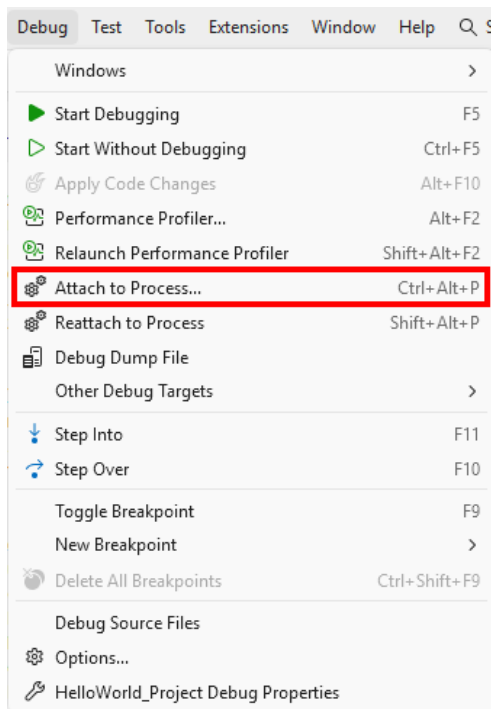
To **debug** a script, the following steps must be performed after compiling.

Step 1 : Add breakpoint

Press keyboard shortcut (depending editor) or use the menu item *Debug > Toggle Breakpoint* to add a breakpoint. The program counter will stop in this line, when reaching this statement.

Step 2 : Attach to imc STUDIO process

Attach the Debugger to the running imc STUDIO process by using the menu item *Debug > Attach to Process* In the dialog, select *imc.Studio.exe* (if multiple processes are listed, select the task whose name matches the experiment's name).



Editor: attach to process

Step 3 : Run the script

Run the script in imc STUDIO according to each script type. Perform the actions needed to reach the breakpoint.

3.1.7 Classes

Class name	Description
Actions	Access to the menu actions: Commands 
Commands	Access to the Sequencer commands 
Core	Access to classes (needed in other script types) <i>also see table on the below</i>
DataProcessing	* Access to Data Processing
DevSetup	Device and channel settings (parameters): Device functions  , Channel functions 
Documents	Informationen about the project and the experiment.
Environment	Informationen about the software environment: Product Information 
Events	Access to events.
InlineFamos	Access to InlineFamos: Inline FAMOS functions 
Logbook	Access to the logbook: Logbook 
Notifications	Access to the notification system.
Panel	Access to the Panel and the Widgets: Panel and Widget functions 
ScriptClipboard	Script Clipboard 
Scripts	Access to the scripts
SharedComponents	*
StorageManager	Used in an Event Script 
UserRequest	Default dialog response 
VariableViewer	* Current values window
Variables	Access to variables: Variables functions 
Windows	Access to tool windows. Show and hide.

* Access not possible or restricted.

Core

Some of the classes above can also be accessed via the Core class. In Addition there are the following classes.

Class name	Description
Accessibility	* Access to the user administration
Options	*
Services	*

* Access not possible or restricted.

3.2 Access to functions

3.2.1 Device functions

Note

- For setting the parameters, class *EClassID* is used. Setting values of the parameter are available in the *ParameterValues*-Class. See also [Device parameters](#)⁹⁷.
- The method `ExecuteActions()` is only needed for device and channel actions. For parameters the method `SetValue()` is used.

To use needed constants/classes the following **libraries** have to be added:

```
using imc.Studio.Tools.DevSetupGraphGenerator; // EClassID
using imc.Studio.Interfaces.DevSetup; // EDeviceCoreType_V2
```

The following line is needed for the following examples. The device `imcDev__17123456` is used. The device name is case sensitive.

```
var device = DevSetup.Devices["imcDev__17123456"];
```

The parameters are used as follows:

```
// EClassID Parameter as string or from EClassID-Class
//var parameter = channel.Parameters["eMyParameter"];
var parameter = device.Parameters[EClassID.eDeviceName]; // Example

// set parameter value
parameter.SetValue("SetValue");
// get parameter value
string parameterValueAsText = parameter.GetValue().ToString();
// get printout of the parameter value
string parameterText = parameter.Text;
// select list of parameter values, if available
string [] paraTextArray = parameter.TextList;
```

For further information, please read the following chapters:

- [Devices control and parameters](#)²³
- [Storage](#)²⁶
- [Diskstart](#)²⁷
- [Misc](#)²⁹

3.2.1.1 Devices control and parameters

Search for devices

The device search is performed by the following command:

```
DevSetup.SearchDevices();
```

Add Device by DNS-Name or IP

Add a device by DNS-Name or IP by the following command:

```
using imc.Studio.Interfaces.DevSetup; // add Library
// IP
DevSetup.SearchDevice("192.168.0.2", DeviceAdapters.imcDevcies2x);
// DNS
DevSetup.SearchDevice("imcDev__17123456", DeviceAdapters.imcDevcies2x);
// The return value of type "EnumItem" is the according device.
EnumItem device = DevSetup.SearchDevice("0.0.0.0", DeviceAdapters.imcDevcies2x);
```

Select a device

If the device name is unknown, the name can be determined by using the device's serial number.

```
// getting the device object via serial number
string serialNumber = "123456";
EnumItem device = DevSetup.Devices.GetItemBySerialNumber(serialNumber);
// getting the device identifier
string deviceName = device.Parameters[EClassID.eDeviceName].GetValue().ToString();
// getting the device name
string deviceNickName = device.Parameters[EClassID.eDeviceNickname].GetValue().ToString();
```

Actual selection of the device is based on the device name (*EClassID.eDeviceNickname*) or the device identifier (*EClassID.eDeviceName*).

```
using imc.Studio.Interfaces.DevSetup;
// selecting device via nickname
DevSetup.SelectDevice(deviceNickName, true, EDeviceCoreType_V2.Measurement);
// selecting device via name
DevSetup.SelectDevice(deviceName, false, EDeviceCoreType_V2.Measurement);
```

The device name can also be obtained by using the device list.

New parameters for the device are available, therefore the device object has to be assigned afterwards.



```
// via device name
EnumItem device = DevSetup.Devices[deviceName];
// via serial number
EnumItem device = DevSetup.Devices.GetItemBySerialNumber(serialNumber);
```

Deselect a device

```
DevSetup.DeselectDevice("DeviceNickName", true, EDeviceCoreType_V2.Measurement);
DevSetup.DeselectDevice("imcDev__17123456", false, EDeviceCoreType_V2.Measurement);
```

Remove device from List

```
DevSetup.RemoveDevice("DeviceNickName", true);
DevSetup.RemoveDevice("imcDev__17123456", false);
```

Device control actions

To connect to the device or start and stop the device, the following functions have to be executed.

```
var status = device.Parameters[EClassID.eDeviceControlAction];
// for action values see ParameterValues.eDeviceControlAction

// connect to device
status.SetValue(ParameterValues.eDeviceControlAction.Connect);
status.ExecuteAction();

// start measurement
status.SetValue(ParameterValues.eDeviceControlAction.Start);
status.ExecuteAction();

// stop measurement
status.SetValue(ParameterValues.eDeviceControlAction.Stop);
status.ExecuteAction();

// disconnect from device
status.SetValue(ParameterValues.eDeviceControlAction.Disconnect);
status.ExecuteAction();
```

For starting/stopping all selected devices:

```
DevSetup.ExecuteAction(ParameterValues.eDeviceControlAction.Start);
DevSetup.ExecuteAction(ParameterValues.eDeviceControlAction.Stop);
```

or

```
Actions.FireAction(this, "acDeviceConnect"); // connecting
Actions.FireAction(this, "acDeviceStart"); // starting
Actions.FireAction(this, "acDeviceStop"); // stopping
Actions.FireAction(this, "acDeviceDisconnect"); // disconnect
```

Query, if device is selected

```
bool isSelected = device.Parameters[EClassID.eConnectionStatus] != null;
```

List of selected devices (serial number)

```
using System.Collections.Generic;
using System.Linq; // may needs to be added to the references
using imc.Studio.Tools.DevSetupGraphGenerator;

List<string> serialNumbers = DevSetup.Devices
    .Where(device => device.Parameters[EClassID.eConnectionStatus] != null &&
        device.Parameters[EClassID.eDeviceSN] != null)
    .Select(device => device.Parameters[EClassID.eDeviceSN].Text)
    .ToList();
```

Measurement state

```
int mState = (int)device.Parameters[EClassID.eMeasurementStatus].GetValue();
string stateText = device.Parameters[EClassID.eMeasurementStatus].Text;
```

Hardware-ID

The device name parameter can be used to retrieve the device's hardware ID:

```
string hardwareID = device.Parameters[EClassID.eDeviceName].HWID;
```

Manufacturer

```
string manufacturer = device.Parameters[EClassID.eManufacturer].Text;
```



Warning

If *ExecuteAction* is used the according parameter/device should be read/assigned again due to internal changing instances. The consequence is a *NullReference* Exception. This could happen when selecting a device or reading TEDs, for instance.



Note

If several actions should be executed, e.g. multi-selection of devices, the methods `DevSetup.BeginTransaction()` and `DevSetup.EndTransaction()` can be used.

Example

Use the first device in the list

```
var deviceList = DevSetup.Devices.GetEnumerator();
deviceList.Reset();
deviceList.MoveNext(); // get next element in the list
EnumItem device = deviceList.Current;
if (device != null){
    string deviceName = device.Parameters[EClassID.eDeviceName].GetValue().ToString();
}else{
    // do something, e.g.
    deviceName = String.Empty;
}

// Alternative
using System.Linq;
// ...
EnumItem device = DevSetup.Devices.First(); // first known device in the list
```

3.2.1.2 Storage

Device specific options - Device

```
// Storage interval ; 0 = End of measurement
device.Parameters[EClassID.eUMIntervalTime].SetValue(60d); // 60 seconds
// Storage interval count
device.Parameters[EClassID.eUMIntervalCount].SetValue(128); // all = 0
// Storage location: 0: Removable medium, 1: Hard disk (HD); 2: Network drive
device.Parameters[EClassID.eUMDrive].SetValue(0);

// Save trigger events in individual files (valid for ALL devices!)
device.Parameters[EClassID.eExtraTriggerDirUMHD].SetValue(1);
```

Delete files on selected disk



All data on the selected storage devices are **deleted**.

Please assure, that the device storage location is set correctly, e.g. to the *removable medium*.

```
device.Parameters[EClassID.eUMDrive].SetValue(0); // Removable medium
device.Parameters[EClassID.eDeleteFilesOnSelectedDisk].ExecuteAction();
```

Experiment options (applies to all devices) - PC



These options can be set by setting the parameter of a device, but it will be applied for **all** devices in the experiment.

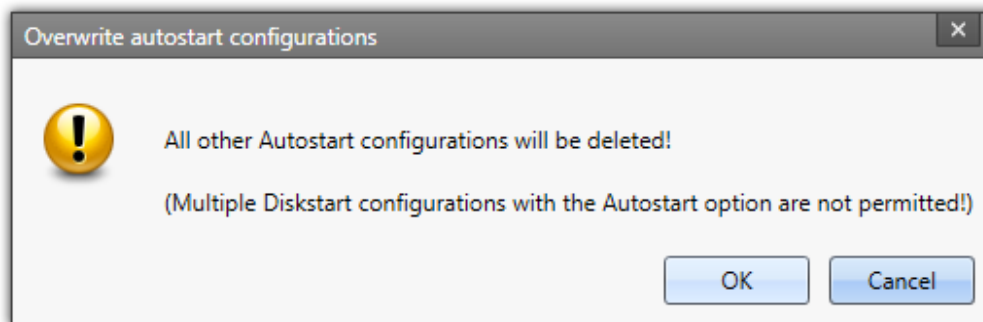
```
// Storage interval ; 0 = End of measurement
device.Parameters[EClassID.ePCIntervalTime].SetValue(60d); // 60 seconds
// Storage interval count
device.Parameters[EClassID.ePCIntervalCount].SetValue(128); // all = 0
// Save trigger events in individual files
device.Parameters[EClassID.eExtraTriggerDirPCHD].SetValue(true);
device.Parameters[EClassID.eExtraTriggerDirUMHD].SetValue(true);
```

3.2.1.3 Diskstart

Write Diskstart to destination

```
device.Parameters[EClassID.eDiskStartSelfStartOption].SetValue(true); // Self-start
var status = device.Parameters[EClassID.eDiskStartAction];
// See ParameterValues.eDiskStartAction for setting values, e.g.
// 2503: Device HD; 2505: Device intern;
// saving to removable device
status.SetValue(ParameterValues.eDiskStartAction.Device_PCMCIA);
status.ExecuteAction();
// saving to PC-HD
status.SetValue(ParameterValues.eDiskStartAction.Device_PC_HD);
status.ExecuteAction();
```

When writing an self-start experiment to the device, a dialog which has to be confirmed **manually** appears.



Dialog: Autostart

If you want to suppress this dialog, open the *Options-Dialog* and go to *General Options > Default dialog response > Overwrite autostart configurations* and choose **OK** from the selection.

This can also be done with Scripting (see chapter [Default dialog response](#) ⁷⁹):

```
using imc.Studio.Interfaces.Gui;
UserRequest.SetDefaultResultValue("Dev2x_OverwriteSelfStart", EMessageBoxResult.OK);
```

Deletes Diskstart in every selected device

```
foreach(var config in DevSetup.Items[EClassID.eConfig])
{
    var deleteAction = config.Parameters[EClassID.eDiskStarDeleteConfig];
    if((deleteAction != null) && (deleteAction.Base != null))
    {
        // deleteAction.SetValue(ParameterValues.eDiskStarDeleteConfig.Delete);
        deleteAction.ExecuteAction();
    }
}
```

Get Diskstart information

```
// Experiment-Name
var names = DevSetup.Items[EClassID.eConfig]
    .ToList()
    .Where(config => config.Parameters[EClassID.eDiskStartConfigName] != null)
    .Select(config => config.Parameters[EClassID.eDiskStartConfigName].Text);

// Storage location
var location = DevSetup.Items[EClassID.eConfig]
    .ToList()
    .Where(config => config.Parameters[EClassID.eDiskStartLocation] != null)
    .Select(config => config.Parameters[EClassID.eDiskStartLocation].Text);

// Show information
if (names.Any() && location.Any())
{
    MessageBox.Show(names.First() + "\n" + location.First());
}
```

3.2.1.4 Supplemental Files

Access to **supplemental files management** is available through `DevSetup.SupplementalFiles`.



If an error occurs, an exception is thrown and must be handled accordingly.

Hardware ID

Via the parameter of the device name the hardware ID of the device can be queried:

```
string hardwareID = device.Parameters[EClassID.eDeviceName].HWID;
```

File names

To get a list of file names currently present in the device, one of the following functions is used.

```
string[] fileNames = DevSetup.SupplementalFiles.GetFilesNamesByDeviceSerialNumber(string deviceSerialNumber);  
string[] fileNames = DevSetup.SupplementalFiles.GetFilesNamesByDeviceHWID(string deviceHWID);
```

Import

The import of additional files (e.g. *.ofa) is done via one of the following functions.

```
DevSetup.SupplementalFiles.ImportByDeviceSerialNumber(string deviceSerialNumber, string configurationFileName);  
DevSetup.SupplementalFiles.ImportByDeviceHWID(string deviceHWID, string configurationFileName);
```

Export

The export of supplemental files (e.g. *.ofa) is done using one of the following functions. It is recommended to check the correct name of the file beforehand using `GetSupplementalFileNames`.

```
DevSetup.SupplementalFiles.ExportByDeviceSerialNumber(string deviceSerialNumber, string supplementalFileName, string targetFileName);  
DevSetup.SupplementalFiles.ExportByDeviceHWID(string deviceHWID, string supplementalFileName, string targetFileName);
```

Delete

The deletion of supplemental files is done using one of the following functions. It is recommended to check the correct name of the file beforehand using `GetSupplementalFileNames`.



No Online FAMOS files should be deleted, otherwise an error message will occur when starting the device.

To "deactivate" the OFA code, the additional file should be replaced by an **empty** OFA code file.

```
DevSetup.SupplementalFiles.DeleteByDeviceSerialNumber(string deviceSerialNumber, string fileName);  
DevSetup.SupplementalFiles.DeleteByDeviceHWID(string deviceHWID, string fileName);
```

3.2.1.5 Misc

Clock type and clock state

```
// Clock type
var clockType = device.Parameters[EClassID.eDeviceClockType].Text;
// Clock state
var clockState = device.Parameters[EClassID.eDeviceClockState].Text;
```

Synchronization

Parameter	ID
Signal input	eSynchSignalInput
Signal output	eSynchSignalOutput
Synchronization master (device)	eSynchMaster
Clock type	eDeviceClockType
Clock state	eDeviceClockState
External synchronization sources	eExternalSynchSignal

You will find the setting values for signal input and output, clock type and state in the ParameterValues class.

Timed start

Parameter	ID
Start option	eStartOption
Start date	eStartDate
Start time	eStartTime
Time offset	eStartTimeOffset
Synchronous start	eSynchStartFlag

You will find the setting values for the start option in the ParameterValues class.

NTP parametes

Parameter	ID	Type
Synchronization NTP Server (1)	eSyncNTPServer1	string
Synchronization NTP Server (2)	eSyncNTPServer2	string
Synchronization waiting time on self start max. [s]	eSyncMaxWaitingTimeOnSelfStart	int
Synchronization NTP deviation time max. [ms]	eSyncNTPMaxDeviation	int
NTP synchronization interval [s] (min;max)	eSyncNTPSyncInterval	string

Example Code:

```
device.Parameters[EClassID.eSyncNTPServer1].SetValue("99.99.99.99");
device.Parameters[EClassID.eSyncNTPServer2].SetValue("11.11.11.111");
device.Parameters[EClassID.eSyncMaxWaitingTimeOnSelfstart].SetValue(1); // int
device.Parameters[EClassID.eSyncNTPMaxDeviation].SetValue(1); // int
device.Parameters[EClassID.eSyncNTPSyncInterval].SetValue("128;1024"); // string
```

Get the fieldbus slot number**Alternative 1 (LINQ):**

```
using System;
using System.Linq;
using System.Windows.Forms;
using imc.Studio.Tools.DevSetupGraphGenerator;

var serialNumber = "123456";
var device = DevSetup.Devices.GetItemBySerialNumber(serialNumber);
var moduleType = ParameterValues.eModuleType.MT_MFB_CANBUS;

var slotID = device.ChildItems
    .Where(child => child.Parameters[EClassID.eModuleType] != null &&
        (int)child.Parameters[EClassID.eModuleType].GetValue() == moduleType)
    .Select(child => child.Parameters[EClassID.eMFBSlotId].Text);

// Gets the first fieldbus slot number
if (slotID != null && slotID.Any()) {
    MessageBox.Show("LINQ: " + slotID.First());
}
```

Alternative 2 (foreach):

```
using System;
using System.Linq;
using System.Windows.Forms;
using imc.Studio.Tools.DevSetupGraphGenerator;

var serialNumber = "123456";
var moduleType = ParameterValues.eModuleType.MT_MFB_CANBUS;
var slot = GetSlotID(moduleType, serialNumber);
System.Windows.Forms.MessageBox.Show("(" + slot);

private string GetSlotID(int eClassID, string serialNumber) {
    string slot = "unkown Slot ID";
    var device = DevSetup.Devices.GetItemBySerialNumber(serialNumber);

    foreach (var para in device.ChildItems) {
        var paraModuleType = para.Parameters[EClassID.eModuleType];

        if (paraModuleType != null) {
            int moduleType = (int)paraModuleType.GetValue();
            if (eClassID == moduleType) {
                slot = para.Parameters[EClassID.eMFBSlotId].Text;
                return slot;
            }
        } // ChannelTypeVal
    } // end of foreach
    return slot;
}
```

3.2.2 Channel functions

Note

- For addressing the parameters, class *EClassID* is used. Possible values of the parameter are available in the *ParameterValues*-Class. See also [Channel parameters](#)⁹⁹.
- The method `ExecuteActions()` is only needed for device and channel actions. For parameters the method `SetValue()` is used.
- The `Channels` class can only be used to access the settings of the channels. The data is available via the `VariableManager` in the `Variables` class, see [Variables functions](#)⁴⁸.

The following line is needed for the following examples. The channel *Channel_001* is used. The channel name is case sensitive.

```
var channel = DevSetup.Channels["Channel_001"];
```

Setting parameters of bits (e.g. comment):

```
var port = DevSetup.Channels["DIN001"];
var bit = port.ChildItems["DIN001_Bit01"];
bit.Parameters[EClassID.eChannelComment].SetValue("MyComment");
```

To use the class *EClassID* the following **library** has to be added:

```
using imc.Studio.Tools.DevSetupGraphGenerator;
```

The parameters are used as follows

```
// EClassID Parameter as string or from EClassID-Class
//var parameter = channel.Parameters["eMyParameter"];
var parameter = channel.Parameters[EClassID.eSampleTime]; // Example
```

Set parameter value

```
parameter.SetValue("SetValue");
```

Get parameter value

```
string parameterValue = parameter.GetValue().ToString();
```

Get printout of the parameter value

```
string parameterText = parameter.Text;
```

Select list of parameter values, if available

```
string [] paraTextArray = parameter.TextList;
```

Metadata columns can be edited via the identifier (here: *MyIdentifier*).

```
var parameter = channel.Parameters["MyIdentifier"];
// get or set a parameter value
parameter.SetValue("MySettingValue");
string parameterText = parameter.GetValue().ToString();
```

For further information, please see the following chapters:

- [Parameter](#)³²
- [Data transfer](#)³⁷
- [Curve window](#)³⁸

3.2.2.1 Parameter

Channel name

```
var deviceParameter = DevSetup.Devices["DeviceName"].Parameters;
var channelParameter = DevSetup.Channels["ChannelName"].Parameters;

// Change Channel name
channelParameter[EClassID.eChannelName].SetValue("Channel_001_new");
```

Balance actions

```
var paraBalanceAction = channel.Parameters[EClassID.eBalanceAction];

// set offset
paraBalanceAction.SetValue(ParameterValues.eBalanceAction.Offset);
paraBalanceAction.ExecuteAction();

// set bridge
paraBalanceAction.SetValue(ParameterValues.eBalanceAction.Bridge);
paraBalanceAction.ExecuteAction();

// set tare
paraBalanceAction.SetValue(ParameterValues.eBalanceAction.Tare);
paraBalanceAction.ExecuteAction();

// set manufacture's calibration
paraBalanceAction.SetValue(ParameterValues.eBalanceAction.Reset);
paraBalanceAction.ExecuteAction();
```

Balance state

```
string balanceState = channel.Parameters[EClassID.eBalanceStatus].Text;
```

Cable compensation

```
var paraCalibAction = channel.Parameters[EClassID.eCalibrationAction];

paraCalibAction.SetValue(ParameterValues.eCalibrationAction.CableCompensation); // Cable
channel.Parameters[EClassID.eCalibrationAction].ExecuteAction();

double cableResistance = (double)channel.Parameters[EClassID.eCableResistor].GetValue();
```

Channel comment

```
channel.Parameters[EClassID.eChannelComment].SetValue("My Channel");
```

Channel name via connector

The channel name of connector "[01] IN01" is queried here. The channel object with its parameters is in the variable *query*.

```
using System.Linq;
string plugInName = "[01] IN01";
var channels = DevSetup.Channels
    .Where(channel => channel.Parameters[EClassID.ePlugInName] != null &&
channel.Parameters[EClassID.ePlugInName].Text == plugInName);

// read channel name
string name = string.Empty;
if (channels != null) {
    name = channels.First().Parameters[EClassID.eChannelName].Text;
}

// write channel name
string name = "MyChannel";
if (channels != null) {
    channels.First().Parameters[EClassID.eChannelName].SetValue(name);
}
```

If there is a problem with the *where* query, please use *foreach* instead.

Channel status



- The channel name must exist and the value to be set must be valid for this type of channel.
- For each active channel, a variable with the same name is created and made available in the `Variables` class after the Process action.

```
// setting status of channel "Channel_1" to "active"
channel.Parameters[EClassID.eStatus].SetValue(ParameterValues.eStatus.Active);

// setting status to "passive"
channel.Parameters[EClassID.eStatus].SetValue(ParameterValues.eStatus.Passive);

// setting status to "write"
channel.Parameters[EClassID.eStatus].SetValue(ParameterValues.eStatus.Write);
// setting status to "read/write"
channel.Parameters[EClassID.eStatus].SetValue(ParameterValues.eStatus.Read_Write);
```

Connector / Enumerated channel number

```
// Connector name
string connectorName = channel.Parameters[EClassID.ePlugInName].Text;

// Enumerated channel number
string channelNumber = channel.Parameters[EClassID.eEnumeratedChannelNumber].Text;
```

Coupling

```
channel.Parameters[EClassID.eCoupling].SetValue(ParameterValues.eCoupling.AC);
channel.Parameters[EClassID.eCoupling].SetValue(ParameterValues.eCoupling.DC);
// See ParameterValue.eCoupling for more setting values
```

Duration

Duration of measurement can be set by the following code whereas the argument is time in seconds.

```
double duration = 60d;
channel.Parameters[EClassID.eDuration].SetValue(0); // undefined
channel.Parameters[EClassID.eDuration].SetValue(duration); // 60 seconds
```

Filtering

The filtering is set to a Butterworth-Bandpass with cutoff-frequencies 10 Hz and 50 Hz.

```
var paraFilterChar = channel.Parameters[EClassID.eFilterCharacteristic];
paraFilterChar.SetValue(ParameterValues.eFilterCharacteristic.Butterworth);

var paraFilterType = channel.Parameters[EClassID.eFilterType];
paraFilterType.SetValue(ParameterValues.eFilterType.Bandpass);

channel.Parameters[EClassID.eFiltCutoff1].SetValue(10); // 10 Hz
channel.Parameters[EClassID.eFiltCutoff2].SetValue(50); // 50 Hz
```

Input range

Input range is set to $\pm 2.5\text{V}$.

```
string str = "\u00b1" + "2.5 V"; // \u00b1 = PlusMinus-Symbol
channel.Parameters[EClassID.eRange].SetValue(str);
```

Input range of channel *Channel_001* is set to a text variable.

```
using imc.Studio.PlugIns.DevSetup.Common;
using imc.Studio.Tools.DevSetupGraphGenerator;

var paraRange = DevSetup.Channels["Channel_001"].Parameters[EClassID.eRange];
string sRange = paraRange.Text; // Display value of range, e.g. "±5 V"
```

Remarks: The only values which can be set are available in the selection list on the Setup page.

A list of possible ranges can be obtained by the following code:

```
var pRange = DevSetup.Channels["Channel_001"].Parameters[EClassID.eRange];
string [] sRange = pRange.TextList;
```

Module type: UNI2-8
Input range
±50 V
±25 V
±10 V
±5 V
±2.5 V
±1 V
±0.5 V
±0.25 V
±0.1 V
±0.05 V
±0.025 V
±0.01 V
±0.005 V

Values for other module types are listed in the corresponding data sheets.

Module type

```
// Return value is language dependent
string sModuleType = channel.Parameters[EClassID.eModuleType].Text;
// -> Bridge, current and voltage amplifier and temperature measurement unit "UNI2-8"
```

Preprocessing

In the following code an arithmetic mean value over 5 sample points is set as preprocessing.

```
var paraProcess = channel.Parameters[EClassID.eProcessing];
paraProcess.SetValue(ParameterValues.eProcessing.ArithMean);
channel.Parameters[EClassID.eProcessingPoints].SetValue(5);
```

Sample time / sample rate

Sample Time is set to 1 ms:

```
// setting Sample Time to 0.001s = 1ms = 1000 Hz
channel.Parameters[EClassID.eSampleTime].SetValue(0.001);
```

Remarks: The only values which can be set are available in the selection list on the Setup page.

A **module-specific** list of possible sample times can be obtained by the following code:

```
var pSampleTime = DevSetup.Channels["Channel_001"].Parameters[EClassID.eSampleTime];
string [] sSampleTimeList = pSampleTime.TextList;
```

Module type: UNI2-8	
Sample rate	Sample time
100 kHz	10 µs
50 kHz	20 µs
10 kHz	100 µs
5 kHz	200 µs
1 kHz	1 ms
500 Hz	2 ms
200 Hz	5 ms
100 Hz	10 ms
50 Hz	20 ms
20 Hz	50 ms
10 Hz	100 ms
5 Hz	200 ms
2 Hz	500 ms
1 Hz	1 s
0.5 Hz	2 s
0.2 Hz	5 s
0.1 Hz	10 s
0.05 Hz	20 s
0.0333333 Hz	30 s
0.0166666 Hz	60 s

Values for other module types are listed in the corresponding data sheets.

Sensitivity/Scale and Offset

The scaling and offset of an channel are set in the following code.

```
channel.Parameters[EClassID.eUserUnit].SetValue("Pa"); // Pascal
channel.Parameters[EClassID.eUserOffset].SetValue(1); // 1 Pa
channel.Parameters[EClassID.eUserScalingFactor].SetValue(5); // = 1/Sensitivity
```

Strain gauge

```

Coupling
channel.Parameters[EClassID.eCoupling].SetValue(ParameterValues.Coupling.HalfBridge;

Bridge resistor
channel.Parameters[EClassID.eBridgeResistor].SetValue(120);

Bridge mode
channel.Parameters[EClassID.eBridgeBridgeMode].SetValue(1);
// 1=>strain; 2=>stress

Bridge factor N
channel.Parameters[EClassID.eBridgeN].SetValue(1);
// 1=>1; 2=>2; 4=>3; (1-v)>4; (1+v)>5; 2*(1-v)>6; 2*(1+v)>7

Gauge factor k
channel.Parameters[EClassID.eBridgeFactor].SetValue(1);

Bridge unit
channel.Parameters[EClassID.eBridgeUnit].SetValue(1);

Transverse strain coefficient v
channel.Parameters[EClassID.eBridgeEps].SetValue(0.3);

Modulus of elasticity
channel.Parameters[EClassID.eBridgeEModule].SetValue(1);

```

TEDs: Read/Write sensor information



Warning

If *ExecuteAction* is used the according parameter/device should be read/assigned again due to internal changing instances. The consequence is a *NullReference* Exception. This could happen when selecting a device or reading TEDs, for instance.

```

// read
var paraReadSensor = channel.Parameters[EClassID.eSensorAction];
paraReadSensor.SetValue(ParameterValues.eSensorAction.ReadSensorFlash);
paraReadSensor.ExecuteAction();

// write
var paraReadSensor = channel.Parameters[EClassID.eSensorAction];
paraReadSensor.SetValue(ParameterValues.eSensorAction.WriteSensorFlash);
paraReadSensor.ExecuteAction();

// delete
var paraReadSensor = channel.Parameters[EClassID.eSensorAction];
paraReadSensor.SetValue(ParameterValues.eSensorAction.ResetChanWithSensorh);
paraReadSensor.ExecuteAction();

```

Two-Point scaling

Set the two-point values as **string**. If they are set as **double**, the scaling factor and offset is considered.

```

channel.Parameters[EClassID.ePoint1SetValue].SetValue("20");
channel.Parameters[EClassID.ePoint1BalanceAction].ExecuteAction();

channel.Parameters[EClassID.ePoint2SetValue].SetValue("50");
channel.Parameters[EClassID.ePoint2BalanceAction].ExecuteAction();

var paraCalibAct = channel.Parameters[EClassID.eCalibrationAction];
paraCalibAct.SetValue(ParameterValues.eCalibrationAction.TwoPoint); // Two Point
channel.Parameters[EClassID.eCalibrationAction].ExecuteAction();

```

Unit



To use the types CValueArray the following reference must be added:

imc.Studio.Plugins.DevSetup.Common

Also add the following using-Directive to the script file: `using imc.Studio.Plugins.DevSetup.Common`

Escaping of quotation marks (") by using backslash (\).

```
// setting User Unit to N/m
channel.Parameters[EClassID.eUserUnit].SetValue("\N/m");

// reading unit
var paraUnit = DevSetup.Channels["Channel_001"].Parameters[EClassID.eUserUnit];
object oUnit = paraUnit.GetValue();
CValueArray cUnit = oUnit as CValueArray;
string sUnit = cUnit[0].ToString(); // unit is the element with index 0
```

Balance at Device start

Set the following parameter to perform a balance action at the device start (Diskstart/Self-start):

```
channel.Parameters[EClassID.eBalanceAtDeviceStart].SetValue(true);
```

3.2.2.2 Data transfer

Save data (Device)

```
channel.Parameters[EClassID.eUMTransferToHD].SetValue(true);
```

Buffer time (RAM)

```
channel.Parameters[EClassID.eBufferTime].SetValue(60d); // 60 seconds; auto = 0
```

Transfer to PC

```
// activate
channel.Parameters[EClassID.eTransferToPC].SetValue(true);
// deactivate
channel.Parameters[EClassID.eTransferToPC].SetValue(false);
```

Save data (PC)

```
// activate saving to Hard Disk
channel.Parameters[EClassID.ePCTransferToHD].SetValue(true);
// deactivate
channel.Parameters[EClassID.ePCTransferToHD].SetValue(false);
```

Display, calculations of measured data

```
// Available Events (Display) ; 0 = all; 1 = last
channel.Parameters[EClassID.eDisplayAvailableEvents].SetValue(0);

// Circular buffer time (Display)
channel.Parameters[EClassID.eDisplayRingTime].SetValue(60d);
```

3.2.2.3 Curve window

Display Time / Circular Buffer Time (Curve Window)

```
channel.Parameters[EClassID.eDisplayRingTime].SetValue(60d);
```

Graph Color

Does not change during measurement.

```
// preset Color
int color = Color.Tomato.ToArgb();
// via RGB
int color = Color.FromArgb(0,255,255).ToArgb();
// set
channel.Parameters[EClassID.eCurveColor].SetValue(color);
```

y-Axis (Scale)

```
channel.Parameters[EClassID.eCurveYAxisOption].SetValue(2);
// Input-Range (1), user-defined (2), auto (3)
channel.Parameters[EClassID.eCurveYAxisMax].SetValue(20d);
channel.Parameters[EClassID.eCurveYAxisMin].SetValue(-20d);
```

3.2.2.4 Examples

Saving all active analog channels to hard disk drive

```

using System.Linq;
using imc.Studio.Tools.DevSetupGraphGenerator;

public override void Run()
{
    DevSetup.Channels
        .Where(channel => IsAnalogChannel(channel) && IsActive(channel))
        .ToList()
        .ForEach(channel => SaveToHardDiskDrive(channel));
}

private bool IsAnalogChannel(EnumItem channel)
{
    bool isAnalog = false;
    var channelTypeParameter = channel.Parameters[EClassID.eChannelType];
    if (channelTypeParameter != null)
    {
        var channelTypeValue = channelTypeParameter.GetValue();
        isAnalog = (int)channelTypeValue == ParameterValues.eChannelType.AnalogInput;
    }

    return isAnalog;
}

private bool IsActive(EnumItem channel)
{
    bool isActive = false;
    var statusParameter = channel.Parameters[EClassID.eStatus];
    if (statusParameter != null)
    {
        var statusValue = statusParameter.GetValue();
        isActive = (int)statusValue == ParameterValues.eStatus.Active;
    }

    return isActive;
}

private void SaveToHardDiskDrive(EnumItem channel)
{
    var saveParameter = channel.Parameters[EClassID.ePCTransferToHD];
    if (saveParameter != null)
    {
        saveParameter.SetValue(true);
    }
}

```

3.2.3 Panel and Widget functions

Here you find the function to access the Panel and the Widget:

- [Panel](#)  40
- [Widget](#)  42
- [Tables](#)  44
- [Curve window](#)  46

3.2.3.1 Panel

Get the active Panel page

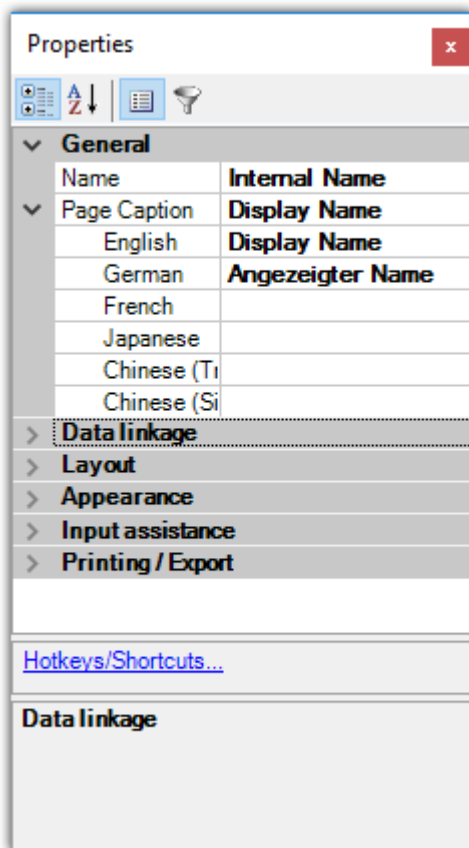
```
PanelPage page = Panel.GetActivePage();
```

Set Panel page with name "Page 1" as active

```
string pageName = "Internal Name";
Panel.SetActivePage(Panel.GetIndexOf(pageName)); // index is needed
```

Note

- For the page name `pageName` the internal name (*Name*) is used and not the displayed name (*Page Caption*) is not used.
- When renaming the *Page Caption* of a side, the *Name* property should also be adapted.



Page properties

Export Panel page

```
using imc.Common.Interfaces.Panel.Scripting;

Panel["Page 1"].Export(eFileFormat.PDF, @"c:\temp\myPage.pdf");
Panel["Page 1"].Export(eFileFormat.BMP, @"c:\temp\myPage.png");
Panel["Page 1"].Export(eFileFormat.DBV, @"c:\temp\myPage.dbv");
```

Import Panel page

```
string filename = @"c:\temp\myPage.dbv";
int pageIndex = 0; // page index, after the page has to be imported
Panel.InsertPageAfter(pageIndex, filename, "Page 1", "Page 1");

string filename = @"c:\temp\myPage.dbv";
int pageIndex = 0; // page index, before the page has to be imported
Panel.InsertPageBefore(pageIndex, filename, "Page 1", "Page 1");
```

Delete Panel page

```
Panel["Page 1"].DeletePage();
```

Change display name of an Panel page

```
// The page's caption (display name) is changed corresponding to the language
Panel["Seite 1"].SetPageCaption("New Name", "en");
```

Rename Panel page

```
// internal name is changed, not the display name
string error = string.Empty;
Panel["Page 1"].SetPageName("Page renamed", out error);
```

Create a Windows.Forms on the Panel page

It is recommended to create this form in a Panel script in the Initialize-method.

```
using System.Windows.Forms;

Control MyForm = new Control();
// ... configure form
Panel["Page 1"].Controls.Add(MyForm); // add form
```

The Windows.Forms are deleted when entering the design-mode. It is recommended to use a **Panel Script**.

Events for "Page added" and "Page deleted"

The Events "PageCreated" and "PageDeleted" can be used to react on adding or deleting a Panel page

These can be used in a **Context script**.

```
// Run
Panel.PageCreated += new Action<object, PanelPageEventArgs>(OnPageCreated);
Panel.PageDeleted += new Action<object, PanelPageEventArgs>(OnPageDeleted);

void OnPageDeleted(object arg1, PanelPageEventArgs arg2)
{
    string text = "New page: " + arg2.PageCaption + " (" + arg2.PageName + ") was delete from position "
+ arg2.Position.ToString() + ".";
    Core.Logbook.LogEntry("CheckPanel", text, 0, ELogbookEntryCategory.Information);
}

void OnPageCreated(object arg1, PanelPageEventArgs arg2)
{
    string text = "New page: " + arg2.PageCaption + " (" + arg2.PageName + ") was added at position " +
arg2.Position.ToString() + ".";
    Core.Logbook.LogEntry("CheckPanel", text, 0, ELogbookEntryCategory.Information);
}

// Dispose
protected override void CheckPanelDispose() {
    // Release all your resources and event hooks here.
    Panel.PageCreated -= OnPageCreated;
    Panel.PageDeleted -= OnPageDeleted
}
```



Example

Adding a button to a Panel page

A button with the text "MyText" is added to a page at a specific position with a given size. An action can be added, when clicking on the button.

```
using System.Windows.Forms;

Button button = new Button();
button.Text = "MyText";
button.Bounds = new Rectangle(100, 50, 100, 50);
// button.Click += new EventHandler(eventFunction);
Panel["Page 1"].Controls.Add(button as Control);
```

Subsequently, the Windows-control can be copied to the script clipboard.

3.2.3.2 Widget

The properties of a Widget can be changed by using `Panel["Page name"]["Widget name"]`. The page name and Widget are case sensitive.

Be sure, that the property is available for the Widget.

Position and size

```
int x = 10; // x-position
int y = 50; // y-position
int h = 100; // height
int w = 200; // width
Panel["Page 1"]["Standard_meter1"].Bounds = new Rectangle(x, y, h, w);
```

Several properties

```
Widget w = Panel["Page 1"]["Standard_meter1"];
//Panel["PageName"]["WidgetName"].PROPERTY = VALUE;

// Range
w.Max = 30;
w.Min = 10;
w.Interval = 10;

// Active
Panel["Page 1"]["Rocker_switch1"].Enabled = true;
// Visibility
w.Visible = true

// Texte setzen
w.Text.SetText("en", "MyText");

// Connect variable to widget
Panel["Page 1"]["Standard_meter1"].Variable.Name = "MyVariable";
```

The first parameter of `SetText()` is a country code for setting texts in different languages.

Standard Widget

To set the text of a Standard Widget the property `StandardText` has to be set. `SetText()` is not available for Standard Widget.

```
Panel["Page 1"]["CheckBox1"].StandardText = "MyText";
Panel["Page 1"]["CheckBox1"].StandardVariable = "MyVariable";
Panel["Page 1"]["Label1"].StandardText = "Hello World!";
```

Create and edit Zones

```
using System.Drawing;
using System.Linq;

var widget = Panel["Page 1"]["List1"];
widget.ZonesController.ZonesClear();
widget.ZonesController.ZoneAdd();
var zone = widget.Zones.Last(); // Alternative: widget.ZoneController.ZoneGet(0);
zone.Text.SetText("en", "Entry_1");
zone.SetValue = 42;
zone.UpperLevel = 100;
zone.LowerLevel = 0;
zone.Color = Color.Blue;
```

Curve window: add Signal

```
// add signal to a new y-axis
Panel["Page 1"]["Curve1"].AddSignal("Channel_001", eAppendConstants.AppendNewAxis);
// add Signal to a new coordinate system
Panel["Page 1"]["Curve1"].AddSignal("Channel_001", eAppendConstants.AppendNewCosy);
// add signal to existing y-axis
Panel["Page 1"]["Curve1"].AddSignal("Channel_001", eAppendConstants.AppendNewLine);
```

Properties of signals can be set via `Panel["Page 1"]["Curve1"].Signals.`

Zones changed (e.g. list)



The event handling methods are run **asynchronously**.

```
public class MyPanel : AbstractMyPanel {  
  
    /// List1 : GaugeWidget  
    public override void Run() {  
        List1.ZoneChanged += new Action<object, EventArgs>(List1_ZoneChanged);  
    } // run  
  
    void List1_ZoneChanged(object arg1, EventArgs arg2) {  
        MessageBox.Show("Hello World!");  
    } // List1_ZoneChanged  
  
    public override void Stop() {  
        List1.ZoneChanged -= List1_ZoneChanged;  
    } // stop  
} // class
```

Events



If errors (exceptions) occur in the registered functions, they are caught during processing and not thrown further.

More information about this can be found in the *ProxyClasses.cs* file.

3.2.3.3 Tables

Link variables to a table cell

The following code links the 1st cell in the 1st column to the variable *Var_01*.

```
using System.Linq;

var table = Panel["Page 1"]["Table1"];
var cell = table.Columns.ElementAt(0).Cells.ElementAt(0); // 1st col, 1st element
cell.Variable.Name = "Var_01";
```

Access to table cells

The folling example sets the text of the first column in the first row to "Hello World".

```
using System.Linq;

var cell = Panel["Page 1"]["Table1"].Columns.ElementAt(0).Cells.ElementAt(0)
cell.Text.SetText("en", "Hello World");
```

Table events

Syntax	Description
OnCellClick	Invoked, when a cell is clicked.
OnCellDisplayValueChangedByUser	Invoked, when the user changes the value of a cell.
OnCellDoubleClick	Invoked, when a cell is double-clicked.
OnColumnHeaderClick	Invoked, when the head of a column is clicked.
OnColoumnHeaderDoubleClick	Invoked, when the head of a column is double-clicked.
OnRowIndicatorClick	Invoked, when the indicator is clicked.
OnRowIndicatorDoubleClick	Invoked, when the indicator is double clicked.
OnSelectionChanged	Invoked, when the selection of the cells has changed.

Table functions

Syntax	Description
Region: Tabelle	
CellClick()	Performs a mouse click to the given cell.
CellDisplayValue()	Gets the display value of the cell.
CellDoubleClick()	Performs a double-click to the given cell.
GetCell()	Gets the cell as an CellWidget object.
GetColumn()	Gets the column as an ColumnWidget object.
SelectionClear()	Clears the current selection.

Table properties

Syntax	Description
Region: Table (TableWidget)	
SelectedCells	Gets the selected cells.
RowCount	Gets the amount of rows.
ColumnCount	Gets the amount of columns.
Region: Columns (ColumnWidget)	
Selected	Returns true, if the column is selected.
Region: Cells (CellWidget)	
Selected	Returns true, if the cell is selected.

3.2.3.4 Curve window

The following line is needed for the following examples.

```
Widget w = Panel["Page 1"]["Curve1"];
```

The following **libraries** are needed for constants and colors:

```
using imc.Common.Interfaces.Panel.Scripting;
using System.Drawing;
```

Add signals

In the constant *eAppendConstants*, the options for adding as a new y-axis or new coordinate system are included.

```
w.AddSignal("Channel_001", eAppendConstants.AppendNewAxis);
w.AddSignal("Channel_002", eAppendConstants.AppendNewLine);
w.AddSignal("Channel_003", eAppendConstants.AppendNewCosy);
```

Display mode

```
w.DisplayMode = eDisplayMode.Stacked;

// Display mode
public enum eDisplayMode {
    Default = 1, Stacked, Waterfall, ColorMap, LastValue, Barmeter, Table, ThreeD
}
```

Loading and saving curve window configurations

```
w.Save(@"c:\Test\MyCurve"); // ccv is appended automatically
w.Load(@"c:\Test\MyCurve"); // ccv is appended automatically
```

Axis settings

```
CurveAxis xAxis = w.xAxis;
CurveAxis yAxis = w.yAxis;

// Scale
xAxis.ScaleType = eAxisScaleConstants.ccwAbsTime; // ccwUnit
// Display range
xAxis.MinMaxType = eAxisMinMaxConstants.ccwFix;
// Minimum / Maximum
xAxis.SetMinMax(-10;10);
double max = xAxis.CurrentMax;
double min = xAxis.CurrentMin;
```

Bar diagram settings

```
// insert reference: imc.Common.Controls.ImcPanel

var myPanel = Panel["panelData_Bar"]; // Panel
var aWidget = myPanel["Curve_T_Bar"]; // Widget
var curveWidget = aWidget.Base as CimcCurveWidget; // Curve Widget
CimcCurveControl myCurveControl = curveWidget.CurveCtrl; // Curve Control !
ICwDataObject myCwDataObject = myCurveControl.DataObjects[0]; // data object
myCwDataObject.BarmeterSettings.Set100PercentRange(-10, 10); // bar settings
myCwDataObject.BarmeterSettings.SetLimits(-5, 5);
```

Signal settings

```
CurveSignalLine line = w.Signals[0];  
  
// Color  
line.Color = Color.Blue;  
// Line shape  
line.Shape = eLineStyle.Line; // Bar, Bar3D, Dots, LineCubic, None, Steps, VertLine  
//  
line.ShowInLegend = eAutoBool.Auto; // Yes , No  
//  
string name = line.SignalName;  
//  
line.width = 3;  
// Delete a signal from the curve window  
line.Delete();
```

Delete all signals

```
foreach (var signal in w.Signals)  
{  
    signal.Delete();  
}
```

3.2.4 Variables functions

Changes



Warning

Fundamental change in access to variables

If you are importing scripts from versions imc STUDIO 4.x or 5.x, please note the following:
The access to variables and measurements has been changed **significantly**.
Manual adjustments to *Scripts* from version 4.x and 5.x are therefore necessary.
The *Datapool* class no longer exists and is replaced by *Variables*.

Scripts from versions imc STUDIO 2025 and later are not affected by this.



Note

Measurements

The *measurements* in the **Data Browser** are treated as *variables*. The *channels* belonging to a *measurement* are the corresponding **children** of this variable.

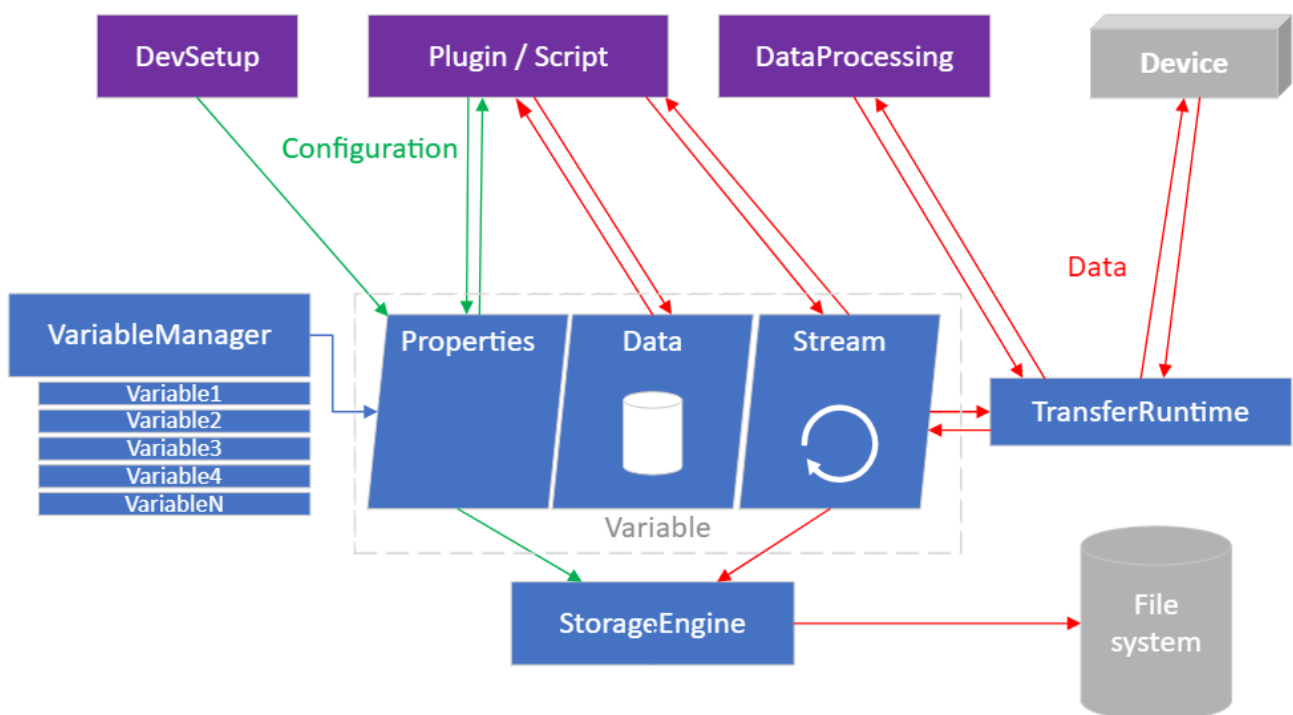


Note

Variables and Context scripts

When using **Context scripts**, note that not all variables are available when the `Run()` method is called.

If variables are to be used immediately, the *VariableCreated* event of the variable should be used in the **Context script**, see [Events](#) ⁶².



Overview to variables access

Overview

The *DevSetup* creates the device *variables* including their properties in the *VariableManager*.

These *variables*, their properties and the *data* can be accessed via the *Scripting* interface (or a plug-in). Likewise *streaming data* can be processed.

The data communication with the *device* itself is done by a *TransferRuntime* component. Data can also be exchanged via *DataProcessing* (including Inline FAMOS Task).

FAMOS Task) kann ebenfalls ein Datenaustausch erfolgen.

The *StorageEngine* saves the variables and their *streaming data* on the *file system*.

Concept

Exceptions must be expected when accessing variables. It is therefore recommended to use the `Try` functions. These return their own `Result` type, which contains the property "IsSuccessful". In case of success the property Value contains the requested value. In case of an error the property Exception contains the corresponding exception.

```
var displayName = new DisplayName("NotExistendVariableName");
var variable = Variables[displayName]; // Exception is thrown

var result = Variables.TryGet(new DisplayName("ExistendVariableName"));
if (result.IsSuccessful)
{
    var variable = result.Value;
    // ...
}
else
{
    // result.ThrowExceptionIfAny();
    Logbook.LogEntry("Script", result.Error);
    // ...
}
```

The different variable types can be found in the corresponding chapter:

- [Variables](#)  50
- [Measurements](#)  54
- [Channels](#)  55
- [Streams](#)  57
- [Complex Variables](#)  61
- [Events](#)  62

3.2.4.1 Variables

Access to variables

```
var variableName = "MeineVariable";
IVariable variable;

variable = Variables[new DisplayName(variableName)];
variable = Variables[variableName];

variable = Variables[new DisplayNamePath(new[] { "Trigger_01", "State" })];
variable = Variables["Trigger_1\\State"];

variable = Variables["meineVariable@Measurement#1"];
variable = Variables["meineVariable@MeineMessung"];

Result<IVariable> result = Variables.TryGet(variableName);
if (result.IsSuccessful) {
    variable = result.Value;
} else {
    // Log(result.Error);
}

result.ThrowExceptionIfAny();
```

Create variables

```
var variable = Variables.Create(new DisplayName("VariableName"));

// Text
var variable = Variables.Create(new DisplayName("TextVariable"));
variable.Properties.Set(EClassID.eDataType, ParameterValues.eDataType.ASCII);
variable.Properties.Set(EClassID.eXFormat, ParameterValues.eXFormat.SingleValue);
variable.Properties.Set(EClassID.eObjectType, typeof(System.String));

// Numeric
var variable = Variables.Create(new DisplayName("NumericVariable"));
variable.Properties.Set(EClassID.eDataType, ParameterValues.eDataType.IEEE754DOUBLE);
variable.Properties.Set(EClassID.eXFormat, ParameterValues.eXFormat.SingleValue);
variable.Properties.Set(EClassID.eObjectType, typeof(System.Double));
```

For creating several variables (or properties, see [Variable properties](#)⁵³) it is recommended to use a **constructor task**.

```
var constructorTask = Variables.Factory.NewConstructorTask();
var variableWriterOne = constructorTask.AddNewVariableConstructor(new
    DisplayName("NumericVariableOne"));
var variableWriterTwo = constructorTask.AddNewVariableConstructor(new
    DisplayName("NumericVariableTwo"));
var variableWriterThree = constructorTask.AddNewVariableConstructor(new
    DisplayName("NumericVariableThree"));
// ... do create properties
constructorTask.Process().ThrowExceptionIfAny();
```

Create variables in variables in variables ...

```
Variables.Create(new DisplayName("A")).Children.Create(new DisplayName("B")).Children.Create(new
    DisplayName("C"));
var name = Variables[new DisplayNamePath(new string[] { "A", "B", "C" })].Name
```

Create variables with scope

- By default, the variables have the scope "volatile" (temporary).
- It is advisable to take care of creating and deleting variables yourself. Only then is it ensured that the appropriate properties are available.
- The following option should therefore be used with caution.



```
using imc.Data.Core.Contract.Identifiers;
using imc.Data.Core.Contract.Tasks;
using imc.Studio.PlugIns.CoreExtensions.UserVariables.Model;
using imc.Studio.Tools.DevSetupGraphGenerator;

var variableName = "TestVariable";
var accessGroup = "UserVariable";

var constructorTask = Variables.Factory.NewConstructorTask();
var variableWriter = constructorTask.AddNewVariableConstructor(new DisplayName(variableName));

variableWriter.Access.SetAccessGroup(accessGroup);

variableWriter.Properties.Set(EClassID.eXFormat, DevSetupCommonValues.eXFormat.SingleValue);
variableWriter.Properties.Set(EClassID.eDataType, DevSetupCommonValues.eDataType.IEEEDOUBLE);

variableWriter.Properties.Set("VariableScope", VariableScope.Experiment.ToString())
    .Attributes.Set(EClassID.eIsPersistent.ToString(), true);

variableWriter.Properties.Set("InitValue", 42);
variableWriter.Properties.Set(EClassIDExtention.eDefaultValue.ToString(), 42);

constructorTask.Process().ThrowExceptionIfAny();
```

VariableScope	
Selection	Description
Volatile	Variable exists only for the running instance of imc STUDIO.
Experiment	Variable exists only in Experiment context.
Sequencer	Variable exists only in Sequencer context.
Project	Variable exists only in Project context.
Persistent	Variable exists only in Application context.

Set the category

```
variable.Properties.Set("eCategory", "CustomCategory");
variable.Properties.Set("eCategory", ""); // User-defined variables
```

Delete variables

```
Variables.Delete(new DisplayName(variableName));
```

Set the value of a variable

```
variable.Value.Set(23d); // numeric
```

The variable must have the properties *eXFormat* and *eDataType*.

Get the variable's content

```
var content = variable.Value.Get<double>(); // whereas double is the desired type
var result = variable.Value.TryGet<double>();
if (result.IsSuccessful)
{
    var content = result.Value;
}
```

If you want to process many variables, use a ReaderTask:

```
List<DisplayName> displayNames = new() {
    new DisplayName("Variable_1"),
    new DisplayName("Variable_2"),
    new DisplayName("Variable_3")
};

List<IDataTaskResult<int>> valueResults = new();

IVariableManagerReaderTask readerTask = variableManager.Factory.NewReaderTask();

foreach (DisplayName displayName in displayNames) {
    IVariableReader reader = readerTask.AddNewVariableReader(displayName);
    valueResults.Add(reader.Data.Get<int>());
}

ProcessValues(readerTask, valueResults);

private void ProcessValues<T>(IVariableManagerReaderTask readerTask, List<IDataTaskResult<T>>
valueResults) {
    var result = readerTask.Process(); // imc.Data.Contract.Result
    if (!result.IsSuccessful) {
        // Error message
        return;
    }

    foreach (IDataTaskResult<T> valueResult in valueResults) {
        if (!valueResult.IsProcessed || !valueResult.IsSuccessful) {
            // Error message
            continue;
        }

        var variableValue = valueResult.Get();
        // do something
    }
}
```

Variable properties

When creating variable properties it is recommended to use a constructor task.

The following (user-defined) properties exist:

```
var constructorTask = Variables.Factory.NewConstructorTask();
var variableWriter = constructorTask.AddNewVariableConstructor(new DisplayName("NumericVariable"));

variableWriter.Properties.Set(EClassID.eCategory, category);
variableWriter.Properties.Set(EClassID.eChannelComment, "");
variableWriter.Properties.Set(EClassID.eDataType, ParameterValues.eDataType.IEEE754DOUBLE);
variableWriter.Properties.Set(EClassID.eXFormat, ParameterValues.eXFormat.SingleValue);
variableWriter.Properties.Set(EClassID.Scope, VariableScope.Experiment);
variableWriter.Properties.Set(EClassID.eObjectType, typeof(System.Double));

// ... further
variableWriter.Properties.Set(EClassID.eCategory, "");
variableWriter.Properties.Set(EClassID.eChannelComment, "");
variableWriter.Properties.Set(EClassID.UserDefinedDataType, UserDefinedDataType.Numeric);
variableWriter.Properties.Set(EClassID.eUserUnit, "");
variableWriter.Properties.Set(EClassID.InitValue, "");
variableWriter.Properties.Set(EClassID.Extention.eDefaultValue, "");

variableWriter.Properties.Set(EClassID.FifoActive, false);
variableWriter.Properties.Set(EClassID.FifoEvents, 0);
variableWriter.Properties.Set(EClassID.FifoRingBufferTime, 0);
variableWriter.Properties.Set(EClassID.StoreActive, false);
variableWriter.Properties.Set(EClassID.StoreEvents, 0);
variableWriter.Properties.Set(EClassID.StoreRingBufferTime, 0);
variableWriter.Properties.Set(EClassID.StoreTriggerName, "");

constructorTask.Process().ThrowExceptionIfAny()
```

EVariableProperty (Extract)	
Property	Description
eCategory	Variable's category
eDisplayColor	Display color in the curve window
eYUnit	y-Unit
eYOffset	y-Offset
eXUnit	x-Unit
eXOffset	x-Offset
eSampleTime	SampleTime
eComment	Comment

Save user-defined properties

If user-defined properties are to be stored on variables or channels, the "IsPersistent" attribute must be set.

```
var constructorTask = Variables.Factory.NewConstructorTask();
var variableWriter = constructorTask.AddNewVariableConstructor(new DisplayName("NumericVariable"));
variableWriter.Properties.Set(EClassID.eXFormat, DevSetupCommonValues.eXFormat.Normal);
variableWriter.Properties.Set(EClassID.eDataType, DevSetupCommonValues.eDataType.IEEE754DOUBLE);

var property = variableWriter.Properties.Set("UserProp_Persistent", "A property which is stored");
property.Attributes.Set(EClassID.eIsPersistent.ToString(), true);

variableWriter.Properties.Set("UserProp_Temporary", "A property which exists only at runtime");

var taskResult = constructorTask.Process();
```

3.2.4.2 Measurements

MeasurementManagerService

```
var measurementManagerService = Variables.Services.Get<IMeasurementManagerService>();
```

Last measurement's name

```
var result = measurementManagerService.FindLastMeasurement();

if (result.IsSuccessful)
{
    var measurementName = result.Value.DisplayName.ToString();
    // ...
}
```

Load measurement

```
Result<IMeasurementService> measurementResult = measurementManagerService.TryGet("FirstMeasurement");
if (measurementResult.IsSuccessful)
{
    var measurement = measurementResult.Value;
    measurement.Load().ThrowExceptionIfAny();
}
```

Unload measurement

```
Result<IMeasurementService> measurementResult = measurementManagerService.TryGet("FirstMeasurement");
if (measurementResult.IsSuccessful)
{
    var measurement = measurementResult.Value;
    measurement.Unload().ThrowExceptionIfAny();
}
```

Delete measurement

```
measurementManagerService.Delete(new DisplayName("SecondMeasurement")).ThrowExceptionIfAny()
```

Rename measurement

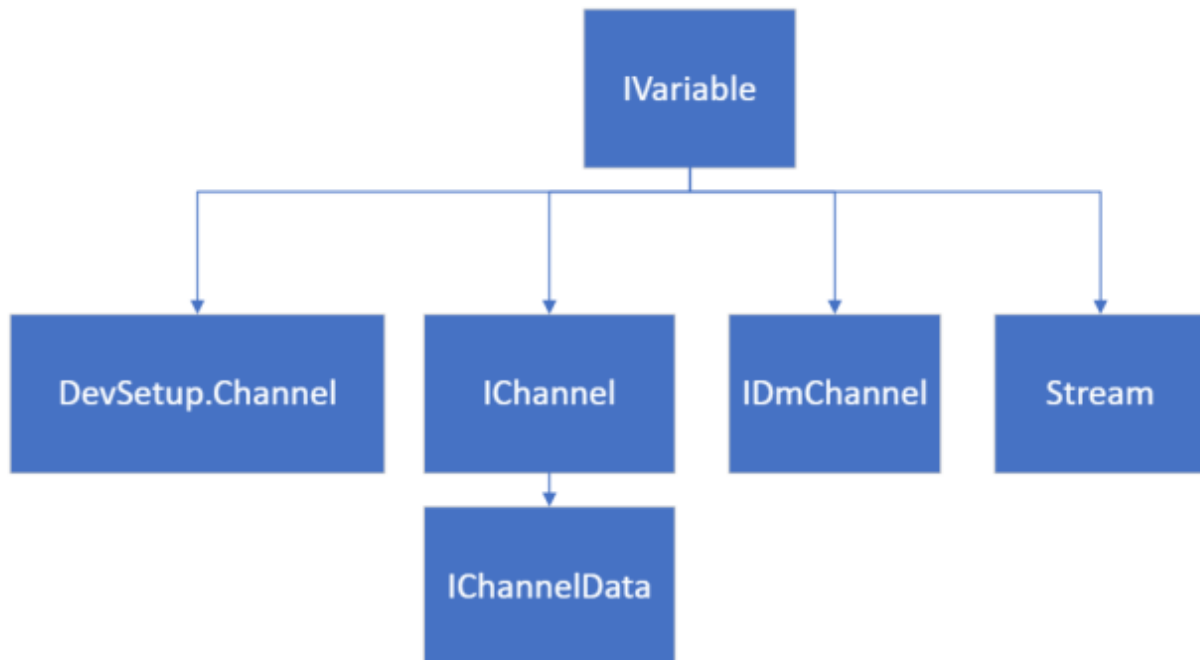
```
Result<IMeasurementService> measurementResult = measurementManagerService.TryGet("FirstMeasurement");
if (measurementResult.IsSuccessful)
{
    var measurement = measurementResult.Value;
    measurement.Rename(new DisplayName("RenamedFirstMeasurement")).ThrowExceptionIfAny();
}
```

3.2.4.3 Channels



For each channel in `DevSetup.Channels` (see: [Channel functions](#))³¹ a variable with the same name is automatically created upon processing the configuration.

```
var channelVariable = Variables["Channel_001"]; // or child from measurement
```



Channel overview

IVariable:	Variables and properties
DevSetup.Channel:	Device-Channel configuration
IChannel:	(saved) channel
IChannelData:	Data of a channel
IDmChannel:	Channel and data
Stream:	Streaming data of a channel

Access to the data of a channel



The measurements must be loaded in the data browser. See [Measurements](#)⁵⁴

Access via IChannel:

```
using imc.Data.Core.Contract.IChannel

var result = variable.QueryInterface<IChannel>();
if (result.IsSuccessful)
{
    var channel = result.Value;
    long n = channel.Data.TotalLength;
    var offset = TimeSpanCore.FromTicks(0);
    var data = channel.Data.Read<double>(offset, n);
    // ..
}
else
{
    Logbook.LogEntry("Script", resultChannel.Error);
}
```

Access via IDmChannel:

```
using imc.Common.Components.DataManager.Interfaces;

public double[] GetData(IVariable variable)
{
    var result = variable.QueryInterface<IDmChannel>();

    if (result.IsSuccessful)
    {
        var channel = result.Value;

        return channel.Data;
    }
    else
    {
        // Error handling
        Logbook.LogEntry("Script", result.Error);
    }
}
```

Usage of IDmChannel/IChannel

The IDmChannel interface can be used arbitrarily for read operations. For write operations it is to be avoided, since it bypasses the management in the VariableManager and therefore inconsistencies can arise.

```
using imc.Common.Components.DataManager.Interfaces;

var result = Variables[variableName].QueryInterface<IDmChannel>();
```

The interface `imc.Data.Storage.HDD.Contract.IDmDataObjectVariableService` is provided as support for exchanging data and properties between **DmChannel** and **IVariable**.

```
var dataObjectService = variable.Services.Get<IDmDataObjectVariableService>();
```

If all data and properties are to be taken from a channel "newDataDmChannel", the following call helps:

```
var configResult = dataObjectService.Configure(newDataDmChannel, DataInclusion.WithAllData);
```

Set data (all existing in the variable will be deleted)

```
dataObjectService.SetData(newDataDmChannel)
```

Data is attached via the IChannelData interface:

```
var channelInterface = variable.QueryInterface<IChannel>().Value;
IChannelData data = channelInterface.Data.QueryInterface<IChannelData>().Value;
data.Append(newDataDmChannel.Data);
```

Usage of IDmGroup/DmGroup

All **DmChannels** passed into a **DmGroup** will be destroyed by it at the end. Therefore the following is forbidden:



```
var dmChannelofVariable = Variables[variableName].QueryInterface<IDmChannel>();
DmGroup.AddObject( dmChannelofVariable );
// ..
DmGroup.Dispose();
```

The only possible current way is to create a clone of the channel beforehand and pass it:

```
group.AddObject(DmChannel.CloneOf(dmChannelofVariable));
```

DmChannel objects, which are the result of Famos operations or were created by yourself, must only be cloned, if you still want to access them after the action with DmGroup.

Set a channel

Create variable

```
using imc.Data.Storage.HDD.Contract;

var service = Variables.Services.Get<IDmDataObjectVariableManagerService>();
var resultVariable = service.Create(resultChannel); // resultChannel of type DmChannel
```

Set variable

The variable must have suitable properties (eDataType, eXFormat, eSampleTime, eObjectTime, ...) before setting.

```
using imc.Data.Storage.HDD.Contract;

var resultVariable = Variables[new Displayname(name)];
var variableService = resultVariable.Services.Get<IDmDataObjectVariableService>();
variableService.SetData(resultChannel);
```

Create and set variable

```
var service = Variables.Services.Get<IDmDataObjectVariableService>();
var resultVariable = service.Configure(resultChannel, DataInclusion.WithAllData);
```

3.2.4.4 Streams

The following basics have to be considered:

There are *cycles*, *events* and the actual *data packages*. A cycle usually corresponds to a measurement, which consists of events, which in turn consist of data packages.

When reading data, it is recommended to use a state machine.

A `StreamReader` is registered to the variable. The `StreamReader` provides a `StreamDataEventReader` via the current event `CurrentEvent`, with which the actual data is read. First it must be checked whether new data are available for reading. If new data is available for **reading** (State 2), it can be read. Otherwise you are in a **wait position** (State 1).

Basic procedure:

1. Attach `StreamReader` via `AttachNewReader()` to `variable.Stream`
2. Register `EventHandler` to `StreamStateChanged`
3. Check in `EventHandler` if new data is present.
 - a. if there is no event, go to the next event
 - i. if there are no more events to read and the cycle is finished -> finish reading the cycle.

- b. if there is an event, but no data to read and the event is not finished -> stay in the waiting state
 - c. if event exists, but there is no data to read and event is finished -> finish reading the event
 - d. if yes, read data
4. If going to next event and this is not successful
 - a. check for `StreamBufferOverflowException` and reset the `StreamReader`.
 5. Detach `EventHandler`
 6. Detach `StreamReader`

**Note****Variables and Context scripts**

When using **Context scripts**, note that not all variables are available when the `Run()` method is called.

If variables are to be used immediately, the `VariableCreated` event of the variable should be used in the **Context script**, see [Events](#)⁶².

Read data streams

The following **using directives** are required,

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

using imc.Data.Contract;
using imc.Data.Contract.Events;
using imc.Data.Core.Contract;
using imc.Data.Core.Contract.Streams;
using imc.Interfaces.Data;
```

To read data streams, a new `StreamReader` is added to the variable and an `EventHandler` is registered to the `StreamChanged` event. The data type (here: `double`) is needed.

```
IStreamReader<double> m_streamReader = variable.Stream.AttachNewReader<double>(); // class member
m_streamReader.StreamStateChanged += OnStreamChanged; // hook
```

A state machine is then used in the **event handler**. In the one state it waits for a new event state, in the other state it processes the data (here of type `double`).

```

// (1) waiting for event state
Result<bool> newEventResult = streamReader.MoveNextEvent();
if (newEventResult.IsSuccessful) {
    if (newEventResult.Value == true) {
        // => move to event state (2)
    }
} else if (newEventResult.Error is StreamBufferOverflowException) {
    // overflow handling
    Result<StreamDataCount> lostData = streamReader.Reset();
    // stay in waiting for event state (1)
}

if (streamReader.IsCycleComplete && streamReader.UnreadEventCount == 0) {
    streamReader.EndReadCycle();
    // stay in waiting for event state (1)
}

// (2) inside event state
IStreamDataEventReader<double> currentEventReader = streamReader.CurrentEvent;
// event header
DateTimeOffsetHighPrecision eventTime = currentEventReader.Time;
DateTimeOffsetHighPrecision referenceTime = currentEventReader.ReferenceTime;
TimeSpanHighPrecision offsetOfFirstSample = currentEventReader.TimeOffsetOfFirstSample;
int eventNumber = currentEventReader.EventNumber;

// current event reader state
var alreadyConsumedDuration = currentEventReader.Duration;
var unreadDuration = currentEventReader.DurationOfUnreadSamples;
long unreadSampleCount = currentEventReader.UnreadSampleCount;
bool isEventComplete = currentEventReader.IsComplete;

Result<double[]> readSamples = currentEventReader.ReadSamples(unreadSampleCount);

if (!readSamples.IsSuccessful && readSamples.Error is StreamBufferOverflowException) {
    // overflow handling
    Result<StreamDataCount> lostData = streamReader.Reset();
    // => move to waiting for event state (1)
}

if (isEventComplete && unreadSampleCount == 0) {
    currentEventReader.EndRead();
    // => move to waiting for event state (1)
}

```

At the end, the `StreamReader` must be unregistered again:

```

m_streamReader?.StreamStateChanged -= OnStreamChanged;
m_streamReader.Detach();
m_streamReader = null;

```



If a Reader is attached later, a `Read()` call should be made first and, if necessary, the overflow should be handled by a `Reset()`.



- If a Reader is attached, it must also read. Otherwise an overflow occurs (`StreamBufferOverflowException`). After the overflow, the `Reset()` method must be called.
 - If changes are made to the variable, it is possible that after Process action the `StreamReader/StreamWriter` is no longer available and an `AggregateException` occurs.
-

Write data streams

To write data streams, a **StreamWriter** is added to the variable. It should be noted that not every variable (e.g. device variables) can be written to.

```

IStreamWriter<float> streamWriter = variable.Stream.OpenWriter<float>();
for (int eventNumber = 1; eventNumber <= 42; eventNumber++) {
    streamWriter.BeginNextDataPacket(
        time: new DateTimeOffsetHighPrecision(DateTimeOffset.Now),
        offset: TimeSpanHighPrecision.Zero,
        eventNumber: 1
    );

    // write data to the stream again and again
    streamWriter.CurrentDataPacket.WriteSamples(
        new[] { 13.5f, 12.1f, 45.8f }
    );

    // optional
    Task readerEndAcknowledgementReaction = streamWriter.CurrentDataPacket.EndWrite();
}
Task readerCloseAcknowledgementReaction = streamWriter.Close();

```

TSA Kanal Handling

```

// ! using imc.Interfaces.Data !
// TimestampSample< UInt48, byte[]>[]
// It MUST be UInt48 (int64 causes a type conversion error in Append()).
// It MUST be byte[] (string returns an error with Append() (type conversion not possible)).
UInt48 myTime = 5;
TimestampSample< UInt48, byte[]>[] testShotNames = new TimestampSample<UInt48, byte[]>[5];
testShotNames[0] = new TimestampSample< UInt48, byte[]>( myTime, Encoding.UTF8.GetBytes( "a" ));
testShotNames[1] = new TimestampSample< UInt48, byte[]>( myTime, Encoding.UTF8.GetBytes( "b" ));
testShotNames[2] = new TimestampSample< UInt48, byte[]>( myTime, Encoding.UTF8.GetBytes( "c" ));
testShotNames[3] = new TimestampSample< UInt48, byte[]>( myTime, Encoding.UTF8.GetBytes( "d" ));
testShotNames[4] = new TimestampSample< UInt48, byte[]>( myTime, Encoding.UTF8.GetBytes( "e" ));

var dpvShotName = CreateTSACHannel( sName);
var chShotName = GetChannel( dpvShotName);
chShotName.Data.Append<TimestampSample<UInt48, byte[]>>( testShotNames);

using imc.Common.Components.DataManager.Interfaces;
using imc.Data.Contract;
using imc.Data.Core.Contract;
using imc.Data.Core.Contract.Identifiers;
using imc.Interfaces.Data;
using imc.Studio.Tools.DevSetupGraphGenerator;

public IVariable CreateTSACHannel( string sName)
{
    var newVar = Variables.Create( new DisplayName( sName));

    newVar.Properties.Set( EClassID.eXFormat, ParameterValues.eXFormat.Timestamp);
    newVar.Properties.Set( EClassID.eDataType, ParameterValues.eDataType.ASCIII);
    newVar.Properties.Set( EClassID.eDataTypeComponent2, ParameterValues.eDataType.UINT48); //??
    newVar.Properties.Set( EClassID.eSampleTime, 1.0);
    newVar.Properties.Set( EClassID.eDisplayAvailableEvents, 0); // 0 = all; 1 = 1.
    newVar.Properties.Set( EClassID.eObjectType, typeof( IChannel));

    return newVar;
}

public IChannel GetChannel( IVariable myVar)
{
    var result = myVar.QueryInterface<IChannel>();
    if (result.IsSuccessful)
        return result.Value;
    else
        return null;
}

```

3.2.4.5 Complex variables

Complex variables are built in a structure and are accessible via the child variables (*children*).

Computer

```
var computerVariable = Variables["Computer"]
var driveVariable = computerVariable.Children["Drives"].Children["C"];
var totalSizeVariable = driveVariable.Children["TotalSize"];
var totalSize = totalSizeVariable.Value.TryGet<double>();
```

Computer\\Drives\\C\\		
Variable name	Internal identifier	Data type
Available free space	AvailableFreeSpace	System.Int64
Total free space	TotalFreeSpace	System.Int64
Total size	TotalSize	System.Int64

Computer\\Process\\		
Variable name	Internal identifier	Data type
Process: GDI handle consumption	GDIHandleConsumption	System.Double
Process: handle count	GDIHandleCount	System.Int32
Process: physical memory consumption	GDIPhysicalMemoryConsumption	System.Int64
Process: private memory consumption	GDIPrivateMemoryConsumption	System.Int64
Process: virtual memory consumption	GDIVirtualMemoryConsumption	System.Int64

SystemClock

```
var systemClockVariable = Variables["SystemClock"];
var systemTimeVariable = systemClockVariable.Children["SYSTEMTIME"];
var systemTime = systemTimeVariable.Value.Get<DateTimeOffset>();
```

SystemClock		
Variable name	Internal identifier	Data type
PC time	PCTIME	System.DateTimeOffset
System time	SYSTEMTIME	System.DateTimeOffset

Trigger

```
var triggerName = "Trigger_01"; // Trigger name, e.g. BaseTrigger
var triggerVariable = Variables[triggerName];
var triggerTime = triggerVariable.Children["TriggerTime"];
```

Trigger_NUMMER Trigger_01, BaseTrigger		
Variable name	Internal identifier	Data type
Zustand	State	System.Double
Eventnummer	EventNumber	System.Double
Trigger Zeit	TriggerTime	System.DateTimeOffset

Device System Variable

```
var variableName = "System_MyDevice_123456"; // SYSTEM_DEVICENAME
var systemDevice = Variables[variableName];
var freeMemory = systemDevice.Children["FreeSpace"];
```

SYSTEM_DEVICENAME SYSTEM_MyDevice_123456		
Variable name	Internal identifier	Data type
Connection 1: Critical channel fill level	WorstFifoLevel	System.Double
Connection 1: Critical channel name	WorstFifoLevelName	System.String
Connection 1: Data rate	DataRate	System.Double
Device: Client count	ClientCount	System.Double
Device: Disk size	DiskSize	System.Double
Device: Free memory	FreeSpace	System.String

SYSTEM_DEVICENAME\Monitor\1 SYSTEM_MyDevice_123456\Monitor\1		
Variable name	Internal identifier	Data type
Connection 2 Monitor 1: Critical channel fill level	WorstFifoLevel	System.Double
Connection 2 Monitor 1: Critical channel name	WorstFifoLevelName	System.String
Connection 2 Monitor 1: Data rate	DataRate	System.Double

Analog for more Monitors.

3.2.4.6 Events

To react to variable events, the EventManager is used.
Since the events must be subscribed and unsubscribed. In imc STUDIO a **Context script** is suitable for this.

```
IEventManager eventManager = Core.Base.GetAPI<API_Core_V2>().Collections["Events"] as IEventManager;
```

Libraries

```
using System.Collections.Generic;
using imc.Common.Interfaces.Logbook;
using imc.Data.Contract.Events;
using imc.Data.Core.Contract.Events;
using imc.Data.Core.Contract.Identifiers;
using imc.Studio.Interfaces.Core;
```

Subscribe event handler

```
eventManager.Subscribe(OnValueChanged, ValueEvent.VALUE_CHANGED);
eventManager.Subscribe(OnVariableCreated, VariableManagerEvent.VARIABLE_CREATED);
eventManager.Subscribe(OnVariableDeleted, VariableManagerEvent.VARIABLE_DELETED);
```

Unsubscribe event handler

```
eventManager.Unsubscribe(OnValueChanged, ValueEvent.VALUE_CHANGED);
eventManager.Unsubscribe(OnVariableCreated, VariableManagerEvent.VARIABLE_CREATED);
eventManager.Unsubscribe(OnVariableDeleted, VariableManagerEvent.VARIABLE_DELETED);
```

Interpret changed events

```
private void OnValueChanged(IEnumerable<IEventData> args)
{
    foreach (var eventArgs in args) {
        string message = Interpret(eventArgs.QueryInterface<IValueChangedEventArgs>().Value);

        if (string.IsNullOrEmpty(message)) {
            continue;
        }

        Log(message);
    }
}

private void Log(string message) {
    Core.Logbook.LogEntry("ListenToValueChanged", message, 0, ELogbookEntryCategory.Information);
}

private string Interpret(IValueChangedEventArgs args) {
    string message = string.Empty;

    // DisplayName displayName = args.Identifier.DisplayName; // DisplayName cannot be used anymore
    // (Performance reasons)
    DataObjectDisplayNamePath displayNamePath = args.Identifier.DisplayPath;
    string variableName = displayNamePath.GetElement(displayNamePath.Depth - 1);

    if (variableName == "MyVariable") // filter
    {
        double newValue = args.Value.Get<double>();
        double oldValue = args.OldValue.Get<double>();

        message = BuildMessage(variableName, oldValue, newValue);
    }

    return message;
}

private string BuildMessage(string variableName, double oldValue, double newValue) {
    return "Variable '" + variableName + "' changed from '" + oldValue + "' to '" + newValue + "'.";
}
```

Interpret created/deleted events

```
private void OnVariableCreated(IEnumerable<IEventData> args)
{
    foreach (var eventArgs in args) {
        var dataObjectEvent = new DataObjectEvent(eventArgs);
        if (dataObjectEvent.DisplayName.ToString() == "Channel_001")
        {
            // do something
        }
    }
}

private void OnVariableCreated(IEnumerable<IEventData> args)
{
    foreach (var eventArgs in args) {
        var dataObjectEvent = new DataObjectEvent(eventArgs);
        if (dataObjectEvent.DisplayName.ToString() == "Channel_001")
        {
            // do something
        }
    }
}

private class DataObjectEvent : AbstractDataObjectEvent
{
    public DataObjectEvent(IEventData eventData) : base(eventData)
    {
    }
}
```

3.2.5 Logbook

Library

```
using imc.Common.Interfaces.Logbook;
// for ELogbookEntryCategory, ILogbookEntryBase, ...
```

Set Logbook entry



Note

For other Script types (Panel, Event, ...) the Logbook is called with `Core.Logbook`.

```
try{
    // ...
} catch (Exception exception){
    Logbook.LogEntry("MyScript.cs", exception);
}

string errorSender = "MyScript.cs";
string errorText = "My Error";
int errorCode = 0;
ELogbookEntryCategory category = ELogbookEntryCategory.Error;
Logbook.LogEntry(errorSender,errorText,errorCode,category);

/*
ELogbookEntryCategory: Error, Fatal, Information, Warning
*/

/*
public void LogEntry(string sender, System.Exception x) ;
public void LogEntry(string sender, System.Exception x,ELogbookEntryCategory category);
public void LogEntry(string sender, string message, int code, ELogbookEntryCategory category) ;
public void LogEntry(ILogbookEntryBase entry) ;
public void LogEntry(ILogbookEntryBase entry, ELogbookEntryCategory category) ;
*/
```

Export Logbook entries since a given date

```
DateTime dateTime = new DateTime(2018, 1, 1);
Logbook.Export(@"c:\temp\logbook.xml", dateTime);
```

Show/Hide Logbook window

```
Core.Windows.ShowWindow("Logbook")
Core.Windows.HideWindow("Logbook")
```

Clear view of the logbook / delete logbook entries in the tool window

```
Logbook.ClearViews();
```

OnError/ OnWarning Event

Use a **Context script** to react to logbook changes.

```
using System.Collections.Specialized;
using System.Windows.Forms;
using imc.Common.Interfaces.Logbook;

// Run()
Core.Logbook.Base.CollectionChanged += new
NotifyCollectionChangedEventHandler(Core_Logbook_Base_CollectionChanged);

// Dispose()
Core.Logbook.Base.CollectionChanged -= Core_Logbook_Base_CollectionChanged;

void Core_Logbook_Base_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
{
    foreach (ILogbookEntryBase entry in e.NewItems)
    {
        LogbookEntry log = LogbookEntry.GetInstance(entry);
        if (log.Category == ELogbookEntryCategory.Warning)
        {
            // do something
        }

        if (log.Category == ELogbookEntryCategory.Error)
        {
            // do something
        }
    }
}
```

3.2.5.1 Examples

Save Logbook entries since start of the script

In this example, all logbook entries which have occurred in the course of the script are written to a text file. For this purpose, at the beginning of the script the time is noted, which is used at the end of the script in order to query the corresponding logbook entries.

```
using System.IO;
using imc.Common.Interfaces.Logbook;

namespace SaveEntries {
    using System;

    public class SaveEntries : AbstractScript {

        public override void Run() {

            DateTime startDate = DateTime.Now; // save Time
            string filename = "x:\\myLogbook.txt";

            // ... body of the script, do something here
            Logbook.LogEntry("MyScript", "My Info", 0, ELogbookEntryCategory.Information);
            Logbook.LogEntry("MyScript", "My Warning", 0, ELogbookEntryCategory.Warning);
            Logbook.LogEntry("MyScript", "My Error", 0, ELogbookEntryCategory.Error);
            // ...

            // entries since the beginning of the script
            ILogbookEntryBase[] entries = Logbook.GetEntries(startDate);

            // save entries to text-file
            using (StreamWriter file = new StreamWriter(filename, true))
            {
                for (int k = 0; k < entries.Length; ++k) {
                    file.WriteLine(entries[k].ToString());
                }
            }

        } // of void run
    } // of class
} // of namespace
```

List of messages with sender "Sequencer"

In this example all messages of the logbook entries with sender "Sequencer" are listed.



In the [Script options](#) ¹¹³ the *advanced code completion* must be enabled.

```
using System.Linq;
using System.Text;
using System.Windows.Forms;
using imc.Common.Interfaces.Logbook;

public void ShowMessages()
{
    string logbookMessages = new StringBuilder();

    foreach (LogbookEntry entry in Logbook) {
        if (string.Compare(e.Sender, "Sequencer") == 0) {
            logbookMessages.AppendLine(e.Message);
        }
    }

    MessageBox.Show(logbookMessages.ToString());
}

public void ShowMessagesAlternative1()
{
    // Alternative
    var query = from entry in Logbook
        where entry.Sender == "Sequencer"
        select entry.GetAPI<API_LogbookEntry_V1>();

    LogbookEntry[] entries = query.ToArray();
}

public void ShowMessagesAlternative2()
{
    // Alternative 2
    var entries = Logbook
        .Where(logEntry => logEntry.Sender == "Sequencer")
        .ToArray();
}
```

3.2.6 Events

The following describes how to access events.



Methods must be registered and unregistered. In imc STUDIO a **Context script** is recommended for this.



Access to user-defined events is currently not possible.

Libraries

```
using imc.Studio.Interfaces.Core;
using imc.Studio.Tools.DevSetupGraphGenerator;
using System.Windows.Forms;
```

Event names

These must be defined within the class.

```
private const string deviceStartedEventName = "Device_Started";
private const string devicesStartedEventName = "Devices_Started";
// further names, see Sequencer
```

Register

Register in Run method

```
Events[deviceStartedEventName].Execute += OnDeviceStarted;
Events[deviceStartedEventName].Execute += OnDevicesStarted;
```

Unregister

Unregister in Dispose method

```
Events[deviceStartedEventName].Execute -= OnDeviceStarted;
Events[deviceStartedEventName].Execute -= OnDevicesStarted;
```

Event handler

```
private void OnDeviceStarted(object sender, ICoreEventEventArgsBase e)
{
    // for every single device
    var api = e.GetAPI<API_CoreEventEventArgs_V1>();
    var parameters = api.Parameters as string[];

    if (parameters != null)
    {
        var hardwareId = parameters[0];
        var device = DevSetup.Devices.GetItemByHWID(hardwareId);
        if (device == null) { return; }
        var serialNumber = device?.Parameters[EClassID.eDeviceSN].Text;
        MessageBox.Show("Device " + serialNumber + " started...");
    }
}

private static void OnDevicesStarted(object sender, ICoreEventEventArgsBase e)
{
    // global start
    var apiCancel = e.GetAPI<API_CancelEventArgs_V1>();
    MessageBox.Show("Devices started...");
}
```

3.2.7 imc FAMOS functions

The signal analysis software **imc FAMOS** is a program for analyzing and evaluating measurement results that provides a wide range of functions for editing, analyzing, visualizing, and documenting measurement data. A large proportion of the functions implemented in **imc FAMOS** can also be used in **imc STUDIO**-scripting. The corresponding functionality can be accessed via the following components (class libraries), which are included in the scope of delivery and must be referenced in your scripting code:

imc.Common.Components.DataObjects.dll	
Data management Definition of the data classes and interfaces used globally in this context. Encapsulation of the imc data manager.	Root: imc.Common.Components.DataManager Key classes include, for example, "DmChannel" (measurement data, numeric values), "DmText/DmTextArray" (imc text variables and text field variables), "DmFile" (file with measurement data), and "DmTime" (imc time format)
imc.Common.Components.Famos.dll	
FAMOS core functionality Editing and analysis of measurement data	Root: imc.Common.Components.Famos The functionality is divided into appropriately named sub-namespaces, e.g. "Basic", "Filter", "Edit", ...
imc.Common.Components.SpectrumAnalysis.dll	
Spectrum Analysis Kit (spectral analysis) Functions for spectral calculations with sliding and overlapping windows and related/derived methods	Root: imc.Common.Components.SpectrumAnalysis The functionality is divided into appropriately named sub-namespaces, e.g. "Spectrum", "Acoustics", ...
imc.Common.Components.ClassCounting.dll	
Class Counting Kit (classification) Functions for classification and rainflow counting	Root: imc.Common.Components.ClassCounting The functionality is divided into appropriately named sub-namespaces, e.g. "RainflowAnalysis", "Counting", ...

imc.Common.Components.OrderTracking.dll	
Order Tracking Kit (order analysis) Functions for order analysis and signature analysis on rotating machines	Root: imc.Common.Components.OrderTracking The functionality is divided into appropriately named sub-namespaces, e.g. "Analysis", "Tacho", ...
imc.Common.Controls.ImcCurves.dll	
CurveWindow Kit (curve window control) Functions for controlling the imc curve window	Root: imc.Common.Controls.ImcCurves Key classes are "CimcCurveWindow" (floating curve window) and "CimcCurveControl" (embedded curve window, e.g. as a panel widget). Curve widgets in the panel can also be configured with this component, see Panel/Curve window ⁴⁶ .

To use the components, they must be referenced in the script project by their file name.

In addition, it is advisable to register the respective root namespace in the script via a corresponding "using" statement, which allows the prefix "imc.Common..." to be omitted in class identifiers and function calls.

The components used here are also part of **imc DATA API**. Detailed help with class/function reference and application examples for the components listed here can be found in the manual for imc DATA API in the application "imc Help and Documentation".

 **License note**

To use these components within **imc STUDIO-Scripting**, an activated **imc FAMOS Runtime** license is required.

 **Compatibility with older versions**

Scripts that were created with older imc STUDIO versions (before V2026) and use imc FAMOS functionality can unfortunately no longer be used without modifications.

A typical error message would be, for example:

*The Script "ScriptName" could not be built. [...]
Could not locate the assembly "imc.Common.Components.FamosKitLibWrapper, Version=5.0.9.6375, Culture=neutral, PublicKeyToken=9350ef35614b8d14".[...]*

The experiment/project must be opened with the Developer edition for this purpose.

For further questions, please contact our [technical support](#) ⁶.

A typical workflow for using these functions is

- * Reading the channels to be processed / calculated from the STUDIO data pool
- * Passing the channels to the desired FAMOS functions
- * Writing the results back to the STUDIO data pool

Read channel

The channels must exist and have the corresponding name.

```
// Read channel "Cosine"
Result<IVariable> inputVar = Variables.TryGet("Cosine");
IDmChannel channel1 = inputVar.Value.QueryInterface<IDmChannel>().Value;
```

Calculate channel

```
// Import types
using imc.Common.Components.DataManager;
using imc.Common.Components.Famos;
```

Basic arithmetic operations, e.g. addition

A reference to `imc.Common.Components.Famos.dll` is required here; see the section ["Adding a Reference to the Project File"](#) ¹⁷³.

```
// Point-by-point addition (FAMOS: + operator)
IDmChannel calcResult = imc.Common.Components.Famos.Basic.Add(channel1, 0, null, double1);
double double1 = 11.1d;
IDmChannel calcResult = imc.Common.Components.Famos.Basic.Add(channel1, 0, null, double1);
```

Integration and differentiation

```
// Integration (FAMOS: int() function)
DmChannel result3 = Basic.Int(channelA);
// Differentiation (FAMOS: Diff() function)
DmChannel result4 = Basic.Diff(channelA);
```

Filtering, e.g. low-pass and smoothing

```
// Low-pass filter (in FAMOS: FiltLP)
// 10th-order Butterworth filter, cutoff frequency 50 Hz
DmChannel filtered1 = Filter.LowPass(channelA, 0, 0, 10, 50);
// Smoothing by weighted averaging (in FAMOS: Smo)
// Smooth over 10 ms
DmChannel filtered2 = Filter.Smooth(channelA, 0.01);
```

Edit, e.g. extract a subset of data

```
// trim data to x-limits (in FAMOS: Cut)
// trim interval between [0.1s...0.2s]
DmChannel part1 = Edit.CopyPartX(channelA, 0.1, 0.2);
// trim data (after Index) (In FAMOS: CutIndex)
// trim from the 5th to the 10th sample in the data set
DmChannel part2 = Edit.CopyPartI(channelA, 5, 10);
```

Regression, backward sampling

```
// linear regression (in FAMOS: LFit)
DmChannel fitLin = Misc.FitLin(channelA);
// backward sampling (in FAMOS: PTast)
DmChannel resampled = Misc.ResampleRef(channelA, channelB);
```

Statistics, e.g. mean, standard deviation

```
// mean (in FAMOS: Mean)
DmChannel meanOfA = Stat.Mean(channelA);
// standard deviation (in FAMOS: StDev)
DmChannel stdevOfA = Stat.StDev(channelA);
```

Search for events

```
// Level search (in FAMOS: SearchLevel)
// e.g. find XY points in the interval [-5;5] with a slope in the range [-1000,1000]
DmChannel xyPoints = Stat.SearchLevel(channelA, 3, -5, 5, 3, -1000, 1000, 0);
// Search for relative maxima above threshold 10 (in FAMOS: xPos)
DmChannel xPosMaximas = Stat.XPosOfMaxima(channelA, 10);
// Zero-point search (in FAMOS: All0)
DmChannel xPosZeroes = Stat.XPosOfZeroes(channelA);
```

Load measurement data from a file and save it to a file

```
// Load channel and save data (in FAMOS: FileLoad, FileSave)
channelA.Load("i:\\dat\\channel1.dat");
DmChannel filtered2 = Filter.Smooth(channelA, 0.01);
filtered2.Name = "channel1_filtered";
filtered2.Save("i:\\dat\\channel1_filtered.dat");

// Load a channel from a multichannel file (in FAMOS: FileOpenDSF, FileObjRead, FileClose)
DmFile file = new DmFile();
file.Open("i:\\dat\\messung.dat", DmFileFlags.Read);
DmChannel level = file.ReadChannel("level");
file.Close();
```

Spectrum Analysis Kit: amplitude spectrum

```
using imc.Common.Components.SpectrumAnalysis;
// An averaged amplitude spectrum (harmonics determined as RMS values) is calculated (in FAMOS:
AmpSpectrumRMS_1)
// Window width 1024, Hamming window
DmChannel myspectrum = Spectrum.AmpSpectrumRMS_1(channelA, 1024, Spectrum.FFTWindow.Hamming, 0,
Spectrum.Averaging.Mean);
```

Class Counting Kit: class crossing method according to DIN 45667

```
using imc.Common.Components.ClassCounting;

// Count class crossings (in FAMOS: ClsLvlCr)
// Range -3 to 3, 12 classes, hysteresis 0.25, edge classes closed
DmChannel distrib = Counting.CLevelCrossing(channelA, 3, -3, 12, 0, 0.25, ClsBorder.Closed, 0);
```

Order Tracking Kit: calculate speed from tachometer signal

```
using imc.Common.Components.OrderTracking;

// Calculate speed signal from tachometer signal (encoder pulses) (in FAMOS: OtrTachoToSpeed, OtrTachoMode)
// Tachometer signal: sampled square-wave signal, 8 teeth
Tacho mytacho = new Tacho(SignalType.RectangledSample, EncoderType.Default, 8, 0);
DmChannel speed = mytacho.TachoToSpeed(channelA);
```

CurveWindow Kit: configure and display curve window

```
using imc.Common.Controls.ImcCurves;

// Create floating curve window, display channel, configure x-axis
// (in FAMOS: CwNewWindow, CwNewChannel, CwAxisSet)
CimcCurveWindow cwnd = new CimcCurveWindow();
cwnd.AppendChannel(channelA, CwAppendConstants.AppendNewLine, 0);
cwnd.xAxes[0].SetMinMax(0, 50);
```

Return result channel

The **channel name** must be set explicitly, because it was changed by the imc FAMOS function.

A reference to `imc.Data.Storage.HDD.Contract.dll` is required here; see the section ["Adding a Reference to the Project File"](#) ¹⁷³.

```
using imc.Data.Storage.HDD.Contract;
calcResult.Name = "NewFamosResult";
IDmDataObjectVariableManagerService service =
Variables.Services.Get<IDmDataObjectVariableManagerService>();
Result<IVariable> result = service.TryCreate(calcResult);
if (!result.IsSuccessful) return;
```

3.2.8 Inline FAMOS functions

- For the functions to be available, the Inline FAMOS needs to be activated in the Product Configurator. This requires a corresponding imc STUDIO Inline FAMOS license.
- If Inline FAMOS is activated later and the functions are added to an existing script, the proxies must be regenerated first. See [Regenerate proxies](#) ¹⁹.
- Please use the return value **IssueResult** of the methods to check, if the action was successful, because you do not get any information in the GUI of imc STUDIO, when configuring IFA-Tasks with Scripting.

Library

```
using imc.Studio.Interfaces.DataProcessing;
```

Get or create IFA-Task

```
InlineFamosTask task = InlineFamos.CreateTask(); // Display name will be generated
// If task available, get it, otherwise create
InlineFamosTask task2 = InlineFamos.GetOrCreateTask("InternalName");
InlineFamosTask task3 = InlineFamos.GetOrCreateTaskByDisplayName("DisplayName");
InlineFamosTask task4 = InlineFamos.GetTaskByName("InternalName");
InlineFamosTask task5 = InlineFamos.GetTaskByDisplayName("DiaplayName");
// Create Task with Owner
InlineFamosTask task5 = InlineFamos.GetOrCreateTask("MyPrivateTask", "Owner", ETaskAccess.Public,
"MyDisplayName");
```

A task created with an *owner* is not saved in the experiment. This allows a dynamically generation of IFA-code. For using the task, the script creating the task needs to be run first.

Create a private task

A **private** task is not shown in the GUI.

```
InlineFamosTask privateTask;
privateTask = InlineFamos.GetOrCreateTask("MyPrivateTask", "Owner", ETaskAccess.Private);
```

Get all public tasks

```
IInlineFamosTaskBase[] tasks = InlineFamos.Tasks;
```

Get all task of an owner

```
IInlineFamosTaskBase[] tasks = InlineFamos.GetTasks("Owner");
```

Delete IFA-Task

```
IssueResult isSuccessful = InlineFamos.DeleteTask(task.Name); // delete
IssueResult isSuccessful = task.Delete();
```

Apply Task

```
IssueResult isSuccessful = task.Apply(); // check result for "IsSuccessful"
```

Task Name

```
string name = task.Name;
IssueResult isSuccessful = task.Setup.SetDisplayName("NewName");
```

Serialize/Deserialize Task

```
task.Setup.GetXml();
task.Setup.ReadXml(stream);
task.Setup.SetXml(xml);
task.Setup.WriteXml(stream);
```

Source code

The source text can be passed directly as a text variable, or a path for the associated IFA-file is specified.

```
// Get the source code
string sourceCode = task.Setup.SourceCode;

// Set source code: Path
IssueResult isSuccessful = null;
string filename = @"c:\temp\MyTask.ifa.xml";
using (StreamReader stream = new StreamReader(filename))
{
    string content = stream.ReadToEnd();
    isSuccessful = task.Setup.SetSourceCode(content);
}

// Set source code: text
task.Setup.SetSourceCode(@"Virt_Chn = Channel_001 * 2");
```



Example

Create task and calculate a channel

A new public task with display name "My IFA Task" is created, which creates the double of an analog channel.

```
InlineFamosTask task = InlineFamos.GetOrCreateTask("MyTask");
task.Setup.SetDisplayName("My IFA Task");
IssueResult resultSource = task.Setup.SetSourceCode("Channel_out = 2 * Channel_001");
IssueResult resultApply = task.Apply();
```

3.2.9 Sequencer Commands in Scripting



Note

In Scripting, all Sequencer commands can generally be executed using the method `Commands.Invoke`. The sequencer command must first be created in imc STUDIO and parameterized within the corresponding dialog. Afterwards, the command is copied via the context menu, which places the associated XML string onto the clipboard.

This XML string will be passed as the argument to `Commands.Invoke`. Specifying a name is optionally.



Example

Delete Panel page *Page 1*

```
Commands.Invoke("Delete page Page: Page 1",
  "<AppPlotFactoryCmd version=\"1\">" +
  "<common version=\"1\" cuid=\"PanelPageDelete\" caption=\"Delete page\">" +
  "<command version=\"1\">" +
  "<properties version=\"1\">" +
  "<DeletePageCommandTemplate>" +
  "<Version>1.0</Version>" +
  "<Page1>Page 1</Page1>" +
  "<Kind>Delete</Kind>" +
  "<ShowExportDialog>False</ShowExportDialog>" +
  "<ShowPrinterDialog>False</ShowPrinterDialog>" +
  "</DeletePageCommandTemplate>" +
  "</properties>" +
  "</command>" +
  "</common>" +
  "</AppPlotFactoryCmd>");
```

Execute a menu action

```
Actions.FireAction(this, "COMMANDNAME"); // for commandos, see table below
```

React to a menu action

With a Context script you can react to the manually execution of a menu action.

```
using imc.Studio.Interfaces.Core;

public override void Run() {
    Actions["COMMANDNAME"].Execute += new
    CommonEventHandler<ICoreActionExecutionEventArgsBase>(Test_Execute);
}

void MyContextScript_Execute(object sender, ICoreActionExecutionEventArgsBase e){
    // do something
}

protected override void MyContextScriptDispose() {
    Actions["COMMANDNAME"].Execute -= MyContextScript_Execute;
}
```

Menu action name (Command name)

The commands are listed in the following table. The menu action names can be shown in the configuration dialog of the command "Execute menu action" under "Additional information".

Example of a menu action name

Sequencer

Action	Command name
start	StartSequencer
stop	StopSequencer

Device actions

Action	Command name
Device configuration (Dialog)	acImcDev30ShowDeviceConfigDlg
open imc Online FAMOS Editor	acImcDev30ShowOFADialog
open Diskstart-Dialog	acDevSetupShowLayout_Diskstart
Device search	acSearchNetwork
connect to selected devices	acDeviceConnect
start selected devices	acDeviceStart
stop selected devices	acDeviceStop
disconnect from selected devices	acDeviceDisconnect
Process	acCheckConfiguration
Prepare	acDeviceDownload
Suspend data storage	acSuspendDiscRecording
Resume data storage	acResumeDiscRecording
Reconfigure	acImcDev30ForceDownload
release all triggers	acReleaseAllTriggers

Panel

Action	Command name
toggle Design mode	cmd_Panel_ToggleDesignMode
toggle Navigation bar	m_biNavBarShow
show Panel	ShowPanel
Panel page in Full-Screen mode	acPanelFullScreen
insert a Standard Panel page	cmd_Panel_InsertPageStandard
toggle lock on Panel page	cmd_Panel_Toggle_Lock
show last Panel page	m_biNavLastPage
show next Panel page	m_biNavNextPage
show previous Panel page	m_biNavPrevPage
show first Panel page	m_biNavHome

Navigation

Action	Command name
absolute time	NavigationActions.SetAbsTime
relative time	NavigationActions.SetRelTime
scroll	NavigationActions.SetXScroll
stretch	NavigationActions.SetXStretch
no scroll	NavigationActions.SetNoScroll

User administration

Action	Command name
log out user	UserLogOut
change user	ChangeUser

File actions

Action	Command name
new experiment (Dialog)	NEW
load experiment (Dialog)	LOAD
save experiment	SAVE
save experiment as	SAVEAS
Import/Export Dialog	IMPORTEXPORTDIALOG

Miscellaneous

Action	Command name
open Help	imcStudioHelpContent
save Current Measurement	imcStudioSaveMeasurement
open Options	imcStudioSharedOptionsDlg
exit imc STUDIO	EXIT
Data (PC)	acImcDev30ShowDataInDevice
Data (Device)	acSetup_ShowDataInPC

3.2.10 Additional functions

3.2.10.1 Product Information

Paths

Path are in the `Environment` class.

```
Core.Environment.ApplicationPath
Core.Environment.InstallationPath
Core.Environment.LogFilePath
Core.Environment.DefaultExperimentDirectory
```

Version information

The version information are in the `Environment` class.

```
Core.Environment.ProductVersionInformation.BuildDate
Core.Environment.ProductVersionInformation.ExtendedVersion; // 2023 R1
Core.Environment.ProductVersionInformation.ProductName
```

3.2.10.2 Default dialog response

In Scripting the options for the default dialog response can be set as follows:

```
using imc.Studio.Interfaces.Gui;
```

Constants for the dialog answers

```
public static class DialogConstants
{
    // Measurement running - reconnect not possible
    public const string StopAndConnect = "DevSetup_StopAndConnect";
    // Overwrite autostart configurations
    public const string OverwriteSelfStart = "Dev2x_OverwriteSelfStart";
    // Overwrite existing file
    public const string OverwriteFile = "VariablesOverwriteFileDialog";
    // Project has changed
    public const string SaveProject = "SaveProjectDocumentUserRequest";
    // Connection failed - transfer configuration
    public const string ConnectionFailed = "DevSetup_ConnectionFailed";
    // Disconnect from running measurement
    public const string DisconnectFromRunningMeasurement = "DevSetup_DisconnectFromRunningMeasurement";
    // No new devices found
    public const string NoNewDevicesFound = "DevSetup_NoNewDevicesFound";
    // Measurement running
    public const string Reconnect = "DevSetup_Reconnect";
    // Devices deselected
    public const string DeselectDevices = "DevSetup_DeselectDevices";
    // Experiment has changed
    public const string SaveExperiment = "SaveExperimentDocumentUserRequest";
    // Channel settings will be reset
    public const string ResetChannelSettingsWarning = "imcDevices2xResetSettingsWarning";
    // Changed supply settings
    public const string ModuleSupplyChangedWarning = "imcDevices2xModuleSupplyChangedWarningDialog";
    // Retrieve device configuration (Retrieve configuration / Stop and Connect)
    public const string RetrieveDeviceConfigurationBeforeReconnect =
    "DevSetup_RetrieveDeviceConfigurationBeforeReconnect";
    // Hardware has changed: (Button1: Discard, Button2: Transfer, None = ShowDialog)
    public const string QuestionDeviceChanged = "imcDevices2xQuestionDeviceChanged";
}
```

Usage

```
UserRequest.SetDefaultResultValue(DialogConstants.StopAndConnect, EMessageBoxResult.OK); // Ok: Stop and
Connect - reconnect not possible

UserRequest.SetDefaultResultValue(DialogConstants.DisconnectFromRunningMeasurement,
EMessageBoxResult.Yes); // Yes: Disconnect, No: Stop and Disconnect

UserRequest.SetDefaultResultValue(DialogConstants.Reconnect, EMessageBoxResult.Yes); // Yes: Reconnect,
No: Stop and Connect

UserRequest.SetDefaultResultValue(DialogConstants.DeselectDevices, EMessageBoxResult.Yes); // Yes:
Discard, No: Transfer

UserRequest.SetDefaultResultValue(ModuleSupplyChangedWarning, EMessageBoxResult.OK);

UserRequest.SetDefaultResultValue(ResetChannelSettingsWarning, EMessageBoxResult.Yes);
// read setting for Overwrite autostart configuration
EMessageBoxResult result = UserRequest.GetDefaultResultValue(OverwriteSelfStart);
// None = Show Dialog
```

EMessageBoxResult	Description
Yes	The dialog will be automatically answered with "Yes".
No	The dialog will be automatically answered with "No".
OK	The dialog will be automatically confirmed with "OK".
Cancel	The dialog will be automatically canceled ("Cancel").
None	The dialog will be displayed
Button1, ..., Button 5	The dialog will be automatically answered with the corresponding button.

Futher information, see chapter Options: Default dialog response.

3.2.10.3 Script clipboard

If objects are to be exchanged between two scripts, the **ScriptClipboard** is available for this purpose.

Objects can be written to the Clipboard for subsequent import to a different script.

The lifetime of the clipboard is linked to the imc STUDIO process.

Copying a new object to the Clipboard (Script A)

```
string text = "My Object";
ScriptClipboard.Add("MyKey", text);
```

Querying an object from the Clipboard (Script B)

```
object o = ScriptClipboard["MyKey"];
string s = o as string;
```

Deleting an object from the Clipboard

```
ScriptClipboard.Remove("MyKey");
```

Testing for existence

```
if (ScriptClipboard.ContainsKey("MyKey")) {
    var value = ScriptClipboard["MyKey"];
}
```

If a device object is to be copied, the corresponding instance must be copied to the Clipboard.

Copying a device object to the Clipboard (Script A)

```
using imc.Studio.Interfaces.DevSetup;

EnumItem dev = DevSetup.Devices.GetItemBySerialNumber("123456");
IEnumItemBase deviceInstance = dev.Base.GetAPI<IEnumItemBase>();
ScriptClipboard.Add("MyDeviceInstance", deviceInstance);
```

Getting a device object from the Clipboard (Script B)

```
using imc.Studio.Interfaces.DevSetup;

object o = ScriptClipboard["MyDeviceInstance"] ;
IEnumItemBase item = o as IEnumItemBase;
EnumItem device = EnumItem.GetInstance(item);
```

Note

- When adding an object to the **script clipboard**, it is necessary to note that this object has a **namespace**. In the case of a Widget, this would be the script's name, e.g. *TestScript.Widget*, due to the proxy-classes.
- When retrieving the object from the script clipboard, *Casting-Exceptions* can occur, since the object type belongs to a different namespace than the script.
- To avoid this, create the script which writes the object to the script clipboard with option "**Separate proxy class library**".
- the script is created with the resource **ProxyClasses**.
- Incorporate the file **ProxyClasses.dll** from this script project into the script project which is to import the object from the clipboard, as a **Resource**.
- When using the type, be sure to specify the namespace correctly.

Reference

For details, see [Using the script clipboard](#) ¹²³

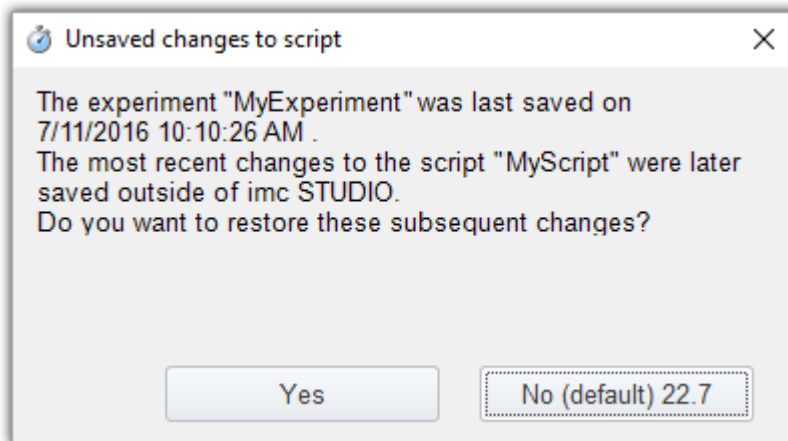
3.2.10.4 Restore script changes

Note

The scripts are not only saved in the **Storage scope** (Experiment, Project, ...), but also in a separate folder. Therefore the saving states can be different.

If modifications are made with the script and the corresponding **Storage scope** is not saved, these **changes** to the script are not saved with the experiment/project.

When reloading, the following dialog will open:



Dialog: Unsaved changes to Script

You have the possibility to restore the changes of the script or to load the current state of the Storage scope (*here: Experiment*).

Yes The last changes of the **Script** are used.

No The current state of the **Storage scope** (*here: Experiment*) is used.

In the [Script options](#) ¹¹³ a default answer can be set for this dialog.

3.2.10.5 Placeholder

For using placeholders, you need to get the *TextPlaceholderResolver* from the SharedComponents.

With the method *ResolveTextAndPLVars()* the placeholder can be resolved.

```
using imc.Common.Interfaces.TextPlaceholder;
using imc.Studio.Interfaces.Core;
using imc.Studio.Interfaces.Core.StandardCollections;

string placeholderToResolve = "<EXPERIMENT.NAME>"
ITextPlaceholderResolverBase placeholderResolver = SharedComponents["ITextPlaceholderResolver"] as
ITextPlaceholderResolverBase;
string experimentName =
placeholderResolver.GetAPI<API_TextPlaceholderResolver_V6>().ResolveTextAndPLVars(this,
placeholderToResolve, false);
```

3.2.10.6 File access (imc-Format)

Read channel from imc file

```
using imc.Common.Components.DataManager;

string filename = @"c:\imc\imc FAMOS\dat\slope.dat";
DmFile file = new DmFile();
file.Open(filename, DmFileFlags.Read);
DmChannel channel = file.ReadChannel("Slope");

file.Close();
file.Dispose();
channel.Dispose();
```

Save channel in imc file

```
using imc.Common.Components.DataManager;

string channelName = "Channel";
DmChannel channel = new DmChannel();
channel.Name = channelName;
channel.Data = new double[]{1, 1, 2, 3, 5, 8, 13};

string filename = @"c:\temp\channel.dat";
DmFile file = new DmFile();
file.Open(filename, DmFileFlags.Write);
file.AddObject(channel.DmHandle);

file.Close();
file.Dispose();
channel.Dispose();
```

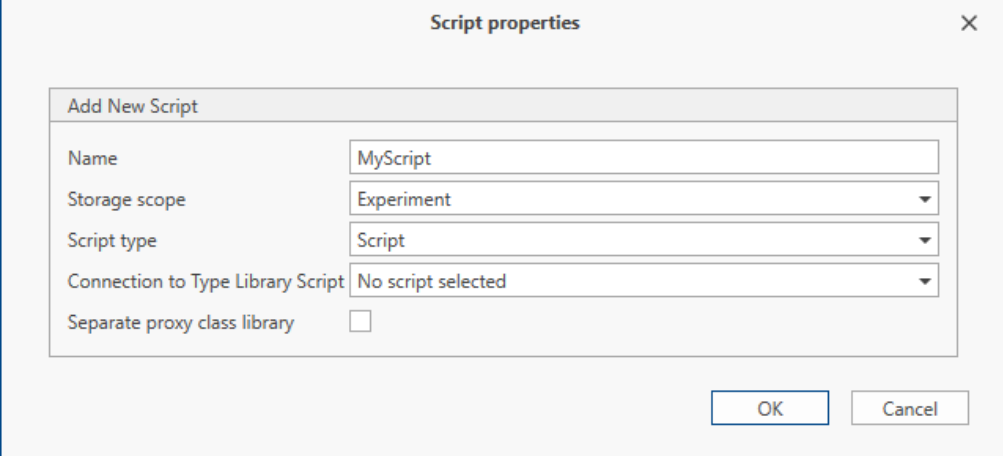
3.3 Script types

3.3.1 Script

Scripts are run in response to:

- the command *Scripts > Run script* in the Sequencer,
- the command *Scripts > Run script* as button events on a Panel page,
- click on *Edit > Run* in the tool window *Scripts*.

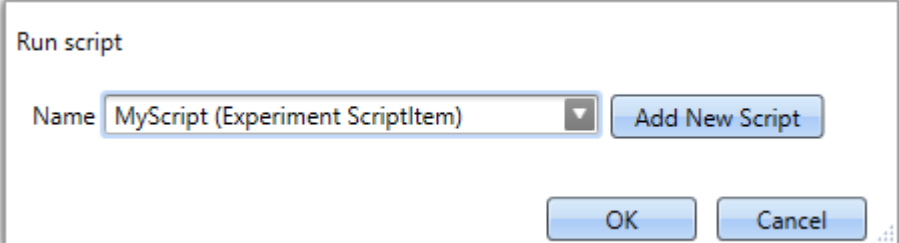
The **Storage Scope** option can be set to Experiment, Project, Application or Sequencer in which the **script** is available.



The 'Script properties' dialog box contains the following fields and controls:

- Add New Script** (Section Header)
- Name**: Text input field containing 'MyScript'.
- Storage scope**: Dropdown menu set to 'Experiment'.
- Script type**: Dropdown menu set to 'Script'.
- Connection to Type Library Script**: Dropdown menu set to 'No script selected'.
- Separate proxy class library**: Unchecked checkbox.
- Buttons**: 'OK' and 'Cancel' buttons at the bottom right.

Dialog: create Script



The 'Run script' dialog box contains the following elements:

- Run script** (Section Header)
- Name**: Dropdown menu showing 'MyScript (Experiment ScriptItem)'.
- Add New Script**: Button next to the Name dropdown.
- Buttons**: 'OK' and 'Cancel' buttons at the bottom.

Command: Run script

Editor

When a new Script is added, the **ScriptEditor** will open with the following code:

```
namespace MyScript {
    using System;

    public class MyScript : AbstractScript {

        public override void Run() {
            // Start your script here
            // Access your variables by using "Variables[variableName]"
            // Access your devices by using "DevSetup.Devices[devicename]"
            // Access your channels by using "DevSetup.Channels[channelname]"
        }
    }
}
```

The code has to be inserted into the method `public override void Run()`.

The entry points for accessing variables, devices and channels are stated in comments.

Helpful **examples** are given in the region *Examples*.

3.3.2 Panel script

Panel Scripts can not be run in response to commands. They are linked with the corresponding **Panel page**.

When creating a **Panel script**, the corresponding Panel page must be selected.

If no page is selected, the script can be linked to a page via its properties.

Dialog: Creating a Panel-script

If a new **Panel Script** is added, the **ScriptEditor** will open with the following code:

```
namespace myScript {
    using System;

    public class myPanel : AbstractmyPanel {

        // Custom ComboBox Example (in Region)
        // Custom ListBox Example (in Region)

        // public override bool PanelScriptInitialize() {}
        // public override bool PanelScriptDispose() {}

        /// Rocker_switch1 : ButtonWidget
        /// Standard_meter1 : PotentiometerWidget
        public override void Run() {
            // Add your initialization code here
        }

        public override void Stop() {
            // Add your cleanup code here
        }
    }
}
```

The Widgets on the Panel page are listed in comments above the function `Run()`. In this example a rocker switch (`Rocker_switch1`) and a gauge Widget (`Standard_meter1`) are on the Panel page.

The finished **Panel Script** must be compiled before being run.

The `Run()` and `stop()` method are executed, when opening or leaving the corresponding Panel-page. 

Method	is executed, when ...	Requirement / Characteristics
Run()	when changing to the corresponding Panel page	an other Panel page is opened
	when loading the experiment	all Panel scripts are executed, which are linked to a Panel page
	when leaving the Design mode	the corresponding Panel page must be opened
Stop()	when changing to an other Panel page	the corresponding Panel page must be opened
	when closing the experiment	the corresponding Panel page must be opened
	when entering the Design mode	the corresponding Panel page must be opened

When changing the main window, **none** of the methods are executed.

Optionally, there are the following override methods available:

Method	is executed, when	Application
PanelScript Initialize()	changing the script ¹ 1: if necessary, the page must be changed or the Design mode must be toggled. loading the experiment and browsing to the Panel page.	To initialize and register events on Widgets or Windows-Forms.
PanelScript Dispose()	closing the experiment before the re-initialization.	To dispose and unregister Events on Widgets or Windows-Forms.

Note

- If events are added on a button action, these events have to be unhooked in the `Stop()` method. Otherwise this action would be performed twice/many times when starting the script again.
- With `this.PanelPage.Name` the current Panel page name can be used.

Widget

The properties of a Widget can be changed by the following line. Be sure, that the property is available for the Widget.

Using class object

```
Standard_meter1.Max = 30;
Standard_meter1.Min = 10;
Standard_meter1.Interval = 10;
```

Access via Panel page

```
Panel["Page 1"]["Standard_meter1"].Max = 30;
Panel["Page 1"]["Standard_meter1"].Min = 10;
Panel["Page 1"]["Standard_meter1"].Interval = 10;
```

Standard-Widget

```
// Change text from Standard-Widgets
Label1.StandardText = "Hello World!";
Label1.TextColor = Color.Aquamarine; // System.Drawing
Button1.StandardText = "Hello World!";
```

For further information, please see chapter [Panel and Widget functions](#) .

Add new Widgets

If new Widgets are added on the Panel page, the following steps are needed in order to use these Widgets in the script:

1. Close the **Editor**
2. Highlight the **Panel Script**
3. Regenerate the *Script Widgets* using the script action in the tool window **Scripts**
4. Edit the **Panel Script**
5. Compile

When deleting Widgets, it is necessary to ensure that the script no longer accesses the widget.

Variables and Logbook

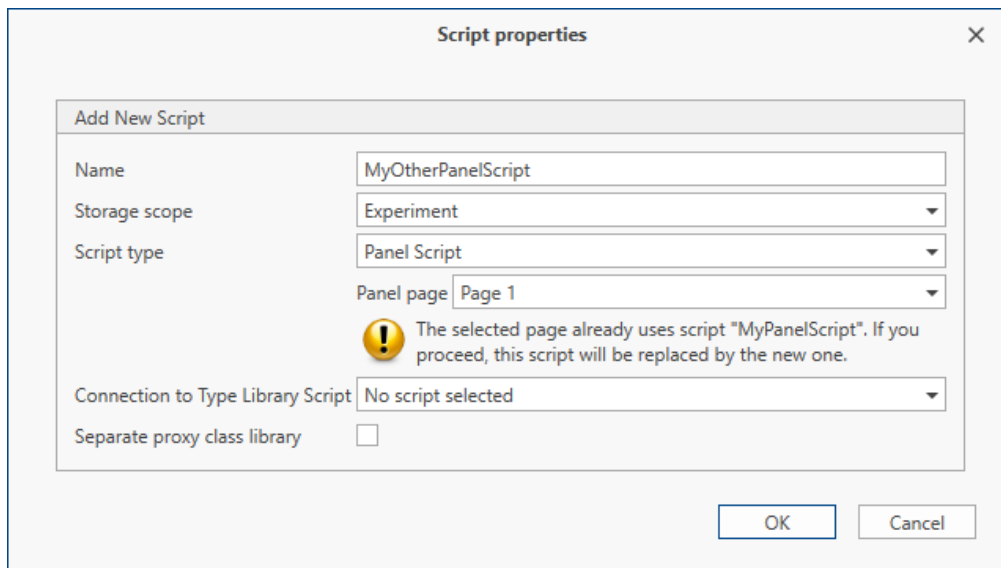
Use class `Core` to access to the variables or logbook.

```
double myVariable = Core.Variables["VariableName"].Value.Get<double>();  
Core.Logbook.LogEntry("MySender", "MyMessage", 0, ELogbookEntryCategory.Warning);
```

Link to Panel page

Warning

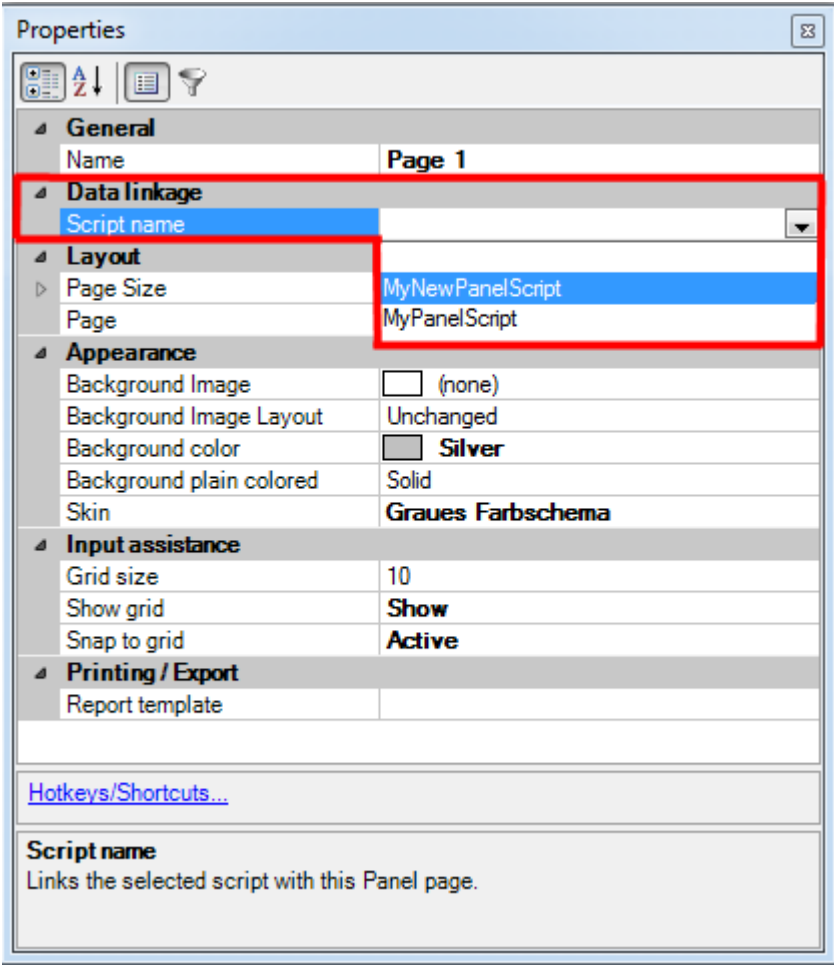
- If a Panel page is already linked with a Panel script, then the linkage is replaced with the new Panel script. The old Panel script is then no longer linked with any Panel page.
- If you import a Panel script, you have to link the script to the corresponding Panel page in the page properties.



The image shows a 'Script properties' dialog box. It has a title bar with a close button. Inside, there's a section 'Add New Script' with several fields: 'Name' (MyOtherPanelScript), 'Storage scope' (Experiment), 'Script type' (Panel Script), and 'Panel page' (Page 1). Below these fields is a yellow warning icon with a text message: 'The selected page already uses script "MyPanelScript". If you proceed, this script will be replaced by the new one.' There's also a 'Connection to Type Library Script' dropdown set to 'No script selected' and a 'Separate proxy class library' checkbox which is unchecked. At the bottom right are 'OK' and 'Cancel' buttons.

Dialog: Panel Script - choose already linked page

In the properties of a Panel page the linked script can be changed.



Page properties: Script name

See [Examples](#) ⁸⁹

3.3.2.1 Examples

Switching visibility of a standard meter by a rocker switch

```
bool isHooked = false;

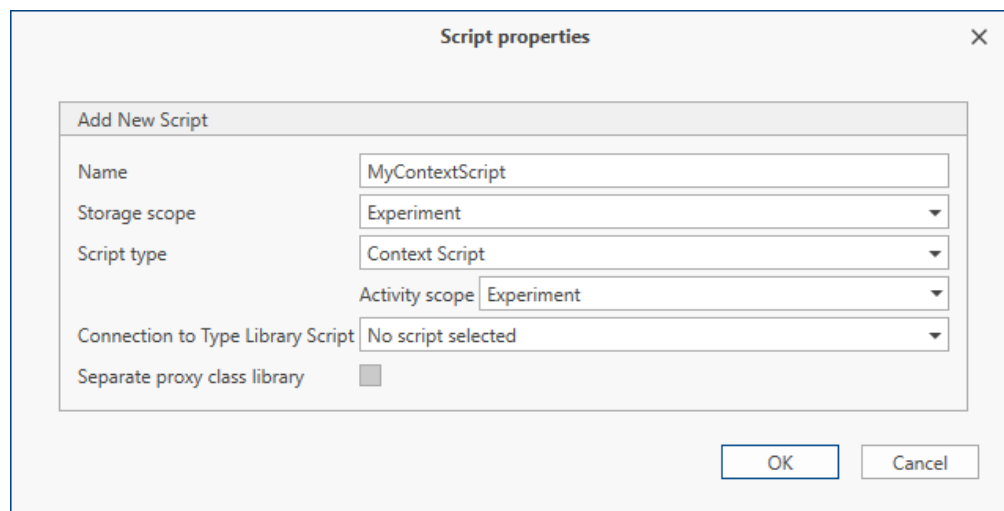
public override void Run() {
    if (!isHooked) {
        isHooked = true; // Status -> angemeldet
        Kippschalter1.SwitchOn += new Action<object, EventArgs>(turnon);
        Kippschalter1.SwitchOff += new Action<object, EventArgs>(turnoff);
    }

    public void turnon(object sender, EventArgs args) {
        if (isHooked) {
            Zeigerinstrument1.Visible = false;
        }
    }

    public void turnoff(object sender, EventArgs args) {
        if (isHooked) {
            Zeigerinstrument1.Visible = true;
        }
    }

    public override void Stop() {
        if (isHooked) {
            isHooked = false; // Status -> abgemeldet
            Kippschalter1.SwitchOn -= turnon;
            Kippschalter1.SwitchOff -= turnoff;
        }
    }
}
```

3.3.3 Context script



Dialog: create Context Script

To add a **Context Script**, the appropriate **Activity Scope** (Experiment, Project, Application, Sequencer) has to be chosen.

The script calls the `Run()` method, when **opening** the corresponding scope. When **leaving** the scope the `Dispose()` method is called.

For example, when opening an experiment containing a **Context Script** with **Activity Scope** *Experiment*, the method `Run()` is called.

Opening other experiments or closing imc STUDIO calls the method `MyContextDispose()`. It is also called, when the script is deleted. The prefix *MyContext* is derived from the script name.

```
namespace MyContext {
    using System;

    public class MyContext : AbstractScript {

        public override void Run() {
            // Start your script here
        }

        protected override void MyContextDispose() {
            // Release all your resources and event hooks here.
        }
    }
}
```



Warning

Since these methods are called while the experiment (or project, ...) is being loaded/closed, it is not permitted to implement any **blocking** functionalities within them.

For example, the display of a message-box would interrupt the loading process. In this case, the system is waiting for the user to provide confirmation to the dialog, and only subsequently can loading of the experiment be completed.

This can cause errors and other unintended effects to occur.



Note

Variables and Context scripts

When using **Context scripts**, note that not all variables are available when the `Run()` method is called.

If variables are to be used immediately, the *VariableCreated* event of the variable should be used in the **Context script**, see [Events](#) ⁶².

React to events

With a Context script you can react to device events. The event names are listed on the Sequencer page.

```
using imc.Common.Interfaces.Miscellaneous;
using imc.Studio.Interfaces.Core;

public override void Run() {
    Events["Devices_Connected"].Execute += new
    CommonEventHandler<ICoreEventEventArgsBase>(MyContextScript_Execute);
}

void MyContextScript_Execute(object sender, ICoreEventEventArgsBase e) {
    // do something
}

protected override void MyContextScriptDispose() {
    Events["Devices_Connected"].Execute -= MyContextScript_Execute;
}
```

Text

With a Context script you can react to the manually execution of a menu action.

```
using imc.Studio.Interfaces.Core;

public override void Run() {
    Actions["COMMANDNAME"].Execute += new
CommonEventHandler<ICoreActionExecutionEventArgsBase>(Test_Execute);
}

void MyContextScript_Execute(object sender, ICoreActionExecutionEventArgsBase e){
    // do something
}

protected override void MyContextScriptDispose() {
    Actions["COMMANDNAME"].Execute -= MyContextScript_Execute;
}
```

For further information, please read chapter [Commands](#) .

See [Examples](#) .

3.3.3.1 Examples

In the following example, a dialog is shown, if the state of a channels has changed. In the method `Run()` the action-method `ValueChanged` is assigned to the event `DevSetup.ValueChanged`. The parameters can be derived from the `DevSetupValueChangedArgs` and be processed.

Within the method `ValueChanged` a `try-finally` and a `if`-condition due to `InValueChanged` should be added.

The script starts, when the experiment is opened (Activity Scope *Experiment*).

Show Dialog, if channel parameter "Status" has changed

```
using System.Windows.Forms;
using imc.Studio.Tools.DevSetupGraphGenerator;

namespace ChannelStateChangedHandler
{
    public class ChannelStateChangedHandler
    {
        private bool isHooked = false;
        private bool InValueChanged = false;

        public override void Run()
        {
            if (!isHooked)
            {
                isHooked = true;
                DevSetup.ValueChanged += new Action<object, DevSetupValueChangedArgs>(ValueChanged);
            }

            private void ValueChanged(object arg1, DevSetupValueChangedArgs arg2)
            {
                foreach (var item in arg2.Changes.Keys)
                {
                    if (item.ClassID == EClassID.eStatus.ToString())
                    {
                        var channel = DevSetup.Channels.GetItemByHWID(item.HWID);
                        if (channel != null && channel.Base != null)
                        {
                            var channelNameItem = channel.Parameters[EClassID.eChannelName];
                            if (channelNameItem != null)
                            {
                                string strChannelName = channelNameItem.GetValue() as string;
                                MessageBox.Show(strChannelName + "=" + item.Text, "State changed");
                            }
                        }
                    }
                }
            } // for each
        } // of void

        protected override void ParameterChangedDispose()
        {
            // Release all your resources and event hooks here.
            if (isHooked)
            {
                DevSetup.ValueChanged -= ValueChanged;
                isHooked = false;
            }
        } // of class
    } // of namespace
}
```

3.3.4 Type library script

The image shows a 'Script properties' dialog box with a close button (X) in the top right corner. Inside the dialog, there is a section titled 'Add New Script'. This section contains three input fields: 'Name' with the text 'MyLibraryScript', 'Storage scope' with a dropdown menu showing 'Experiment', and 'Script type' with a dropdown menu showing 'Type Library Script'. Below these fields is a descriptive text: 'Enables you to develop classes, interfaces etc. which are usable within other scripts.' At the bottom of this section is a checkbox labeled 'Separate proxy class library' which is currently unchecked. At the bottom right of the dialog are two buttons: 'OK' and 'Cancel'.

Dialog: create Type Library Script

In the **Type-Library-Script**, it is possible to enter one's own classes and methods which can then be embedded into the other script types where they are subsequently available for use. A **Type-Library-Script** can also be used in multiple other scripts. For this reason, this kind of script is especially useful when certain classes and methods are to be used repeatedly. Thus, the process of embedding these into each script separately is not necessary.

You can implement your own classes and methods in a **Type Library Script**.

A **Type Library Script** can be connected to other script types in the *Script options* dialog, when creating scripts.

```
using System;
namespace MyLibrary {
    // Insert your TypeLibrary code here
}
```



Warning

- Please make sure when using **Type Library Scripts**, that the *Storage Scope* is chosen so that the **Type Library Script** is accessible from the linked scripts, even if you change the *Storage Scope*.
- After making changes to a linked type library script, you must either update the linked script or the type library script using the "Run" function, or restart imc STUDIO for the changes to take effect.



Note

It is recommended to set the **Assembly Version** (see file *AssemblyInfo.cs*):

```
[assembly: AssemblyVersion("1.0.1.0")]
```

See [Examples](#) 

3.3.4.1 Examples

Type Library Script

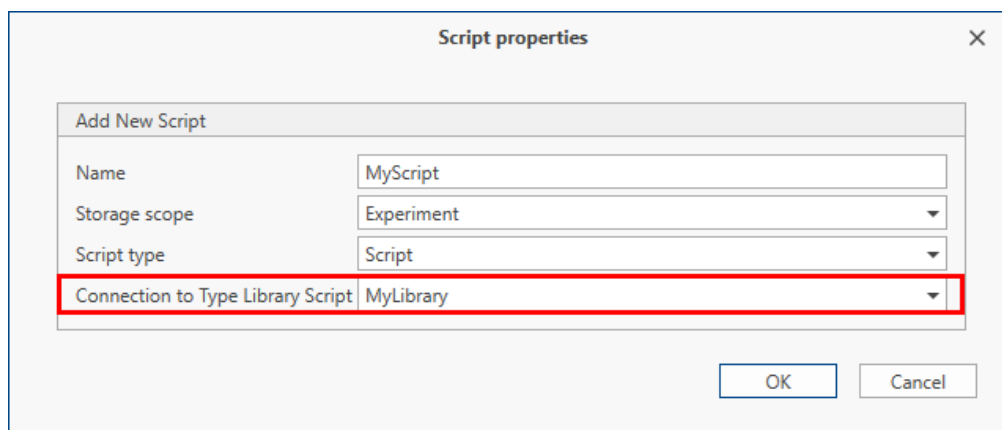
```
using System;
namespace MyLibrary {
    // Insert your TypeLibrary code here
    public class MyClass
    {
        private double dNumber = 12;
        public string ClassName = "MyClass";

        public void SetNumber(double d) {
            this.dNumber = d;
        }

        public double GetNumber() {
            return this.dNumber;
        }
    } // end of class MyClass
}
```

Script

The actual script is created with the connection to the Type Library Script *MyLibrary*.



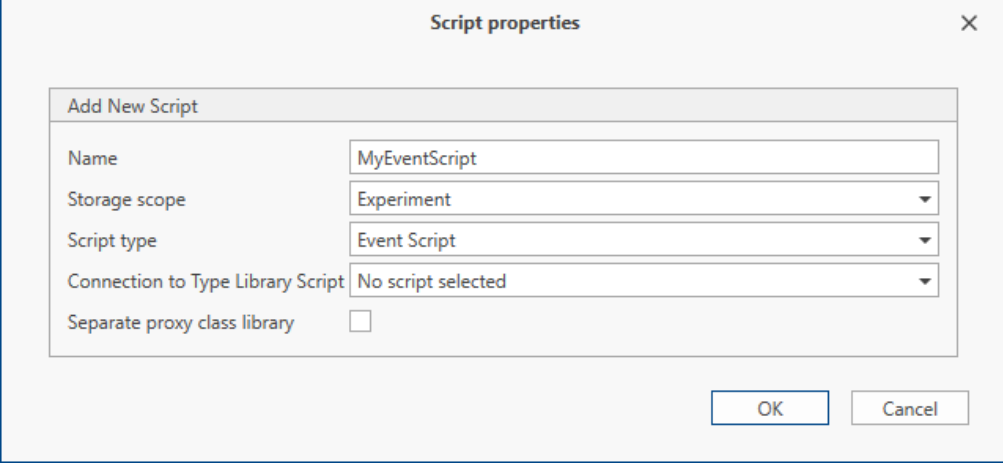
Script with Type Library Script

The class and methods of the library can be used in the script.

```
namespace MyScript {
    using System;
    using MyLibrary;

    public class MyScript : AbstractScript
    {
        public override void Run()
        {
            MyClass a = new MyClass();
            a.SetNumber(8);
            double dNum = a.GetNumber();
            string name = a.ClassName;
        }
    }
}
```

3.3.5 Event script



The image shows a 'Script properties' dialog box with a close button (X) in the top right corner. Inside the dialog, there is a section titled 'Add New Script'. Below this title, there are five fields: 'Name' with the text 'MyEventScript', 'Storage scope' with a dropdown menu showing 'Experiment', 'Script type' with a dropdown menu showing 'Event Script', 'Connection to Type Library Script' with a dropdown menu showing 'No script selected', and 'Separate proxy class library' with an unchecked checkbox. At the bottom right of the dialog, there are two buttons: 'OK' and 'Cancel'.

Dialog: Create Event script

To use an **Event script**, add the Run script command to an event in the Sequencer.

Within the script, the event arguments can be accessed.

```
namespace MyEventScript
{
    public class MyEventScript : AbstractScript
    {
        public override void Run()
        {
        }
    }
}
```

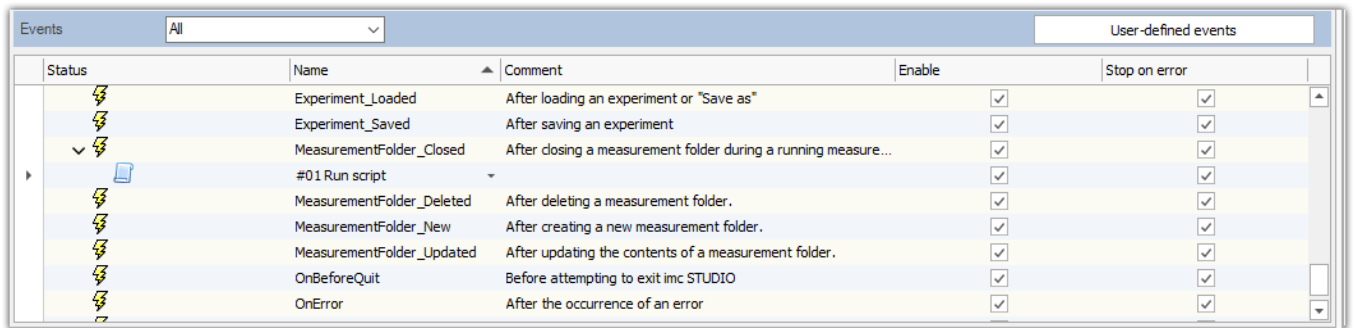
3.3.5.1 Measurement events

In the Sequencer the following measurement events are available:

Event name	Description: Fired when
MeasurementFolder_Closed	...when a measurement folder has been completely closed, e.g. after a measurement.
MeasurementFolder_Deleted	... when a measurement folder has been deleted
MeasurementFolder_New	... when a measurement folder is created
MeasurementFolder_Updated	... when new data has been added to the measurement.

The former event "Storage_DirectoryUpdate" is transferable to the event "MeasurementFolder_Closed".

The following EventArgs are available in the Event script: **this.MeasurementFolderEventArgs**



Sequencer: Measurement events

```
using System.IO;
using System.Windows.Forms;

namespace MeinEvent
{
    public class MeinEvent : AbstractScript
    {
        public override void Run()
        {
            var args = this.MeasurementFolderEventArgs;

            if (args != null)
            {
                MessageBox.Show(args.DisplayName.ToString());
            }
            else
            {
                MessageBox.Show("null");
            }
        } // Run
    }
}
```

3.3.6 Third party device script

Reference

For more information see chapter [Third Party Device Script](#)

3.4 EClassID - Parameters

3.4.1 Device parameters

Note

For setting the parameters, class *EClassID* is used. Setting values of the parameter are available in the *ParameterValues*-Class. The `EClassID` device parameters are listed below.

To use the class *EClassID* the following **library** has to be added:

```
using imc.Studio.Tools.DevSetupGraphGenerator;
```

Please click on the parameters to see the corresponding tables.

Device Control (Action)

eDeviceControlAction	
Connect	1000
Prepare	2000
Start	3000
Stop	5000
Disconnect	7000

`ExecuteAction()` has to be run after setting the value.

Diskstart (Action)

eDiskStartAction	
PC + Device HD*	2501
PC HD	2502
Device HD	2503
Removable Device	2504
Device intern	2505
Device intern RAM*	2506
All Device mem space*	2507

ExecuteAction() has to be run after setting the value.

* Functions not implemented.

eDiskStartOption	
immediately	1
defined time	2
next second	3
next minute	4
next hour	5
next day	6
automatically	7
next 10th minute	8

Synch Signal: Input/Output

eSynchSignalInput eSynchSignalOutput	
no signal	1
DCF	2
GPS	3
SYNC + GPS	4
Synch. signal NTP	5
Synch. Ethercat	6
PTP	7
IRIG B002W	8

Start option

eStartOption	
immediately	1
defined time	2
next second	3
next minute	4
next hour	5
next day	6
automatically	7
next 10th minute	8

Connection state

eConnectionStatus	
disconnected	0
connected	1

Measurement state

eMeasurementStatus	
stopped	1
running	2
reconfigured	3

3.4.2 Channel parameters

Note

For setting the parameters, class *EClassID* is used. Setting values of the parameter are available in the *ParameterValues*-Class. The *EClassID* channel parameters are listed below.

To use the class *EClassID* the following **library** has to be added:

```
using imc.Studio.Tools.DevSetupGraphGenerator;
```

Please click on the parameters to see the corresponding tables.

Status

eStatus	
active	1
passive	2
write	1001
read/write	1002

Channel Type

eChannelType	
Analog Input	1
DAC-Output	2
Digital inputs/outputs (ports)	3
Trigger	4
Virtual Channels	5
Digital inputs/outputs (bits)	6
Virtual bits	7
Net Bits	8
Counter inputs	9
Temperature reference Channel	10

eChannelType	
Monitor: Analog inputs	11
Monitor: Digital inputs / outputs (ports)	12
Monitor: Counter inputs	13
Monitor: Digital inputs / outputs (bits)	14
Display Variables	15

eChannelType	
Fieldbus: Analog inputs	17
Fieldbus: Digital input / output (ports)	18
Fieldbus: Digital input / output (bits)	19
Fieldbus: Sum event	20
Fieldbus: Sending Channels	21

eChannelType	
Synthesizer	22
Communication Object	23
Process-Vector Variables	24
Video	1001
Monitor: Video	1002
Digital inputs/outputs (ports)	4099
Digital inputs/outputs (ports)	8195

Balance (Action)

eBalanceAction	
Offset	2011
Bridge	2012
Tare	2013
Manufacture's calibration	2014

`ExecuteAction()` has to be run, after setting the value.

Balance Status

eBalanceStatus	
balance state undefined	0
not balanced	1
bridge balance run	2
bridge balance ok	3
tare run	4
tare ok	5
offset run	6
offset ok	7
manufacture run	8
manufacture ok	9

Cable Calibration (Action)

eCalibrationAction	
Two-Point Scaling	2021
Invalid	2022
Cable Compensation	2023
Reset Cable	2024
Reset Two-Point	2025

`ExecuteAction()` has to be run, after setting the value.

Calibration status

eCalibrationStatus	
undefined	0
not balanced	1
Point 1 running	10
Point 1 ok	11
Point 2 running	12
Point 2 ok	13
Point 1 and 2 ok	14
Two-Point scaling running	15
Two-Point scaling ok	16
Two-Point reset running	17
Two-Point reset ok	18
Cable compensation running	19
Cable compensation ok	20
Cable compensation reset running	21
Cable compensation reset ok	22

Filter Type

eFilterType		
Low Pass		1
High Pass		2
Band Pass		3
without		4
AAF		5
Slow		6
Fast		7
Impulse		8
Peak		9

Filter Characteristics

eFilterCharacteristic		
Butterworth		1
Bessel		2
Tschebbychew		3
Cauer		4
A-weighting		5
B-weighting		6
C-weighting		7
D-weighting		8
without weighting		9

Event

eEvent	
Signal = 1	381
Signal = 0	384
Signal change 1 -> 0	383
Signal change 0 -> 1	384
Positive slope	391
Negative slope	392
Signal > Threshold	393
Signal < Threshold	394
Signal entering range	401
Signal exiting range	402
Inside of range	403
Outside of range	404

Event-Type

eEventType	
Threshold	1
Range	2
Ratio	3

Coupling

eCoupling	
AC	1
DC	2
Half-Bridge	3
IEPE	4
Full-Bridge	5
Quarter-Bridge	6
Microphone	7
DC charge	8
AC charge	9
TTL	10

eCoupling	
Strain gauge - quarter-bridge	1001
Poisson half bridge	1002
Half bridge with 2 active strain gauges in uniaxial direction	1003
Half bridge with one active and one passive strain gauge	1004
Full bridge with Poisson strain gauges in opposed branches	1005
Full bridge with Poisson strain gauges in adjacent branches	1006
Full bridge with 4 active arms	1007
General strain gauge - full-bridge	1008
General strain gauge - half-bridge	1009

Channel mode

eChannelMode	
Sampling / Distance (diff.)	1
Bit Output / Angle (diff.)	2
Bit Input / Speed	3
RPM	4
Events	5
Frequency	6
Time	7
Pulse time	8
Angle(abs)	9
Distance (abs)	10
Voltage	1000
Current	1001
Strain gauge	1002
Charge	1003

Preprocessing

eProcessing	
none	1
Arithmetic mean value	2
Minimum	3
Maximum	4
RMS	5
MinMax	6
Reduction	7

3.4.3 Experiment parameters



Note

The *EClassID* parameters for the **experiment** are set as *string* listed in the following table.



Example

```
DevSetup.Experiment["CompanyName"].SetValue("imc");
DevSetup.Experiment["TestEngineer"].SetValue("Average Joe");
```

Project

Description	EClassID parameter
Company name	CompanyName
Hint	Hint
Project document	ProjectDoc
Linked project document	LinkedProjectDoc
Project documentation path	ProjectDocPath
Project officer	ProjectOfficer
E-mail address project officer	EMailProjectOfficer

Test engineer, test setup, test object, test operator

Description	EClassID parameter
Test engineer	TestEngineer
Test engineer e-mail	EMailTestEngineer
Test setup description	TestObjectDesc
Test station photo	TestObjPic
Test setup document	TestObjectDoc
Test object description	TestSetupDesc
Test object picture	TestStationPhoto
Test object document	TestSetupDoc
User	User
E-mail address user	EMailUser

Annotation

Start: before measurement

End: after measurement

Description	EClassID parameter
Comment (start)	AnnotationStart
Environmental conditions (start)	ConditionsStart
Conditions parameter name (start)	ConditionsParamNameStart
Conditions parameter value (start)	ConditionsParamValueStart
Picture (start)	PicStart
Location	Location
Date	Date
Time	Time
Comment (end)	AnnotationEnd
Environmental conditions (end)end)	ConditionsEnd
Conditions parameter name (end)	ConditionsParamNameEnd
Conditions parameter value (end)	ConditionsParamValueEnd
Picture (end)	PicEnd

Result

Description	EClassID parameter
Result status (name)	ResultStatusName
Result status (value)	ResultStatusValue
Result characteristic name	ResultCharacName
Result characteristic	ResultCharac
Result description	ResultDesc
Result document	ResultDoc
Result picture	ResultPic

Misc

Description	EClassID parameter
Test part number	TestPartNo
Test object number	TestObjNo
Short Description	DescriptionShort
Description	DescriptionLong

3.4.4 Finding parameters and setting values

Finding parameters and their values

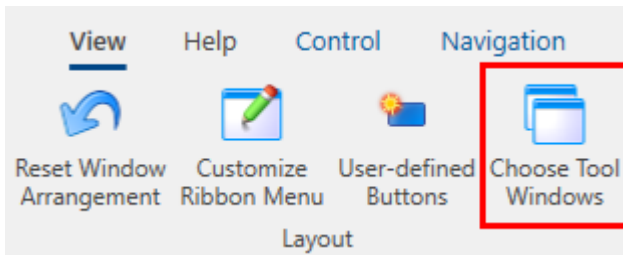


You can find a look-up table of the **EClassID parameters** in the document *imc_STUDIO-EClassID-Parameters.pdf* in the **installation package** under `\Products\Development\Documentation\en\`

The following describes an alternatively way to find the **EClassID** parameters on the basis of the table description. Another way is described in [Finding setting values](#) ¹⁰⁸.

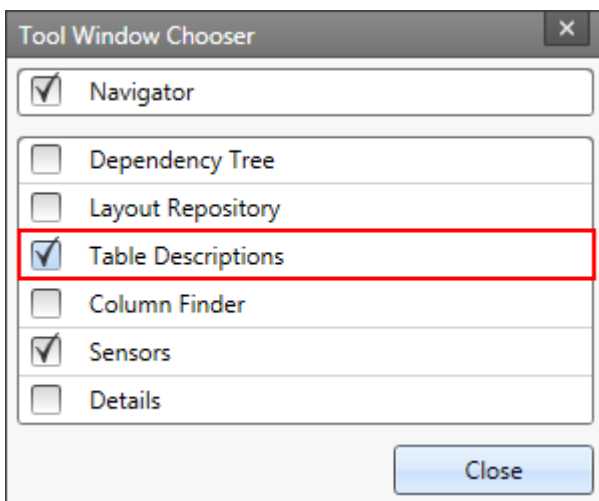
Activating the Table Description tool window

Start by activating the tool window *Table description* on the **Setup** page. Once you are on the Setup page, click in the menu ribbon on *View > Layout > Tool Windows*.



View > Layout > Tool Windows

A new dialog will open. Put a check in the box for *Table Descriptions* and close the window. A new tab is added at the right edge of the screen.

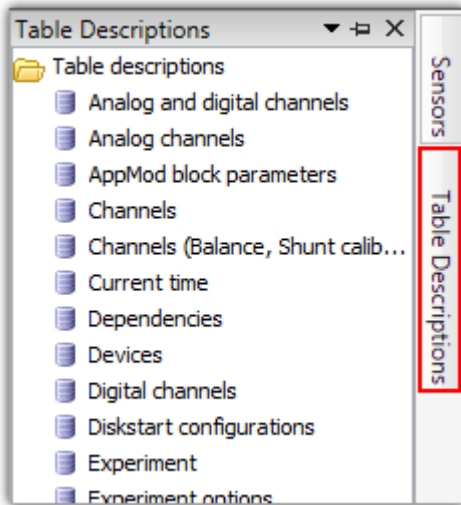


Dialog: Tool Window Chooser

The view can also be saved, so that the tool window is available once again the next time imc STUDIO is started.

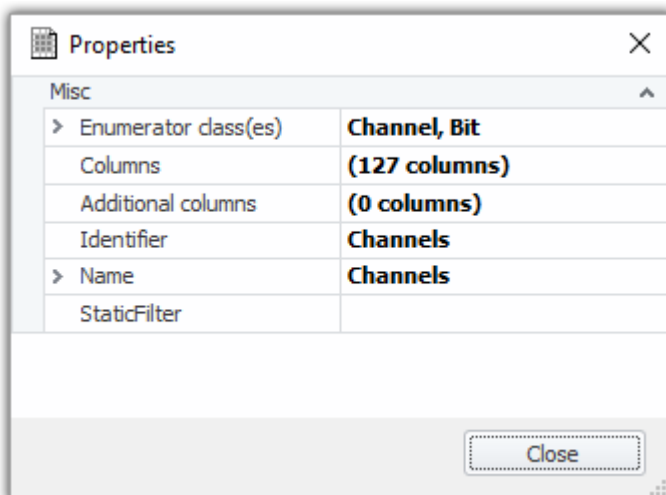
Viewing a table description

Click on the tab *Table description* on the right edge to open the description.



Tool window: Table description

Click in the context menu of the table in which the desired parameter is located, for example on the item *Properties* in the context menu for *Analog Channels*.



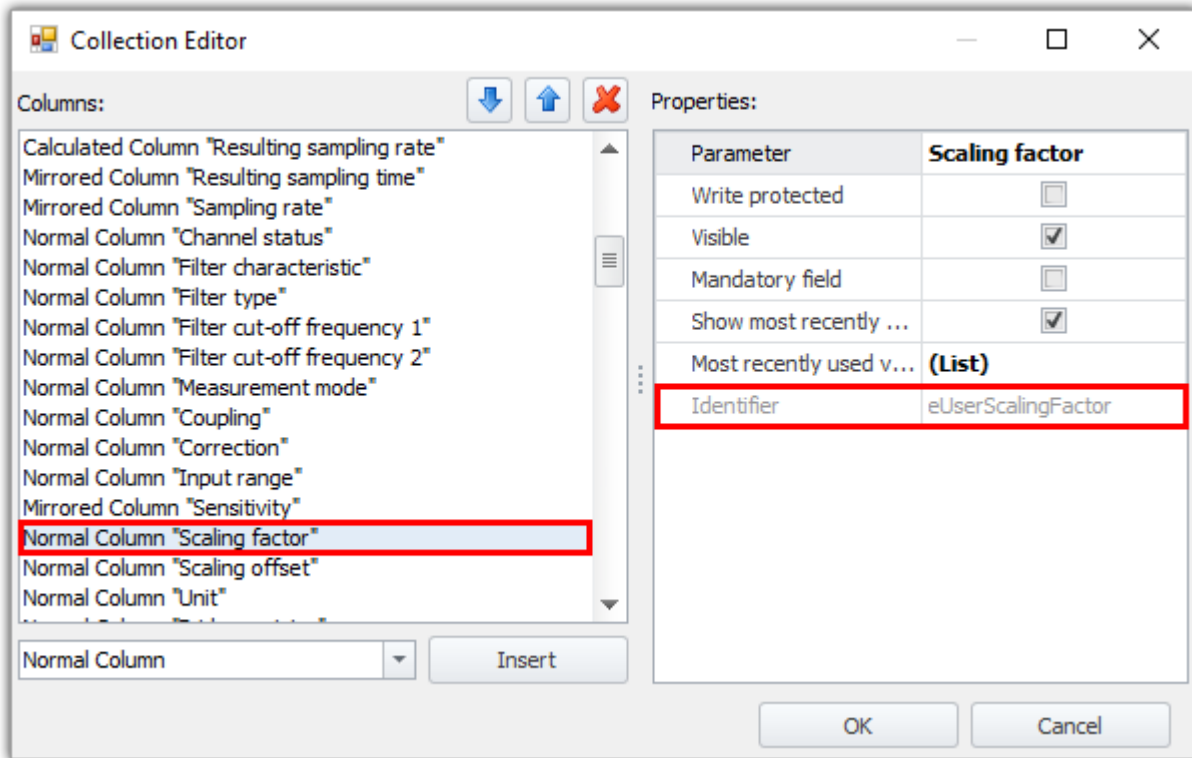
Properties: Analog channels

Click on the button with three dots "..." at the entry for Columns.

Parameter Identifier

After opening the column descriptions, select the pertinent parameter on the left side. On the right side, the associated identifier is stated.

In this example, the *Scaling factor* has been selected and is the identifier indicated is *eUserScalingFactor*.



Collection Editor

Finding setting values

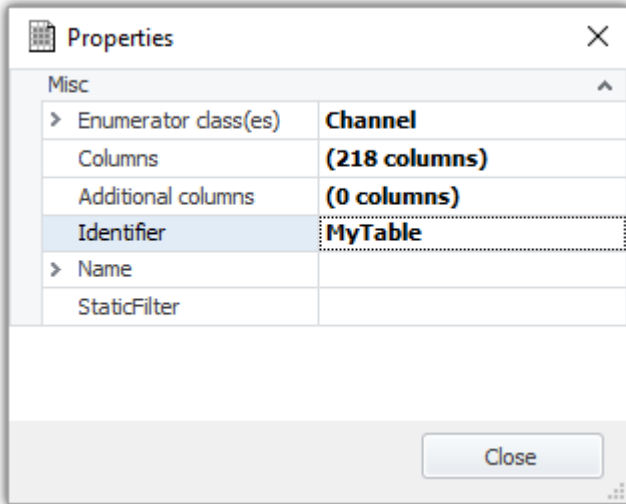


The setting values can be found in the class `ParameterValues` or in `imc.Studio.Tools.DevSetupGraphGenerator.DevSetupCommonValues`.

Another way to get the parameters and the respective values to which they are set is described in the section below.

Creating one's own table description

The easiest way is to create one's own table description. To do this, right-click in the *Table Descriptions* tool window on the folder icon (also titled *Table Descriptions*) to open the context menu, where you then select the item *Add*. Enter a name for the table description.

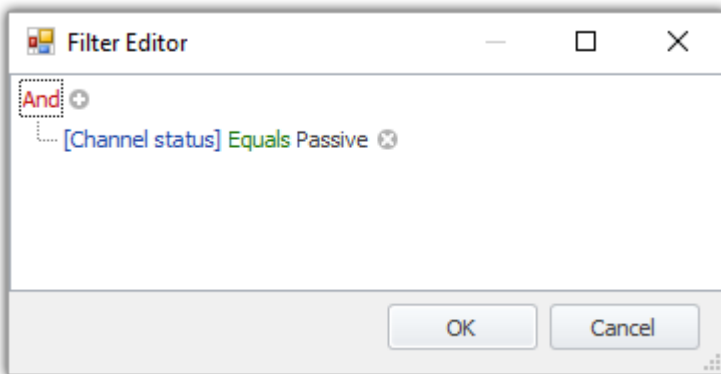


Properties of a user-defined table description

As the Enumerator class, select the desired class containing the parameter, e.g. *Channel*.

Filter Editor

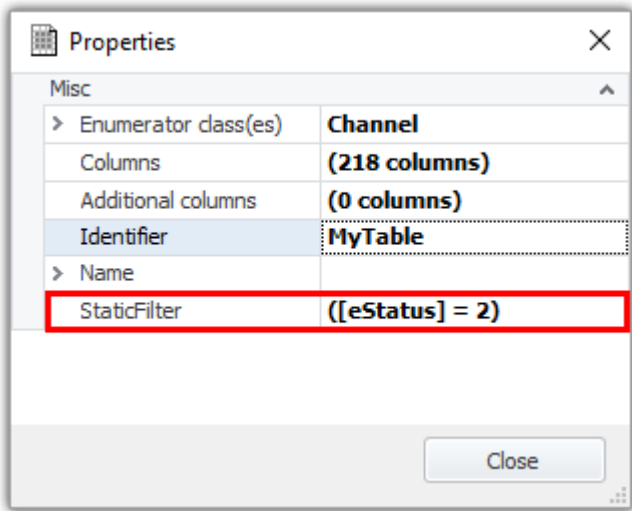
Under *StaticFilter*, select the parameter of which you wish to know what value is set for it, e.g. *Channel status*. For the value, either enter the value desired or select it from the drop-down list. Close the window.



Filter Editor

 Viewing

Subsequently, you can view the **EClassID**-name and the value to which it is set under *StaticFilter* in the Properties window, e.g. ([eStatus] = 2). In this case, it means that the channel status is passive.

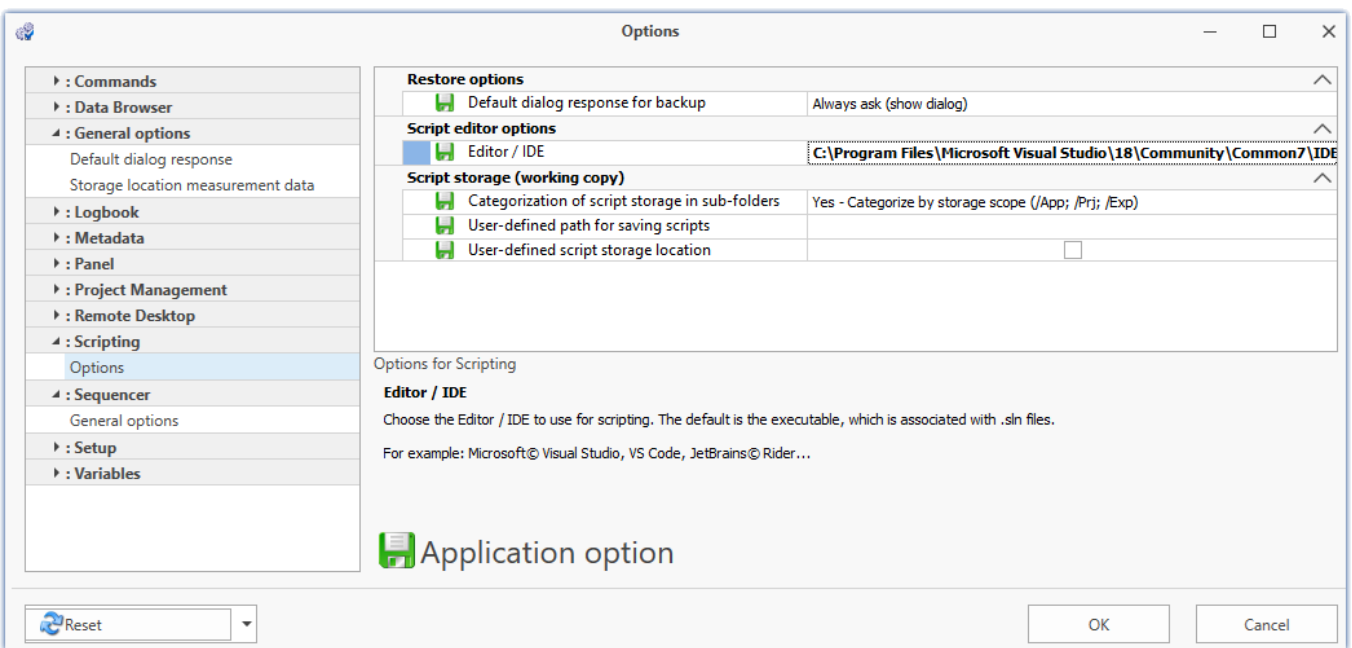


Properties: StaticFilter

3.5 Script options

The options can be reached by using the ribbon or by clicking on the button *Script options*  in the tool window *Scripts*.

Ribbon	View
Extra > Options 	All



Script options

Restore options

Option	Description
Default dialog response for backup	The scripts will be stored outside of imc STUDIO. The current state of the scripts is only applied when the associated storage scope is saved. If any changes were made to the script in the Script Editor after the last saving of the storage scope, this option can be used to automatically restore scripts to the last saved state of the script. • Always ask (show dialog): A <i>dialog</i> appears with the options to restore the last saved state. • Always restore (use last script saving state): Changes are always restored. The last saved state from the <i>Script Editor</i> is used. • Never restore (use latest backup in the respective storage scope): Changes are never restored. The last saved state of the scripts from the <i>Experiment</i> or <i>Project</i> is used. For more information, see chapter Restore script changes .

Script Editor options

Option	Description
Editor / IDE	This setting allows the user to select a custom program as the script editor. The editor selected here corresponds to the “Script Editor” described earlier in this manual. When a script is opened in the editor, the .sln file is opened. In the case of Visual Studio Code, the directory containing the .sln file is opened as a workspace. For optimal performance, we recommend one of the following editors or IDEs: • Microsoft Visual Studio 2022 • Microsoft Visual Studio 2026 • Visual Studio Code • VSCodium

Script storage (working copy)

Option	Description
Categorization of script storage in subfolders	The categorization specifies if different subfolders depending on the Script storage should be used or one common folder.
User-defined path for saving scripts	Folder in which a working copy of all scripts is saved, if user-defined script storage location is activated.
User-defined script storage location	The scripts are stored at a different location as the experiment or project. This can be helpful when using subversion tools (SVN). Stores a copy of all scripts in an arbitrary user-defined location. All scripts can be found in the given folder.

3.6 Tutorial

Here you will find a list of tutorials:

Tutorial	Description
Device and channel settings ¹¹⁵	A device is added to the device list. After selection of the device, the analog channels are configured.
Channel calculation ¹¹⁹	Different methods to calculate channels are shown.
List linked Variables ¹²¹	Shows a list of Widgets and their connection to variables.
RS232-Interface ¹²²	A example how to control a serial device.
Using the script clipboard ¹²³	The script clipboard is explained in a little example.

3.6.1 Device and channel settings

Channel settings

In this tutorial, a device is to added to the device list on the basis of its serial number (here: 123456) and then selected. Furthermore, all analog channels are to be set as follows:

Parameter	Value
State	active
Save to HDD	on
Sample rate	1000 Hz
Duration:	10 s
Measurement mode	Voltage
Coupling	DC
Input Range	±5 V

The following **libraries** are needed:

```
using imc.Common.Interfaces.Logbook;
using imc.Studio.Interfaces.DevSetup;
using imc.Studio.Tools.DevSetupGraphGenerator;
```

Once imc STUDIO has been started, a scripted named *DeviceSettings* is created by means of the Sequencer-page.

In the first step, the data above are recorded.

```
string errorSender = "DeviceSettings Script"; // Sender for Logbook
string serialNumber = "123456";

double sampleTime = 0.001; // 1 ms = 1000 Hz
double duration = 10.0; // 10 s
double range = 5; // +/- 5 V
string rangeText = "\u00b1" + range.ToString() + " V"; // +/- Symbol and Unit
```

If the device still does not appear in the Devices list, a device search is performed. Subsequently, the device is selected, if it is present.

```
// Get Device via serial number
EnumItem device = DevSetup.Devices.GetItemBySerialNumber(serialNumber);

// if device is not in list, do device search
if (device == null) {
    DevSetup.SearchDevices();
    device = DevSetup.Devices.GetItemBySerialNumber(serialNumber);

    // if device is not still in the list -> Warning
    if (device == null) {
        Logbook.LogEntry(errorSender, "Device with S/N " + serialNumber + " could not be found.", 0,
            ELogbookEntryCategory.Warning);
        return; // end script
    } // if
} // if

// get device name
string nickName = device.Parameters[EClassID.eDeviceNickname].GetValue().ToString();
string name = device.Parameters[EClassID.eDeviceName].GetValue().ToString();

// select device for measurement
DevSetup.SelectDevice(nickName, true, EDeviceCoreType_V2.Measurement);
```

For all analog channels, the above parameters are to be set as follows:

```
foreach( var channel in DevSetup.Channels){
    var paraType = channel.Parameters[EClassID.eChannelType];    //
    if (paraType != null){
        int chnType = (int)paraType.GetValue();
        // for analog channels
        if ( chnType == ParameterValues.eChannelType.AnalogInput) {

            // ... next block

        } // chnType
    } // paraType
} // foreach

// Process
Action.FireAction(this, "acCheckConfiguration");
```

Setting of the parameters is performed as follows:

```
var paraStatus      = channel.Parameters[EClassID.eStatus];
var paraSaveHD      = channel.Parameters[EClassID.ePCTransferToHD];
var paraSampleTime   = channel.Parameters[EClassID.eSampleTime];
var paraDuration     = channel.Parameters[EClassID.eDuration];
var paraRange        = channel.Parameters[EClassID.eRange];
var paraMode         = channel.Parameters[EClassID.eChannelMode];
var paraCoupling     = channel.Parameters[EClassID.eCoupling];
```

```

// set state to active
if (paraStatus != null){
    paraStatus.SetValue(ParameterValues.eStatus.Active);
} // paraStatus

// enable saving to PC
if (paraSaveHD != null){
    paraSaveHD.SetValue(true);
} // paraSaveHD

// set sample rate
if (paraSampleTime != null){
    paraSampleTime.SetValue(sampleTime);
} // paraSampleTime

// set duration
if (paraDuration != null){
    paraDuration.SetValue(duration);
} // paraDuration

// set mode to "Voltage"
if (paraMode != null){
    paraMode.SetValue(ParameterValues.eChannelMode.DAQ_Voltage);
} // paraMode

// set coupling to DC
if (paraCoupling != null){
    paraCoupling.SetValue(ParameterValues.eCoupling.DC);
} // para Coupling

// set Range
if (paraRange != null){
    paraRange.SetValue(rangeText);
} // paraRange

```

After the script has been successfully compiled, the Editor can be closed. Subsequently, the script can be run by means of a command, for example.

If the device does not yet appear in the Devices list, a device search is performed and the device is added to the list. The channel settings are performed and can be reviewed on the page *Analog Channels*. If the device is not found at all, a warning is recorded in the Logbook.

Extension: Input via Widgets

Supposing we want to expand on the example above. The goal is to be able to enter the sampling interval, the input range, and the measurement duration by means of Widgets.

First, to represent these parameters, create the user-defined variables *SampleTime*, *Range* and *Duration* in the Data-Browser. These variables are dragged from the Data Browser to a Panel page and displayed as a numerical display widget. It is also helpful to display the Setup-page *Analog channels* (Widget: Device Control Setup).

The value of each of these variables is queried in the script and assigned to the corresponding variable.

```

string errorSender = "DeviceSettings Script"; // Sender for logbook
string serialNumber = "123456";

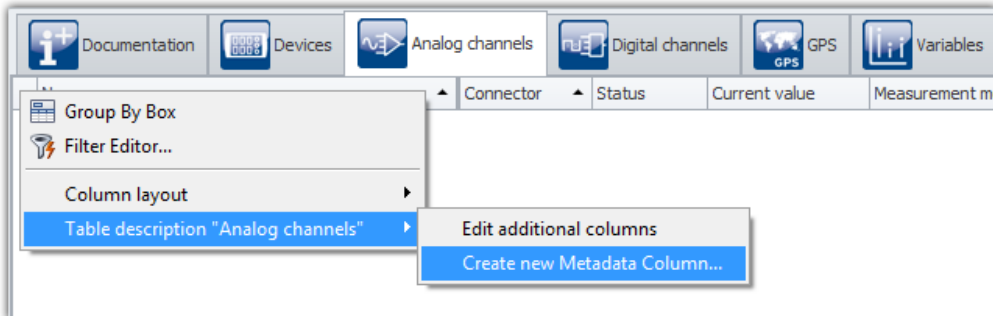
double sampleTime = Variables["SampleTime"].Value.Get<double>(); //
double duration = Variables["Duration"].Value.Get<double>(); //
double range = Variables["Range"].Value.Get<double>(); //
string rangeAsText = "\u00b1" + range.ToString() + " V"; // +/- Symbol and Unit

```

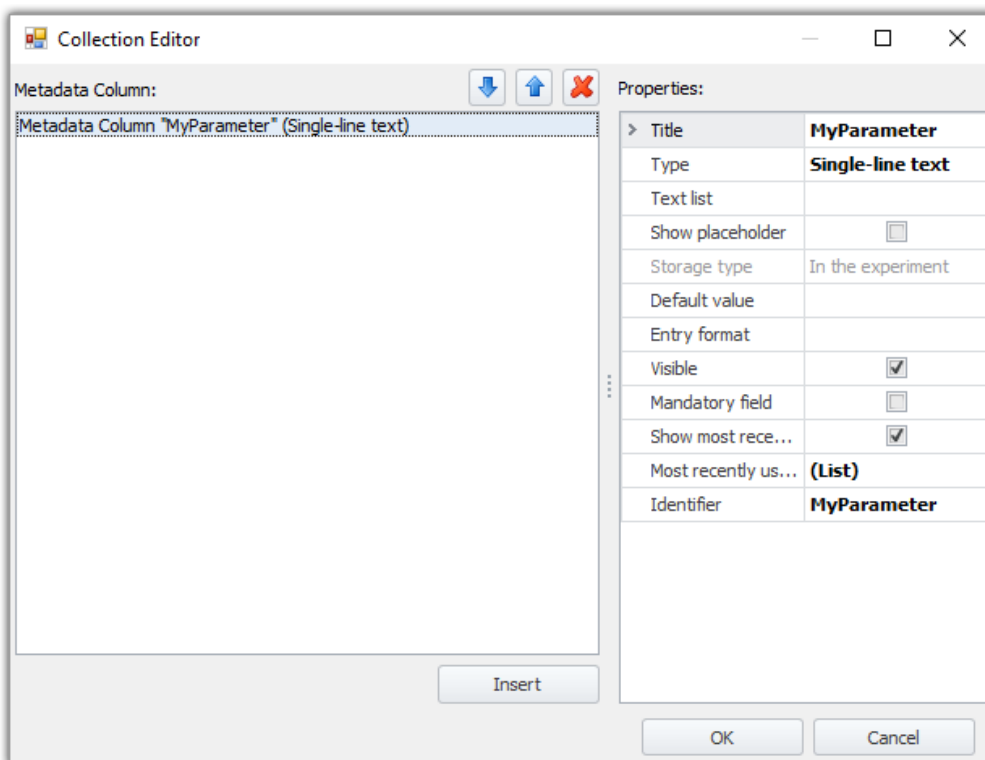
After the script has been compiled, it is possible to set the parameters by means of the widgets while the Design-mode is deactivated. A button which causes the script to run when pressed (command: *Run Script*) is added to the Panel page. When the button is activated, the values are updated to the *Analog channels* page.

Extension: Metadata columns

On the Setup-page *Analog channels*, create your own metadata column *MyParameter* by opening the context menu of **the upper left corner** and clicking on *Table description "Analog channels" > Create new Metadata Column...*



Analog Channels: Add Metadata column



Collection Editor: Metadata column "MyParameter"

This column is to be filled by the script with progressively higher numbers. Toward this end, a counting variable is generated. Save this view.

```
int count = 1;
foreach ( var channel in DevSetup.Channels) {
    // ...
} // foreach
```

The parameter setting process is supplemented with the following code:

```
var paraCustom = channel.Parameters["MyParameter"]; // Identifier
// ...
if (paraCustom != null) {
    paraCustom.SetValue("#" + count.ToString());
    count++;
}
```

After running the script, the newly added table column of the analog channels fills itself.

3.6.2 Channel calculation

Manual calculation

In this tutorial, analog channels are calculated. The following settings are to be made for two analog channels:

Parameter	Value
State	active
Save to HDD	on
Sample rate	1000 Hz
Duration:	10 s
Measurement mode	Voltage
Coupling	DC
Input Range	± 5 V

The script should be started after a measurement and add the current channels and write them to an output channel.

The following **libraries** are needed:

```
using imc Common Components DataManager Interfaces
using imc Data Contract
using imc Data Core Contract
using imc Data Core Contract Identifiers
using imc Data Storage Core Contract
using imc Studio Tools DevSetupGraphGenerator
```

Within the Run() method the sequence is implemented. Here, the variables are queried, the measurement is loaded, the channels from the variables are calculated and returned in a result channel:

```
public override void Run()
{
    const double sampleTime = 0.001; // better get from channel
    string measurementName = GetLastMeasurementName();

    IVariable measurementVariable = GetVariable(measurementName);

    Result<IVariable> inputVar1 = Variables.TryGet("Cosine");
    Result<IVariable> inputVar2 = Variables.TryGet("DC");

    LoadMeasurement(measurementName);

    IDmChannel channel1 = inputVar1.Value.QueryInterface<IDmChannel>().Value;
    IDmChannel channel2 = inputVar2.Value.QueryInterface<IDmChannel>().Value;

    var resultChannelVariable = CreateChannelVariable("ResultChannel", sampleTime);

    var sumResult = AddChannels(channel1, channel2);

    WriteDataToChannelVariable(resultChannelVariable, sumResult);
}
```

The following functions are used:

```
private T EvaluateResult<T>(Result<T> result)
{
    if (result.IsSuccessful)
    {
        return result.Value;
    }
    else
    {
        // ErrorHandling
        throw result.Error;
    }
}

public string GetLastMeasurementName()
{
    var measurementManagerService = Variables.Services.Get<IMeasurementManagerService>();
    var result = measurementManagerService.FindLastMeasurement();

    if (result.IsSuccessful)
    {
        return result.Value.DisplayName.ToString();
    }

    throw result.Error;
}

public IVariable GetVariable(string variableName)
{
    var result = Variables.TryGet(new DisplayName(variableName));

    return EvaluateResult(result);
}

public double[] AddChannels(IDmChannel channel1, IDmChannel channel2)
{
    var data1 = channel1.Data;
    var data2 = channel2.Data;
    var result = new double[data1.Length];

    for (int i = 0; i < result.Length; i++)
    {
        result[i] = data1[i] + data2[i] + 42;
    }

    return result;
    // return data1.Zip(data2, (x, y) => x + y).ToArray();
}

public void WriteDataToChannelVariable(IVariable channelVariable, double[] data)
{
    channelVariable.Value.Set(data);
}

public void LoadMeasurement(string measurementName)
{
    var measurementManagerService = Variables.Services.Get<IMeasurementManagerService>();
    Result<IMeasurementService> measurementResult = measurementManagerService.TryGet(measurementName);

    if (measurementResult.IsSuccessful)
    {
        var measurement = measurementResult.Value;
        measurement.Load().ThrowExceptionIfAny();
    }
}

public IVariable CreateChannelVariable(string name, double sampleTime)
{
    var result = Variables.TryGet(new DisplayName(name));

    if (result.IsSuccessful)
    {
        Variables.Delete(new DisplayName(name));
        // return result.Value;
    }
}
```

The following functions are used:

```
var variable = Variables.Create(new DisplayName(name));
variable.Properties.Set(EClassID.eDataType, DevSetupCommonValues.eDataType.IEEE754DOUBLE);
variable.Properties.Set(EClassID.eXFormat, DevSetupCommonValues.eXFormat.Normal);
variable.Properties.Set(EClassID.eObjectType, typeof(IDmChannel));
variable.Properties.Set(EClassID.eSampleTime, sampleTime);

return variable;
}

public IDmChannel GetChannel(IVariable channelVariable)
{
    var result = channelVariable.QueryInterface<IDmChannel>(); // measurement needs to be loaded,
    otherwise this interface is not supported
    return EvaluateResult(result);
}

public IVariable GetChannelVariable(IVariable measurementVariable, string variableName)
{
    var result = measurementVariable.Children.TryGet(new DisplayName(variableName));

    return EvaluateResult(result);
}
```

3.6.3 List linked variables

In this tutorial all Widgets on all Panel pages should be listed including the variable link.

The following **Libraries** are used:

```
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

In the `Run()` method you need to add the following lines.

```
StringBuilder text = new StringBuilder();

// for special Widgets
Panel.Pages.ToList()
    .ForEach(page => page.Widgets.ToList()
        .Where(widget => widget.Variable != null)
        .ToList()
        .ForEach(widget => text.Append(page.DisplayedCaption + " | " + widget.Name + " | " +
widget.Variable.Name + "\n")));

// for Standard-Widgets
Panel.Pages.ToList()
    .ForEach(page => page.Widgets.ToList()
        .Where(widget => widget.Variable == null)
        .ToList()
        .ForEach(widget => text.Append(page.DisplayedCaption + " | " + widget.Name + " | " +
widget.StandardVariable.Name + "\n")));

MessageBox.Show(text.ToString());
```

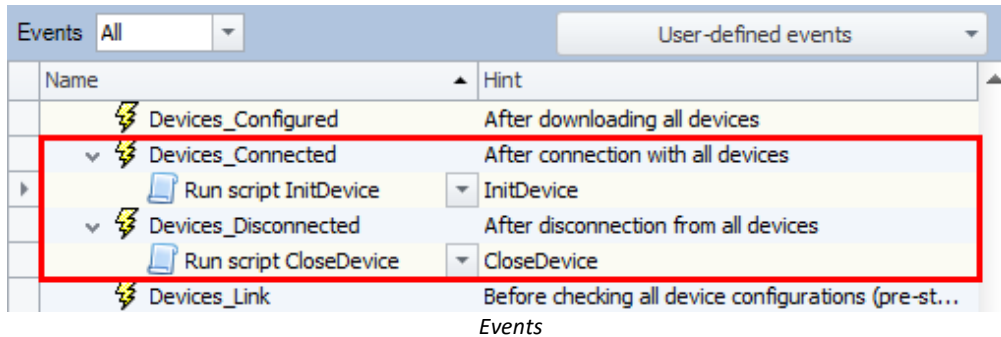
In the message box a list of all Widgets on all Panel pages and their variable connection are shown.

You can also write the results to a file.

3.6.4 RS232-interface

The following **libraries** are needed:

```
using System.IO.Ports;
```



Initialisation / Connect

Add the command *Run script* with the following code to the event *Devices_connected*. The event is released, when connected to devices.

InitDevice.cs

```
try{
    // Create a COM port object and define the port's settings
    SerialPort serialPort = new SerialPort("COM1", 9600, Parity.None, 8, StopBits.One );
    // Open the port
    serialPort.Open();
    // Store the COM port in the clipboard
    ScriptClipboard.Add("COM1", serialPort);
} catch (Exception x){
    // error handling
}
```

Close / Disconnect

Add the command *Run script* with the following code to the event *Devices_disconnected*. The event is released, when disconnected from devices.

CloseDevice.cs

```
try{
    // Get the COM port from clipboard
    SerialPort serialPort = ScriptClipboard["COM1"] as SerialPort;
    // Close the COM port
    serialPort.Close();
    // Remove the COM port from clipboard
    ScriptClipboard.Remove("COM1");
} catch (Exception x){
    // error handling
}
```

Read data

ReadData.cs

```
try{
    // Get the COM port from clipboard
    SerialPort serialPort = ScriptClipboard["COM1"] as SerialPort;
    // Read a message from the device
    string dataAsText = serialPort.ReadLine();
} catch (Exception x){
    // error handling
}
```

Write data

WriteData.cs

```
try{
    // Get the COM port from clipboard
    SerialPort serialPort = ScriptClipboard["COM1"] as SerialPort;
    // Send a message to the device
    serialPort.WriteLine("*IDN?");
} catch (Exception x){
    // error handling
}
```

3.6.5 Using the script clipboard

In this tutorial, the script clipboard is to be used to make a Widget belonging to a Panel-page available to a different script.

ScriptA.cs

When adding an object into the script clipboard, it is necessary to be aware that the object has a namespace. In the case of a Widget, this would be the script's name due to the proxy-classes.

For this reason, the script must be created with the option "*Separate Proxy class library*". Then the proxy-class will be integrated into the script as a resource.

```
Widget myWidget = Panel["Page 1"]["MyWidget"]; // Type: ScriptA.Widget
ScriptClipboard.Add("MyWidget", myWidget);
```

ScriptB.cs

The script can be created without the option "Separate proxy class library".

Under *References > Add Reference*, the **ProxyClasses.dll** is added from the **ScriptA-Project**. The file is located in the **ScriptA-project's bin-folder**.

```
object o;
bool b = ScriptClipboard.TryGetValue("MyWidget", out o);

if ( (b) && (o != null) ){
    ScriptA.Widget myWidget = o as ScriptA.Widget;
}
```

3.7 Important information



Warning

Important changes when moving from a imc STUDIO 2022 ... 2025 version

When switching from a 2022 version to 2025, your scripts will be converted as far as possible, primarily due to the transition from .NET Framework 4.8 to .NET Core (8.0).

However, some manual adjustments may still be necessary, especially for more complex scripts:

- During conversion, the tool attempts to incorporate the user's changes to the .csproj file. However, this cannot be guaranteed.
- The .sln file is converted to a modern format. This should be relatively straightforward.
- The .NET SDK is required, but it is installed along with imc STUDIO.
- Some classes and functions, as well as NuGet packages, etc., behave differently in .NET Core (8.0) than in .NET Framework 4.8. Manual adjustments may be necessary.
- Since there is no longer a "Global Assembly Cache" (GAC) in .NET Core (8.0), registry entries are used for directory name resolution.
- Users can specify which version of imc STUDIO they want to use with this script by changing the version number in the <IMCStudioPath> tag of the .csproj file.



Warning

Important changes when moving from a imc STUDIO 5.x version

When switching from a 5.x version, your scripts will be converted as far as possible. The instructions for upgrading from a 2022 ... 2025 version apply, and additional manual adjustments may be necessary (including variable accesses):

- Change variable accesses from "Datapool" to "Variables"
- imc.Studio.Interfaces.V2 has been integrated into imc.Studio.Interfaces and is therefore obsolete.

(older) Experiment import: System.IO.FileNotFoundException

When importing experiment from former versions, there might be error messages, when running the scripts.

Esp. when the scripts were imported as DLL (assembly). In this case the script cannot be compiled automatically.

If the source code is available and you have a Developer-Edition, the error can be fixed by compiling the code again and may correct the references. You might need to [Regenerate proxies](#) .

In the case of a DLL-script the developer of the script needs to compile the script in the corresponding version. Please contact the developer of the script.

Close Script Editor before "Experiment > Save as"



Note

When executing *Experiment > Save As..*, the **Script editor** must be closed, otherwise it will still have the script belonging to the original experiment open, and will not automatically alter the path.

imc STUDIO GUI can not be operated

Note

While the script's `Run ()` method is in progress, the imc STUDIO user interface can not be operated. For this reason, the script may not contain any blocking functions.

Renaming elements

Note

If devices, channels or Widgets which are accessed by imc STUDIO Scripting are renamed or deleted in imc STUDIO, then the script must be adapted accordingly.

Event handling is asynchronous

Note

The event handling methods are run **asynchronously**.

4 Third Party Device Interface

Documentation on the **imc STUDIO** plug-in **imc STUDIO Third Party Device Interface**.

4.1 Getting started

4.1.1 Introduction



Note

- A **imc STUDIO ThirdPartyDevice License** (inclusive/exclusive) is needed to run a Third Party Device script.¹
- A **imc STUDIO Developer-Edition** is needed to edit a Third Party Device script.
- The 3rd Party Device Management is available in all editions except Runtime.

¹The Third Party Device *AudioDevice*, *ChannelLoader* and *FunctionSimulator* can be used without any 3PDI license.

With the plugin **imc STUDIO Third Party Device Interface** devices from other manufacturers (Third Party Devices) can be added to the Setup page in **imc STUDIO** easily.

Therefore a C#-Script has to be implemented containing properties of the Third Party Device. A template code and fundamental functions are given by **imc STUDIO**. In this script the interface to the Third Party Device has to be implemented. After finishing the script, the Third Party Device will appear in the device list on the Setup page. The Third Party Device can now be configured in **imc STUDIO** and can also be combined with devices manufactured by **imc**. For example an oscilloscope or a multimeter can be added.

The advantages of Third Party Device Script are:

- consistent measurement format
- synchronous visualization of the measurement data
- common data saving
- common project management
- common management of the device configuration
- no GUI Programming needed, integration of the device in **imc STUDIO**
- existing device and channel parameters can be used for the Third Party Device
- new parameters can be created

Needs:

- Experience in C# programming is necessary (C++ for an *unmanaged* data transfer)
- The interface of the Third Party Device has to be known.

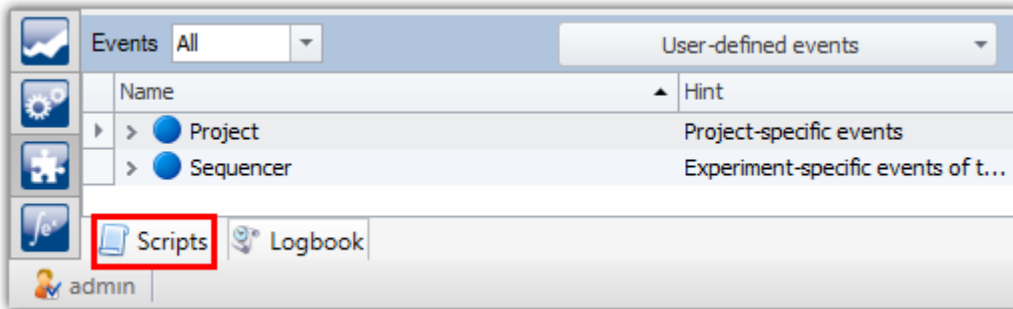
4.1.2 Add a new third party script



Reference

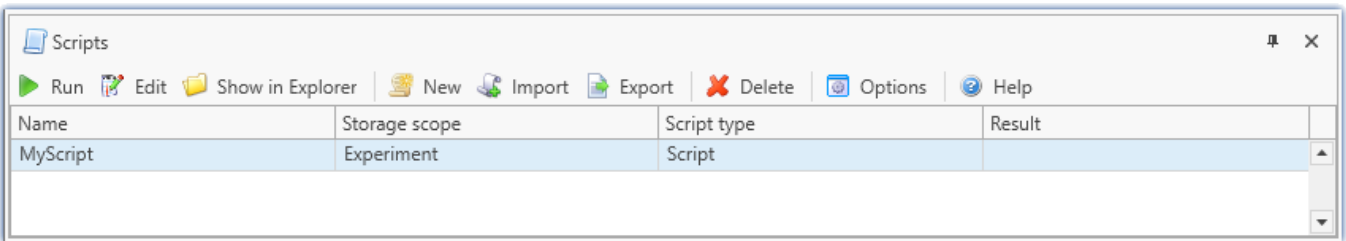
For more information about **Scripting**, see [here](#) .

Third party scripts are created in the Scripts tool window, just like all other script types.



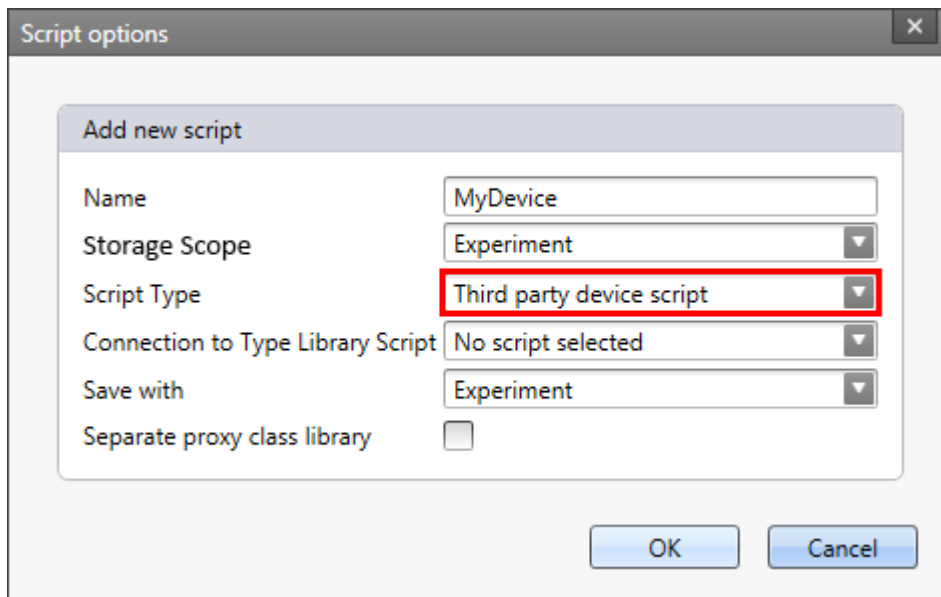
Sequencer: Tool window "Scripts"

The tool window *Scripts* will open. Click on **New**  to add a new Script



Sequencer: Tool window "Scripts"

The Script Type has to be chosen: Third party device script.



Dialog: Third Party Device Script

The editor will open automatically after adding a script containing a template of C#-Code.



Note

You can find additional templates for third party device scripts in the [3rd Party Device Management](#) .

4.2 Serial interface

4.2.1 Device control

If an integration to the setup page is not needed, and data should be *polled* from the device by an event or clicking on a button, a normal script can be used.

For example a climate cabinet controlled by RS232 which sends single values of the temperature on command.

Here scripts for opening and closing the connection and for reading and writing data are described.

These code snippets can be bound to the Sequencer events, e.g. "Device connected/disconnected".

Open COM Port/ connect to the device

```
try{
    // Create a COM port object and define the port's settings
    System.IO.Ports.SerialPort serialPort = new System.IO.Ports.SerialPort("COM1", 9600, Parity.None, 8,
    StopBits.One);
    // Open the port
    serialPort.Open();
    // Store the COM port in the clipboard
    ScriptClipboard.Add("COM1", serialPort);
} catch (Exception x){
    // error handling
}
```

Close COM Port / disconnect from device

```
try{
    // Get the COM port from clipboard
    System.IO.Ports.SerialPort serialPort = ScriptClipboard["COM1"] as System.IO.Ports.SerialPort;
    // Close the COM port
    serialPort.Close();
    // Remove the COM port from clipboard
    ScriptClipboard.Remove("COM1");
} catch (Exception x){
    // error handling
}
```

Write data

```
try{
    // Get the COM port from clipboard
    SerialPort serialPort = ScriptClipboard["COM1"] as SerialPort;
    // Send a message to the device
    serialPort.WriteLine("*IDN?");
} catch (Exception x){
    // error handling
}
```

Read data

```
try{
    // Get the COM port from clipboard
    SerialPort serialPort = ScriptClipboard["COM1"] as SerialPort;
    // Read a message from the device
    string strData = serialPort.ReadLine();
} catch (Exception x){
    // error handling
}
```

4.3 Third party device script

4.3.1 Methods

void GetAvailableDevices(SetupNodeList deviceList)

Description:	Gets all the currently available devices which could be selected for measurement later on.
Is called when:	search for devcies is performed.
To implement:	Search for the 3rd-party devices, as well as for device parameters of the devices found

void GetDeviceDescription(SetupNode device)

Description:	Gets the complete definition of the device with all modules and channels.
Is called when:	the device is selected for a measurement.
To implement:	Additional parameters for the selected devices and add channels and their properties.

void DeselectDevice(SetupNode device)

Description:	Is called when the device is deselected by the user. Here it is possible to do some cleanup.
Is called when:	the device is deselected from a measurement.
To implement:	Take all actions necessary for de-selecting the device.

void PrepareMeasurement(SetupNode device)

Description:	Prepares the specified device for measurement, downloads the configuration from the PC to the device. Can throw an exception if an error occurred.
Is called when:	Performing a <i>Process</i> action.
To implement:	Actions necessary when performing a Process action. E.g. processing user inputs.

void ProcessConfiguration(SetupNode device)

Description:	Processes the device configuration and checks if the the configurations have errors. If an error is detected throws an exception.
Is called when:	Performing a <i>Prepare</i> action.
To implement:	Actions necessary when performing a Prepare action. E.g. processing user inputs.

void ConnectDevice(SetupNode device)

Description:	If necessary establishes the connection to the specified device. If the connection can not be established an exception is thrown.
Is called when:	the user connects to the devices.
To implement:	stablishment of a connection to the device.

void DisconnectDevice(SetupNode device)

Description:	If the implementation of the method ConnectDevice() established the connection, closes the connection to the specified device. If there was an error an exception is thrown.
Is called when:	the user disconnects from the device.
To implement:	Disconnection from the device.

void StartMeasurement(SetupNode device)

Description:	Starts the measurement for the specified device. If there was an error an exception is thrown.
Is called when:	Measurement is started.
To implement:	Start the device. Access to the device driver can also be done in the <i>DataTransferDriver</i> .

void StopMeasurement(SetupNode device)

Description:	Stops the measurement for the specified device.
Is called when:	Measurement is stopped.
To implement:	Stop the device. Access to the device driver can also be done in the <i>DataTransferDriver</i> .

void GetParameterConstraints(SetupNode node, SetupParameter parameter, ParameterConstraint constraint)

Description:	Gets the constraints for a parameter. The constraints can be defined with calls to the constraint instance. Constraints can be upper and/or lower value borders for numeric parameters, value lists for combo boxes, enabled or disabled states according to other parameters.
Is called when:	the device and channel configuration is loaded.
To implement:	Selection options for parameters, e.g. the sampling interval.

Please use the correct numeric format as parameter, e.g.:

```

if (parameter.Name == EClassID.eSampleTime.ToString()) {
    constraint.ValueList.Add(0.1);
    constraint.ValueList.Add(1d); // for double
    // constraint.ValueList.Add(1); // for double -> error
    // ...
}

```

void ExecuteParameter(SetupNode node, int actionStepID)

Description:	Is executed if a non standard action should be executed. If there was an error an exception is thrown.
Is called when:	Device action (e.g. balancing, eAction, ...) is invoked.
To implement:	Regarding to the ActionID further actions can be performed.

string GetParameterDisplayText(SetupNode node, SetupParameter parameter, object value, bool reciprocal, bool useRoundTripFormat)

Description: Gets the display text of the parameter's value.
 Is called when: the device and channel configuration is loaded.
 To implement: Display texts for (numerical) parameter presets.

```
if (parameter.Name == EClassID.eAction.ToString()) {
    if (object.Equals(value, 1)) {
        return "Some action";
    }
    // ...
}
```

object ConvertParameterTextToValue(SetupNode node, SetupParameter parameter, string inputText, bool reciprocal)

Description: Converts the input text of a parameter to the value type of the parameter. If the input text contains an error, throws an exception.
 Is called when: the user enters a value to a parameter.
 To implement: Conversion of inputs to parameter values.

bool? ValidateParameter(SetupNode node, SetupParameter parameter, object oldValue)

Description: Validates the parameter. Here it is possible to change a parameter value according to the values of other parameters or to prohibit the change.
 Is called when: the user enters a value to a parameter.
 To implement: Parameter validation with respect to certain boundaries, such as limiting the measurement time to a maximum value.

void LoadDeviceConfiguration(SetupNode device, string configurationVersion)

Description: Informs the device provider about a configuration that was loaded by the device Setup. Here it is possible to convert the device configuration to the current version, i.e. by adding new parameters.

4.3.2 Add a device

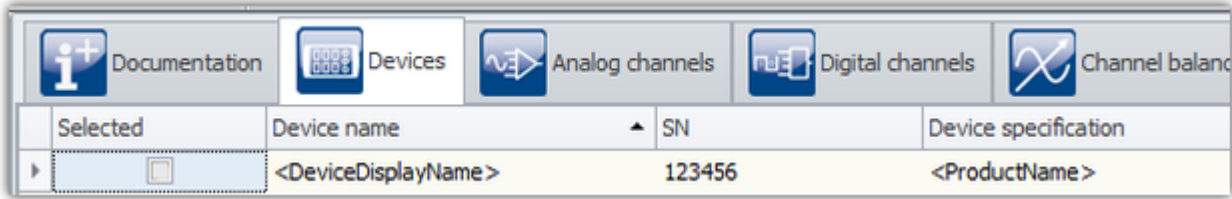
First of all the device (or the devices) has to be configured. In the method `GetAvailableDevices()` the properties of the devices is programmed.

In this example a device is added with the serial number "123456". Further device parameter can be added, too. Therefore the according **EClassID** parameters have to be added.

GetAvailableDevices

```
protected override void GetAvailableDevices(SetupNodeList deviceList) {
    // Add the currently available devices here! Example:
    var device = deviceList.GetOrAdd(ESetupNodeType.Device, "<DeviceName>");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceNickname, value: "<DeviceDisplayName>");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceSN, value: "123456");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceProduct, value: "<ProductName>");
}
```

After closing the editor, the Third Party Device is shown on the Setup in the device list.



Setup: Device list

If the device is selected, the device configuration (channels and variables) is loaded. The next step is to [add channels and variables](#)¹³² to the device.

4.3.3 Add a channel

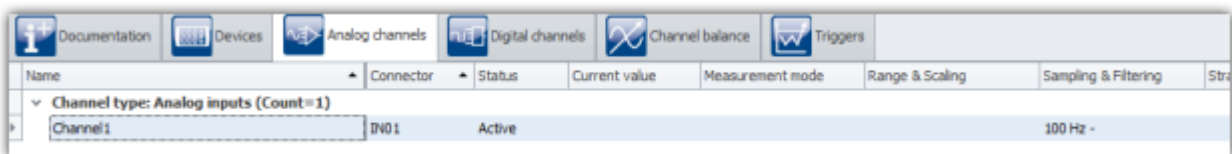
The channel configuration of the parameters is set in the following procedure. This method is run when the device is selected. In this example the configuration is made for only one channel. To configure [more channels](#)¹⁵⁸, use a loop or create a method. Here, the channel type is set to analog input, but also Display variables or other types are possible. For all possibilities see `ParameterValues.eStatus`.

GetDeviceDescription

```
protected override void GetDeviceDescription(SetupNode device) {
    // Add the full device description here! Channels, modules... Example:
    var channel = device.Children.GetOrAdd(ESetupNodeType.Channel, "<DeviceName>.Channel1");
    channel.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "Channel1");
    channel.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "IN01");
    channel.Parameters.UpdateOrAdd(EClassID.eStatus, value: ParameterValues.eStatus.Active);
    channel.Parameters.UpdateOrAdd(EClassID.eChannelType, value:
        ParameterValues.eChannelType.AnalogInput);
    channel.Parameters.UpdateOrAdd(EClassID.eDivisionUnit, value: "V"); // The original (electrical)
    measurement unit
    channel.Parameters.UpdateOrAdd(EClassID.eUserUnit, value: "V");
    channel.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.IEEEFloat);
    channel.Parameters.UpdateOrAdd(EClassID.eSampleTime, value: 0.01); // 100 Hz - set also the
    selectable values in the method
}

// ParameterValues.cs
public sealed class eChannelType {
    public const int AnalogInput = 1;
    public const int DAC = 2;
    public const int DIOPort = 3;
    public const int VirtChannel = 5;
    public const int DIOBit = 6;
    public const int VirtBit = 7;
    public const int NetBit = 8;
    public const int EncoderInput = 9;
    // ...
    public const int DisplayVar = 15;
    // ...
    public const int PVVar = 24;
    public const int VideoChannel = 1001;
    // ...
    public const int DIO_In = 4099;
    public const int DIO_Out = 8195;
}
```

After selecting the device, the channel is available in the analog channel list.



Setup: analog channels

To [add a Display Variable](#)¹⁶⁰ to the device, the channel type has to be set to "DisplayVar".

You have to Remove some parameters for Display variables and pv-variables:

```
variable.Parameters.Remove(EClassID.eSampleTime);
variable.Parameters.Remove(EClassID.eDuration);
variable.Parameters.Remove(EClassID.ePCTransferToHD);
```

4.3.3.1 Single Value

Single value as text:

```
SetupNode singleValueText = device.Children.GetOrAdd(ESetupNodeType.Channel, "SingleVarString");
singleValueText.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "SingleVariableString");
singleValueText.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "IN01");
singleValueText.Parameters.UpdateOrAdd(EClassID.eStatus, value: ParameterValues.eStatus.Active);
singleValueText.Parameters.UpdateOrAdd(EClassID.eXFormat, value: ParameterValues.eXFormat.SingleValue);
singleValueText.Parameters.UpdateOrAdd(EClassID.eChannelType, value:
ParameterValues.eChannelType.DisplayVar);
singleValueText.Parameters.UpdateOrAdd(EClassID.eObjectType, value: typeof(string).ToString());
singleValueText.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.ASCII);

singleValueText.Parameters.Remove(EClassID.eSampleTime);
singleValueText.Parameters.Remove(EClassID.eDuration);
singleValueText.Parameters.Remove(EClassID.ePCTransferToHD);
```

Single value as number/Integer

```
SetupNode singleValueInt = device.Children.GetOrAdd(ESetupNodeType.Channel, "SingleVarInteger");
singleValueInt.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "SingleVariableInteger");
singleValueInt.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "IN02");
singleValueInt.Parameters.UpdateOrAdd(EClassID.eStatus, value: ParameterValues.eStatus.Active);
singleValueInt.Parameters.UpdateOrAdd(EClassID.eXFormat, value: ParameterValues.eXFormat.SingleValue);
singleValueInt.Parameters.UpdateOrAdd(EClassID.eChannelType, value:
ParameterValues.eChannelType.DisplayVar);
singleValueInt.Parameters.UpdateOrAdd(EClassID.eObjectType, value: typeof(int).ToString());

singleValueText.Parameters.Remove(EClassID.eSampleTime);
singleValueText.Parameters.Remove(EClassID.eDuration);
singleValueText.Parameters.Remove(EClassID.ePCTransferToHD);
```

4.3.3.2 Ports and Bits

```
const int numberOfBitsPerPort = 8;

var port = device.Children.GetOrAdd(ESetupNodeType.Channel, "Poll_DIn_Port");
port.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "Poll_DIn_Port");
port.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "DIN");
port.Parameters.UpdateOrAdd(EClassID.eChannelType, value: ParameterValues.eChannelType.DIOPort); // or
DIO_In, DIO_Out
// ... other Parameters

for (int k = 0; k < numberOfBitsPerPort; k++)
{
    string name = k.ToString("D1");
    var bit = port.Children.GetOrAdd(ESetupNodeType.Channel, "Poll_DIn_Bit" + name);
    bit.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "Poll_DIn_Bit0" + name);
    bit.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "DIN_Bit0" + name);
    bit.Parameters.UpdateOrAdd(EClassID.eChannelType, value: ParameterValues.eChannelType.DIOBit);
    // ... other Parameters
}
```

4.3.3.3 FFT

In order to create a FFT-channel, the `eXFormat` must be set to FFT. Additionally, various segment parameters must be set:

```
var channel = device.Children.GetOrAdd(ESetupNodeType.Channel, "<DeviceName>.Channel1");
channel.Parameters.UpdateOrAdd(EClassID.eXFormat, value: ParameterValues.eXFormat.FFT);
channel.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.IEEEFloat);
channel.Parameters.UpdateOrAdd(EClassID.eSamplesPerSegment, value: 512);
channel.Parameters.UpdateOrAdd(EClassID.eSegmentOffset, value: 0);
channel.Parameters.UpdateOrAdd(EClassID.eSegmentDelta, value: 5);
channel.Parameters.UpdateOrAdd(EClassID.eSegmentUnit, value: "Hz");
channel.Parameters.UpdateOrAdd(EClassID.eSampleTime, value: 0.1, editorType: EEditorType.ReadOnlyEdit);
```

The section above belongs to the source text in the collapsed region "Example for FFT channel parameters (streaming vector / segmented channel)"-

4.3.3.4 Time stamped



The use of time-stamped channels is explained using the *SimplePollDevice* template.

The example code is not generally valid and must be adapted according to the third-party device.

GetDeviceDescription

For a time-stamped channel, in the `GetDeviceDescription` method the channel parameter `eXFormat` is set to "Timestamp".

```
var channel = device.Children.GetOrAdd(ESetupNodeType.Channel, "Channel1");
channel.Parameters.UpdateOrAdd(EClassID.eXFormat, value: ParameterValues.eXFormat.Timestamp);
channel.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.IEEEFLOAT);
channel.Parameters.UpdateOrAdd(EClassID.eSampleTime, value: 0.001, editorType: EEditorType.Hidden);
// for TimeStamped-ASCII (log channel) set DataType to ASCII
channel.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.ASCII);
```

DataTransferDriver

A `TimeStampedSample` array must be passed to the `WriteData`-Method in the data transfer driver.

In this example, the *SimplePollDevice* template is used.

```

// OnProcess-Method
// ...
if (dataWriter != null)
{
    var timeStampedData = GetTimeStampedData(dataTransferNode); // get time stamped data from the device

    if (timeStampedData != null)
    {
        EResult result = dataWriter.WriteData(timeStampedData, 0); // write time stamped data to imc
        STUDIO // ...
    }
}
// ...

private TimeStampedSample<double>[] GetTimeStampedData(DataTransferNode dataTransferNode)
{
    // get time stamped data from the device
    // use simulated data from the DeviceDriver
    double[] data = m_deviceDriver.ReadData(dataTransferNode.Name); // simulated data
    var sampleTime = m_deviceDriver.GetSampleTime(dataTransferNode.Name); // method was added in
    DeviceDriver
    var timeStampedData = ConvertToTimeStamped(data, sampleTime); // conversion needed to get time
    stamped data
    return timeStampedData;
}

private double lastArgument = 0;
private DataTransferDateTime deviceStartTime;

private TimeStampedSample<double>[] ConvertToTimeStamped(double[] data, double sampleTime)
{
    if (data == null)
    {
        return null;
    }

    var timeStampedData = new TimeStampedSample<double>[data.Length];
    var delayedTime = m_deviceDriver.CurrentTime - deviceStartTime;

    for (int i = 0; i < timeStampedData.Length; i++)
    {
        lastArgument = delayedTime.Ticks / (sampleTime * DataTransferTimeSpan.TicksPerSecond);
        var argument = lastArgument + i;
        var argumentAsLong = Convert.ToInt64(argument);

        timeStampedData[i].Offset = argumentAsLong;
        timeStampedData[i].Value = data[i];
    }

    return timeStampedData;
}

```

DeviceDriver

A method `GetSampleTime` is added to the device driver.

```

public double GetSampleTime(string channelName)
{
    var sampleTime = 0d;

    Channel channel;
    if (m_channels.TryGetValue(channelName, out channel))
    {
        sampleTime = channel.SampleTime;
    }

    return sampleTime;
}

```

4.3.3.5 eRange parameter

The following instructions are needed to add the eRange parameter to a channel.

GetDeviceDescription

The parameter *eRange* needs to be added to the devices description. The values can be selected in a combo box.

```
// var channel = ...
// ...
channel.Parameters.UpdateOrAdd(EClassID.eRange, value: 10, editorType: EEditorType.ComboBox);
```

GetParameterConstraints

In the method **GetParameterConstraints** (in the region "Optional methods") the items of the selection are set.

```
if (parameter.Name == EClassID.eRange.ToString()) {
    constraint.ValueList.Add(20);
    constraint.ValueList.Add(10);
    constraint.ValueList.Add(5);
    constraint.ValueList.Add(2.5);
    constraint.ValueList.Add(1);
}
```

GetParameterDisplayText

If a different text should be shown in the list, you need to add the following code to the method **GetParameterDisplayText** (in the region "Display text handling").

```
if (parameter.Name == EClassID.eRange.ToString()) {
    // if (object.Equals(value, 20) ) {
        return "\u00b1" + value + " V";
    // }
}
```

If you need to distinguish between the values, you can use an if- or switch-case statement.

The methods also needs to return a value, e.g. `null`.

DataTransferDriver.cs : Prepare()

To use the parameter, you need to iterate over the SetupNode-Children:

```
foreach (SetupNode child in configuration.Children) {
    if (child.NodeType == EClassID.eChannel.ToString()) {
        SetupParameter range = child.Parameters.GetByName(EClassID.eRange);
        // do something with range.Value;
    }
}
```

4.3.4 Action column (eAction)

It is possible to add an action-column for the device/channels. This may be desirable when intending to have a configuration re-imported from the device at the push of a button. In order for this column to appear in the user interface, it must also be added to the associated table by means of the Table description tool window.

The source code already contains examples pertaining to the parameter **eAction**, which are commented-out

GetDeviceDescription

```
device.Parameters.UpdateOrAdd(EClassID.eAction, value: 1, editorType: EEditorType.ListButton);
```

Subsequently, values must be written to this selection list:

GetParameterConstraints

Add new elements with corresponding ID:

```
if (parameter.Name == EClassID.eAction.ToString()) {
    constraint.ValueList.Add(1);
    constraint.ValueList.Add(2);
    constraint.ValueList.Add(3);
}
```

GetParameterDisplayText

In the selection list, the corresponding texts are listed according to their ID.

```
if (parameter.Name == EClassID.eAction.ToString()) {
    if ((int)value == 1) {
        return "Edit configuration";
    } else if ((int)value == 2) {
        return "Import configuration";
    } else if ((int)value == 3) {
        return "Export configuration";
    }
}
```

ExecuteParameter

Here you determine which actions are to be performed in response to a particular selection.

```
switch (actionStepID) {
    case 1:
        MessageBox.Show("Edit Configuration");
        // do something
        break;
    case 2:
        MessageBox.Show("Import Configuration");
        // do something
        break;
    case 3:
        MessageBox.Show("Export Configuration");
        // do something
        break;
}
```

Once the column has been successfully added to the table in the user interface, it is operational when the corresponding device is selected. The associated action is executed upon pushing the button.

4.3.5 Data transfer

When creating a third-party script, you are also provided with a data transfer template in the `DataTransferDriver.cs`.

The next issue to address is whether the target device is polled (see [SimplePollDevice](#)^[152]), or whether it causes a function call (*callback*) when it has data (see [SimplePushDevice](#)^[155]).

The `SimplePollDevice` and the `SimplePushDevice` are good examples of such Third Party Device.

For further information, please read [DeviceDriver and PushDeviceDriver](#)^[146].

Access to the logbook

In the `DataTransferDriver` you can access to the logbook via the `Environment` class.

```
Environment.Logbook.LogEntry("MySender", "MyMessage", 0, ELogbookEntryCategory.Warning);
```

4.4 3rd Party Device Management

Using the **3rd-Party Device Management**, it is possible to select executable 3rd-party device script templates, e.g. the `Function-Simulator`, the `SimplePollDevice` or the `SimplePushDevice`. Likewise, there are already pre-made 3rd-party devices. Along with the `AudioDevice` and the `ChannelLoader`, the `fos4x` is also available. `fos4x` requires an extra license.

 Note

- The 3rd-Party Device Management is available in the editions *Developer*, *Professional* and *Standard*.
- For the use of 3rd-party devices, an activated **imc STUDIO 3PDI-Inclusive** or **imc STUDIO 3PDI-Exclusive** license is required.
- The **AudioDevice**, e.g. the audio interface **2-channel IEPE/ICP-signal conditioner (FH/ICP2-USB)**, the **ChannelLoader** and the **FunctionSimulator** can be used without any **3PDI** license.
- For the templates a **3PDI-license** is needed.
- For the fos4x device an **imc STUDIO 3PDI-fos4x** license is required.

Ribbon	View
Setup-Control > 3rd Party Device Management (🔧)	Complete

Select script to use

Available scripts

Third party devices

AudioDevice

ChannelLoader

fos4x

FunctionSimulator

Device script templates

FunctionSimulatorTemplate

SimplePollDevice

SimplePushDevice

User-defined Third party device scripts

Current active Third party device scripts

Browse

Script details

Name

Storage scope

Back

Next

Cancel

3rd Party Device Management Dialog

When selecting a script-template, there is an option for subsequently opening the Script-Editor.

User-defined scripts can be added by clicking on "Browse".

In "Current active Third party device scripts" all Third Party Device Scripts are listed and can be deleted by selecting and clicking on "Next".

© 2026 imc Test & Measurement GmbH

imc STUDIO 2026 - Programming interface - Manual, Doc. Rev.: 9.1 - 2026-08-27

Page 138

AudioDevice

In order to use connected Audio-devices, select the device-script *AudioDevice* and click on *Next*.

Subsequently, all Audio-devices appear in the device list. When one of the Audio-devices is selected, the computer's Audio-input channels appear on the page *Analog Channels*. In this way, when a measurement starts, the computer's Audio-inputs can be graphed and recorded.

For more information on this subject, see the chapter [AudioDevice](#) ¹⁴⁰.

ChannelLoader

The *ChannelLoader* serves to play back data already recorded. To do this, select the *ChannelLoader* and click on *Next*. Subsequently, the device *ChannelLoader* appears in the device list. When *ChannelLoader* is selected, a file selection dialog with multi-selection appears. Here, multiple already-recorded data or imc FAMOS-data sets can be selected. On the page *Analog Channels*, the corresponding channels appear. When a measurement starts, then the selected data are played back cyclically at their respective sampling rates.

For more information on this subject, see the chapter [ChannelLoader](#) ¹⁴¹.

Device script templates

The devices from the script-templates simulate a sine-signal, or in the case of the *FunctionSimulator*, a number of different signals (trapezoid, sawtooth, rectangle, ...).

4.5 Complete third party devices

Section	Description
AudioDevice ¹⁴⁰	Access to input channels of audio devices (sound card, microphone, ...)
ChannelLoader ¹⁴¹	Playback device for replay measurement data.
fos4x ¹⁴²	Fos4x devices
FunctionSimulator ¹⁴⁴	FunctionSimulator

4.5.1 AudioDevice

It is possible to add the audio devices of your computer to imc STUDIO. Therefore open the 3rd Party Device Management and add the AudioDevice. Afterward all audio devices are listed in the device list.

Documentation	Devices	Analog channels	Digital channels	GPS	Variables
Selected	Device name	SN	Device specification		
☐	Headset Microphone (Microsoft LifeChat LX-3000) -1	1518 - 1	Audio Device		
☐	Mikrofon (High Definition Audio-Device) -0	1518 - 0	Audio Device		

Device list: AudioDevice (Example)

On the **analog channel** page the audio channels are listed and also can be saved to the PC.
If multiple channels are available for a device, it is only possible to set one sample frequency for each channel of the device.

Documentation	Devices	Analog channels	Digital channels
Name	Connector	Status	Sampling & Filter
Channel type: Analog inputs (Count=2)			
(High Definition Audio-Device - Audiochannel 0)	AudioDevice 0	Active	88.2 kHz -
Microsoft LifeChat LX-3000 - Audiochannel 1	AudioDevice 1	Active	44.1 kHz -

Channel list: AudioDevice (Example)

 **Note**

- If the audio device is disconnected during the measurement, a error for each audio channel will appear in the logbook, showing that the data transfer is interrupted for this channel. To continue the measurement of this channel, start the audio device by the device control action in the Device table on the Setup page.
- If the experiment is opened on another computer, the corresponding hardware (USB device, sound card) must be available.
- If necessary, check your firewall for settings or rules: the process "imc.Studio.exe" (Program folder) needs the rights to access to the audio device.

 **Note**

Remarks to certain devices

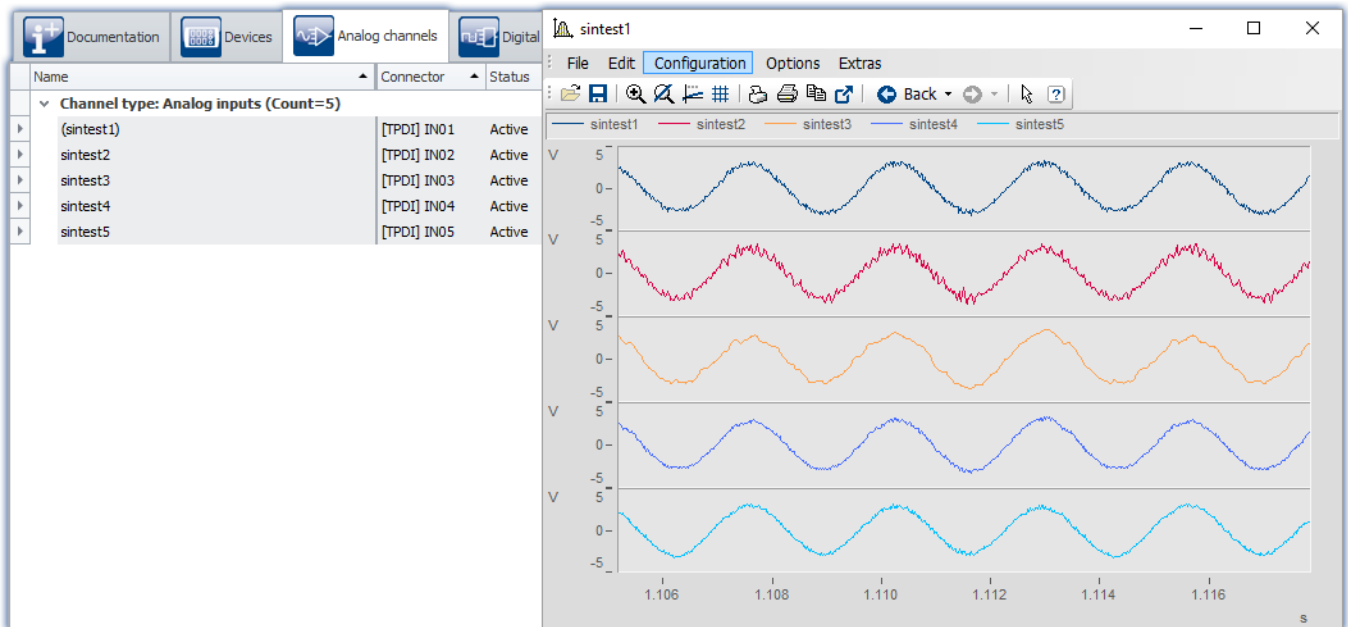
- With the webcam **Logitech HD Pro C920**, it was noticed that it was only delivering one audio channel, although according to the Windows settings there should be two channels.
This may also apply to other audio devices.
- With the Headset **Microsoft LifeChat LX-3000**, it was noticed that after disconnecting and reconnecting the USB cable, a new measurement is only possible after the *second* start.

4.5.2 ChannelLoader

The ChannelLoader is for simulating or replaying data (e.g. recorded measurement data or imc FAMOS files. It can be added using the 3rd Party Device Management.

After selecting the ChannelLoader a file-open-dialog appears where one or more files can be selected. After the preparation the files are listed as channels on the analog channels pages.

After starting the device the signals are played periodically as input signal.



ChannelLoader: sintest1 - sintest5

Missing data

If the file to be played back is missing, an error message appears which provides the expected path and filename of the missing file.

The 3rd-party device uses experiment-specific files

When a file is selected which is located below the Experiment-folder, the location is saved in "relative" terms with respect to the experiment. If the location of the experiment changes, the file will still be found as long as it is also in the new experiment. The file from the old experiment-folder is not us.

Example: You create a dat-file in the "Meta"-folder. If you now export the experiment and import it to a different device or under a new name, the "Meta"-folder is also present there (together with the dat-file). By means of the relative path, the copied dat-file is found and used.

Note: Other folder names can also be used. See the advantage of the "Meta" folder.

Whether the path is saved in relative or absolute terms depends on the relationship to the Experiment-folder. If the file is below the Experiment-folder, the path is "relative". If the file is outside of the Experiment-folder, the path is "absolute".

4.5.3 fos4x

It is possible to use **fos4x** devices in imc STUDIO as a 3rd-Party-Device. These devices can be configured and controlled via the imc STUDIO.



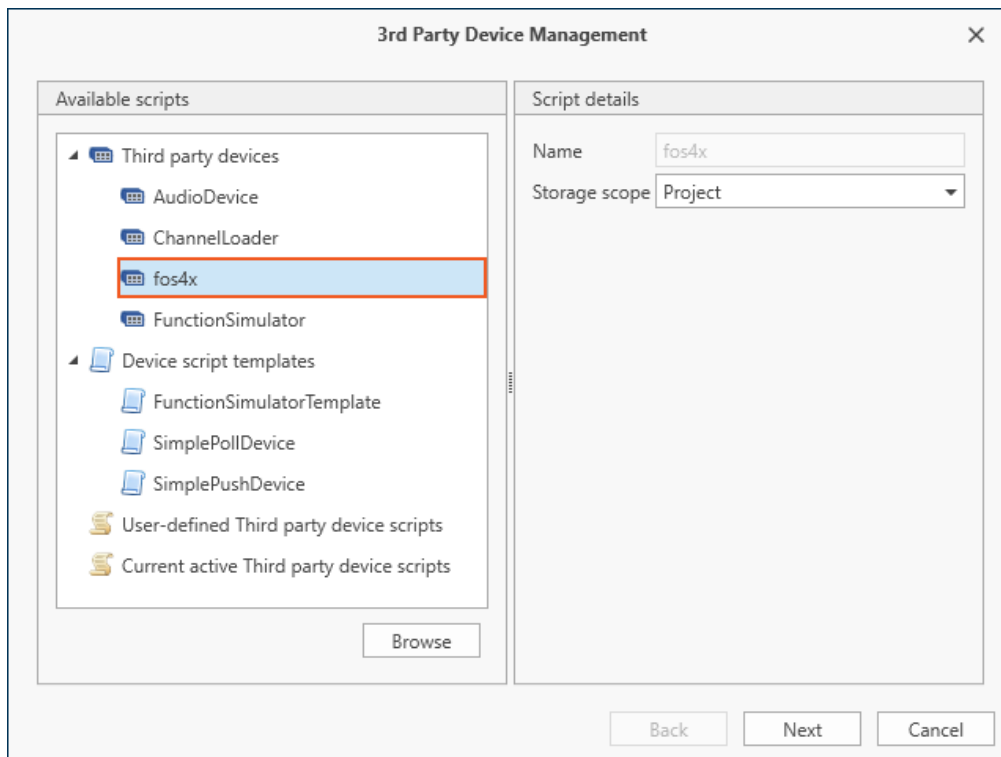
Note

License activation

To use a fos4x-Device the license imc STUDIO 3PDI-fos4x is required.

Adding a device - Integrating a fos4x-Device

Go to the Setup page. In the menu ribbon, click on *Setup Control > 3rd Party Device Management* and select *fos4x*.



Third-Party-Device Management

4.5.3.1 Device settings

After the device has been added to the Devices list by means of the *3rd Party Device Management*, the device can be configured.

If a **fos4x nSens** is selected, the following dialog will be opened:

Connector Configuration Dialog [nSens]

Connectors

Connector[1]

Connector[2]

Connector[3]

Connector[4]

Sensor(FBG) Name	Lower Wavelength Boundary	Upper Wavelength Boundary
Sensor(FBG)[1]	1525,443	1569,330
Sensor(FBG)[2]	1525,443	1530,008
*		

Record 1 of 2

Clear

Ok

Cancel

Configuration dialog (fos4x nSens)

!

Note

Verification of the measurement range

If the expected measurement value is either at the measurement range boundary or even outside of the range possible, the resulting bandwidth will not be practicable because imc STUDIO automatically applies a headroom of approx. 5 nm. Therefore, overwrite the settings later, in order to ensure an adequately large bandwidth. If this isn't possible, change the location of the measurement value on the fos4X page.Seite.

4.5.3.2 Channel settings

On the "Analog channels" page there is a new dialog "**fos4x**".

Channel definition

Measurement mode

Range & Scaling

Filtering

Sampling & Preprocessing

Data transfer

Curve properties

fos4X

Channel name

Channel_001_fos4Test_dyn_expert_52002

Signal quality

Good

Reference wavelength λ_0

1551.5 nm

Sensor unit $[X]$

Temperature [°C]

☐ Non-linear characteristic curve

$X = X_0 + 1/k_X [\lambda - \lambda_0]$

Parameters

Reference temperature X_0

23 °C @ λ_0

Sensitivity k_X

38.9737e-3 nm/°C Δ 25.12 ppm/°C

Balancing

Tare set point X_t

23 °C

not balanced

Tare

Compensation

0 °C (0 nm); -

Time

Analog channels (fos4x)

Reference

For further information, see **fos4x** data sheet or **fos4x** manual.

4.5.4 FunctionSimulator

The FunctionSimulator is device which simulates various kinds of mathematics functions, including sine, cosine, triangle, sawtooth and squarewave.

The analog channels then appear on the Setup-page under "Analog channels".

Documentation		Devices		Analog channels	
Name			Connector	Status	
▼ Channel type: Analog inputs (Count=8)					
Cosine			cos	Active	
DC			dc	Active	
Noise			noi	Active	
Rectangle			rect	Active	
Sawtooth			saw	Active	
Sine			sin	Active	
Trapeze			tra	Active	
Triangle			tri	Active	

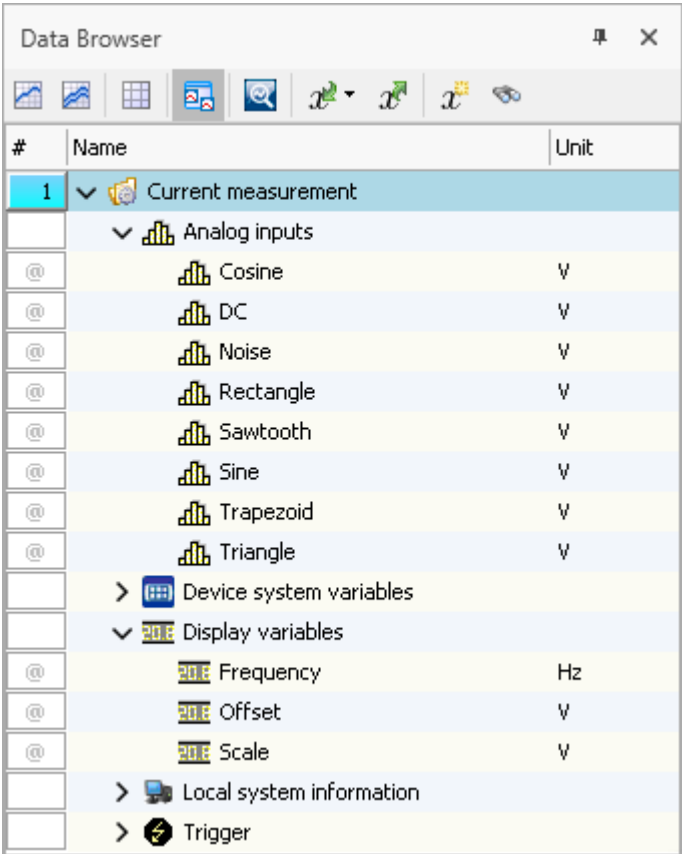
Setup: FunctionSimulator (Analog Channels)

It is also possible to adjust the frequency, the scaling and the offset. The Display variables appear on the Setup page under "Variables".

 Documentation		 Devices		 Analog channels		 Digital channels		 GPS		 Variables			
Channel name ▲				Connector ▲		Comment		Status		Unit		Data type	
▼ Channel type: Display-variables (Count=3)													
Frequency				Frequency		Active		Hz		64-bit double			
Offset				Offset		Active		V		64-bit double			
Scale				Scale		Active		V		64-bit double			

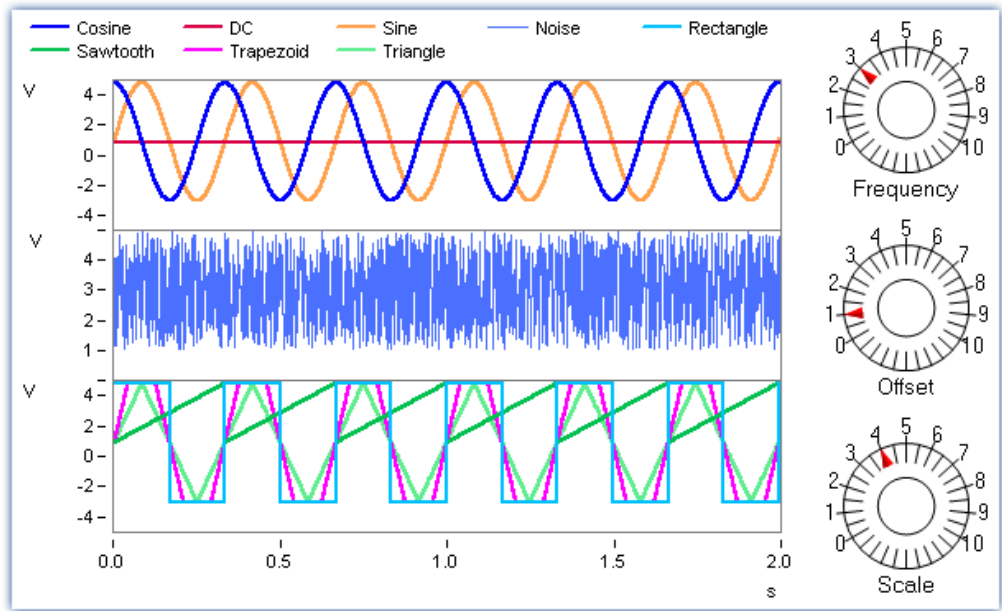
Setup: FunctionSimulator (Variables)

After selection and configuration of the device, the channels and Display variable are listed in the Data Browser.



Data Browser: FunctionSimulator

The signals can be displayed in the curve window.



Curve window: FunctionSimulator

By changing the Display variables, it is possible to alter the frequency, amplitude and offset during runtime.

Note

- The FunctionSimulator can be used as trigger target action of an imc device.
- Important: The time zone in the imc device must be corresponding to the PC's time zone.

4.6 Device script templates

Section	Description
DeviceDriver and PushDeviceDriver  146	Common component for data transfer.
FunctionSimulator  147	Device for simulating signals (sine, cosine, rectangle, triangle, ...)
SimplePollDevice  152	Template for a device getting data via <i>polling</i> .
SimplePushDevice  155	Template for a device which runs a <i>callback</i> function when measurement data is available.

Note

Using different third party devices based on the same template

If you want to use different third party devices based on the same template, please read the following instructions:

- Select a different name, when using the template in the Third-Party-Device management
- Rename the class name via keyboard (CTRL + R)
- Change the AssemblyName in the C# project settings.

If this note is not considered, only one device will be shown in the device list.

4.6.1 DeviceDriver and PushDeviceDriver

Note

- Here, the virtual device classes which simulate data for the **examples** (SimplePollDevice, SimplePushDevice and FunctionSimulator) are described.
- They can be used for making one's own Third Party Device.
- To write an unmanaged Data-Transfer-Driver, you need some header files that can be provided to you on request.

DeviceDriver

For the Third Party Devices FunctionSimulator and the SimplePollDevice, there is a device class called DeviceDriver available in the Third Party Device script. It describes a virtual device which is used and governed by the respective Third Party Device Script.

In this class, there is a description of the device and its channels, as well as of methods for retrieving simulated data.

One important method here is ReadData(), by means of which you can access the virtual device's data from the DataTransferDriver. In this method, the corresponding mathematics functions (sine, cosine, trapezoid, ...) are calculated and returned in double-Array format.

PushDeviceDriver

For the device SimplePushDevice, there is a class PushDeviceDriver, which is derived from the DeviceDriver. When the object is created, a callback-method must be passed, which is called when data are available in the device.

```
internal class PushDeviceDriver : DeviceDriver {
    public PushDeviceDriver(DataTransferEnvironment environment, DataCallback dataCallback)
        : base(environment) {

        m_dataPumpTimer = new Timer(DataPumpProc, null, Timeout.Infinite, Timeout.Infinite);
        m_dataCallback = dataCallback;
    }
    // ...
}
```

In the simulation, by means of a timer, the method DataPumpProc() is implemented, which passes the data accordingly.

```
private DataCallback m_dataCallback = null;

private void DataPumpProc(object state) {
    if (Monitor.TryEnter(this)) {
        try {
            Synchronize();

            foreach (Channel channel in m_channels.Values) {
                double[] data = channel.ReadData(); // read data from device
                if (data != null && m_dataCallback != null) {
                    m_dataCallback(channel.Name, new DataEventArgs(DataEvent.Data, data));
                }
            }
        } finally {
            Monitor.Exit(this);
        }
    }
}
```

4.6.2 FunctionSimulator template

The FunctionSimulator is device which simulates various kinds of mathematics functions, including sine, cosine, triangle, sawtooth and squarewave.

It is also possible to adjust the frequency, the scaling and the offset.

The analog channels and Display variables are set in the device/channel descriptions.

GetAvailableDevice

```
protected override void GetAvailableDevices(SetupNodeList deviceList) {
    // Add the currently available devices here! Example:
    var device = deviceList.GetOrAdd(ESetupNodeType.Device, "FunctionSimulator");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceNickname, value: "Function Simulator");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceSN, value: "");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceProduct, value: "");
}
```

GetDeviceDescription

```
protected override void GetDeviceDescription(SetupNode device) {
    // Add the full device description here! Channels, modules...

    AddDisplayVariable(device, "Scale", "Scale", "V");
    AddDisplayVariable(device, "Frequency", "Frequency", "Hz");
    AddDisplayVariable(device, "Offset", "Offset", "V");

    AddChannel(device, FunctionType.Sine.ToString(), "sin");
    AddChannel(device, FunctionType.Cosine.ToString(), "cos");
    AddChannel(device, FunctionType.Rectangle.ToString(), "rect");
    AddChannel(device, FunctionType.Sawtooth.ToString(), "saw");
    AddChannel(device, FunctionType.Triangle.ToString(), "tri");
    AddChannel(device, FunctionType.Noise.ToString(), "noi");
    AddChannel(device, FunctionType.DC.ToString(), "dc");
    AddChannel(device, FunctionType.Trapezoid.ToString(), "tra");
}
```

The capabilities of the FunctionType are stated in the [Data transfer](#)¹³⁷.

To make it easier to add analog channels, the function AddFunctionChannelToDevices() is provided here.

```
private void AddChannel(SetupNode device, string functionName, string functionNameShort) {
    var channel = device.Children.GetOrAdd(ESetupNodeType.Channel, functionName);
    channel.Parameters.UpdateOrAdd(EClassID.eChannelName, value: functionName);
    channel.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: functionNameShort);
    channel.Parameters.UpdateOrAdd(EClassID.eStatus, value: ParameterValues.eStatus.Active);

    channel.Parameters.UpdateOrAdd(EClassID.eChannelType, value:
ParameterValues.eChannelType.AnalogInput);
    channel.Parameters.UpdateOrAdd(EClassID.eXFormat, value: ParameterValues.eXFormat.Normal);
    channel.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.IEEEDOUBLE);

    channel.Parameters.UpdateOrAdd(EClassID.eSampleTime, value: 0.001, editorType:
EEditorType.ComboBox); // 1 kHz (default value)

    channel.Parameters.UpdateOrAdd(EClassID.eDivisionUnit, value: "V"); // The original (electrical)
measurement unit
    channel.Parameters.UpdateOrAdd(EClassID.eUserUnit, value: "V");

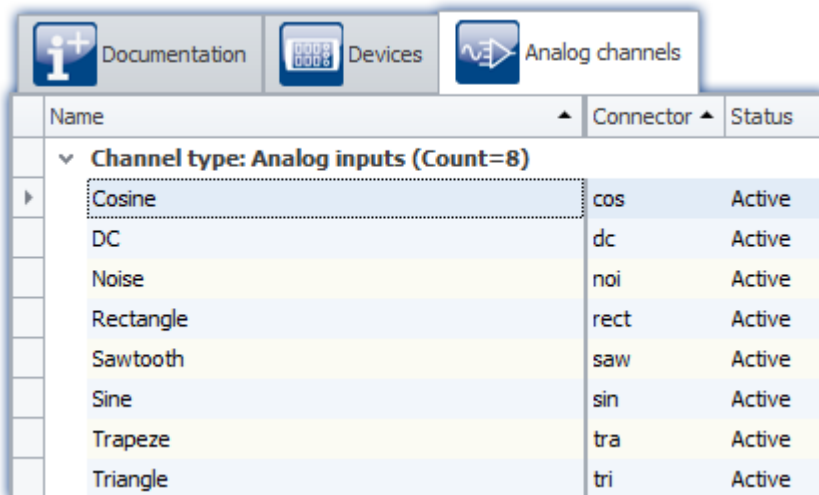
    channel.Parameters.UpdateOrAdd(EClassID.eExternalTriggerTarget, value: true);
}
```

The frequency, the scaling and the offset can be implemented with the help of the method AddDisplayVariableToDevice().

```
private void AddDisplayVariable(SetupNode device, string variableName, string variableNameShort, string
unit) {
    var channel = device.Children.GetOrAdd(ESetupNodeType.Channel, variableName);
    channel.Parameters.UpdateOrAdd(EClassID.eChannelName, value: variableName);
    channel.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: variableNameShort);
    channel.Parameters.UpdateOrAdd(EClassID.eStatus, value: ParameterValues.eStatus.Active);
    channel.Parameters.UpdateOrAdd(EClassID.eChannelType, value:
ParameterValues.eChannelType.DisplayVar);
    channel.Parameters.UpdateOrAdd(EClassID.eDivisionUnit, value: unit); // The original (electrical)
measurement unit
    channel.Parameters.UpdateOrAdd(EClassID.eUserUnit, value: unit);
    channel.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.IEEEDOUBLE);

    channel.Parameters.Remove(EClassID.eSampleTime);
    channel.Parameters.Remove(EClassID.eDuration);
    channel.Parameters.Remove(EClassID.ePCTransferToHD);
}
```

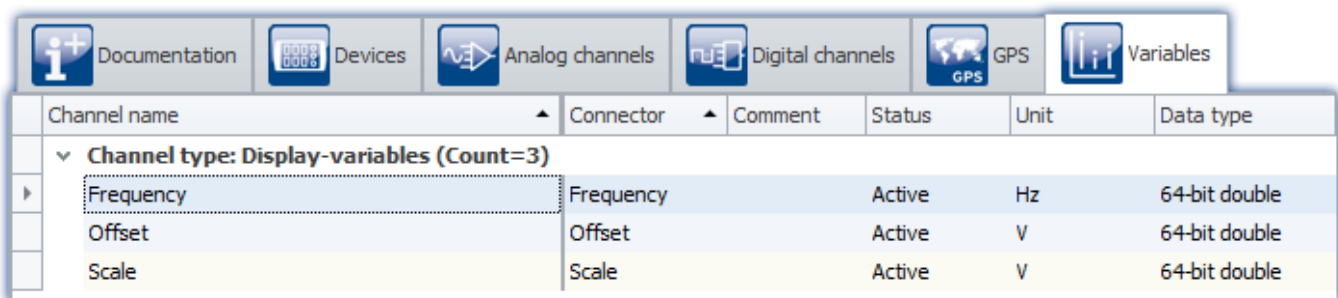
The analog channels then appear on the Setup-page under "Analog channels".



Name	Connector	Status
Channel type: Analog inputs (Count=8)		
Cosine	cos	Active
DC	dc	Active
Noise	noi	Active
Rectangle	rect	Active
Sawtooth	saw	Active
Sine	sin	Active
Trapeze	tra	Active
Triangle	tri	Active

Setup: FunctionSimulator (Analog Channels)

It is also possible to adjust the frequency, the scaling and the offset. The Display variables appear on the Setup page under "Variables".



Channel name	Connector	Comment	Status	Unit	Data type
Channel type: Display-variables (Count=3)					
Frequency	Frequency		Active	Hz	64-bit double
Offset	Offset		Active	V	64-bit double
Scale	Scale		Active	V	64-bit double

Setup: FunctionSimulator (Variables)

DataTransferDriver.cs

The DataTransferDriver handles the data transfer from Third Party Device to imc STUDIO.

For this purpose, the Third Party Device-object must be known to the DataTransferDriver. With the help of this object, it is then possible to control the Third Party Device.

In this example, upon initialization the corresponding object [DeviceDriver](#)¹⁴⁶ is created.

```
public class DataTransferDriver : AbstractDataTransferDriver {
// ...
    private DeviceDriver m_deviceDriver = null; // device object

    protected override ILogbookEntryBase[] Initialize(DataTransferEnvironment environment) {
        m_deviceDriver = new DeviceDriver(environment); // initializing device
        return null;
    }
// ...
}
```

Upon performing a Prepare action, the channel settings are applied and transferred to the device.

```
protected override ILogbookEntryBase[] Prepare(SetupNode configuration, DataTransferNodeList
dataTransferNodes) {
// ...
    m_deviceDriver.ConfigureChannel(child.Name, FunctionType.Sine, (double)sampleTime.Value); // setting
channels in the device
// ...
}
```

Additionally, the Display variables are set.

```

if (dataTransferNode.Name == "Frequency") {
    dataWriter.WriteData(1.0);
} else if (dataTransferNode.Name == "Offset") {
    dataWriter.WriteData(0.0);
} else if (dataTransferNode.Name == "Scale") {
    dataWriter.WriteData(1.0);
}

```

When starting and stopping, the datWriter is started/stopped.

During measurement, the data are continually polled by the device via a timer and exported to imc STUDIO by the dataWriter. Likewise, the Display variables are imported and processed.

```

public DataTransferDriver() {
    m_processDataTimer = new Timer(OnProcessData, null, Timeout.Infinite, Timeout.Infinite);
}

private void OnProcessData(object state) {
    // ...
    // processing Display variables
    if (dataTransferNode.Name == "Frequency") {
        if (dataReader.ReadData(out frequency) == EResult.Success) {
            if (frequency <= 0.1) {
                frequency = 0.1;
                dataWriter.WriteData(frequency);
            } else if (frequency >= 10000.0) {
                frequency = 10000.0;
                dataWriter.WriteData(frequency);
            }
        }
    } else if (dataTransferNode.Name == "Offset") {
        if (dataReader.ReadData(out offset) == EResult.Success) {
            if (Math.Abs(offset) > 1e20) {
                offset = Math.Sign(offset) * 1e20;
                dataWriter.WriteData(offset);
            }
        }
    } else if (dataTransferNode.Name == "Scale") {
        if (dataReader.ReadData(out scale) == EResult.Success) {
            if (Math.Abs(scale) > 1e20) {
                scale = Math.Sign(scale) * 1e20;
                dataWriter.WriteData(scale);
            }
        }
    }

    // data retrieved from the device and exported to STUDIO
    double[] data = m_deviceDriver.ReadData(dataTransferNode.Name);
    EResult result = dataWriter.WriteData(data);
    //...
}

```

DeviceDriver.cs

For details on this topic, see the chapter [DeviceDriver and PushDeviceDriver](#) 146.

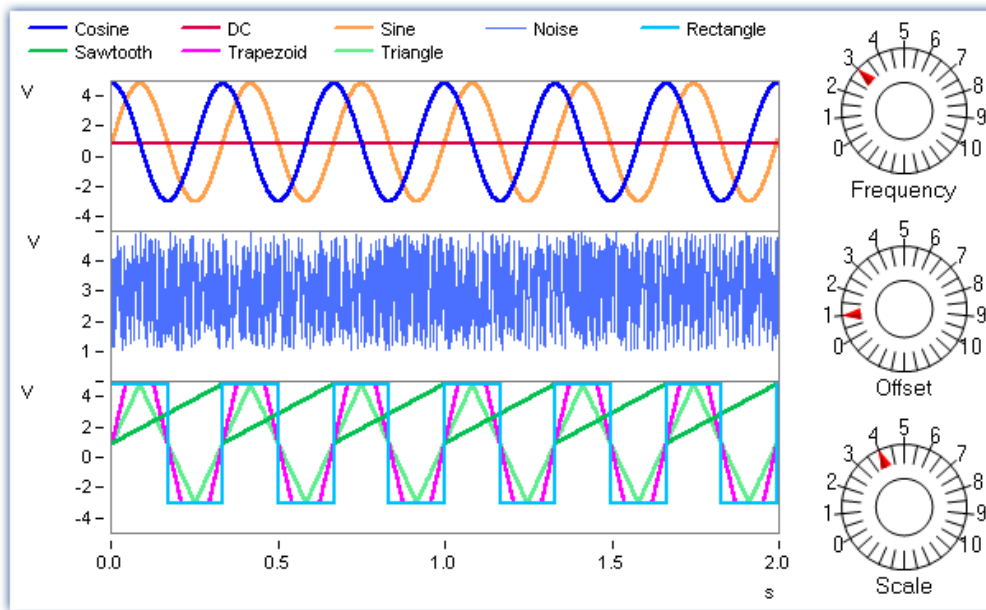
Functionality in imc STUDIO

After selection and configuration of the device, the channels and Display variable are listed in the Data Browser.

#	Name	Unit
1	Current measurement	
	Analog inputs	
@	Cosine	V
@	DC	V
@	Noise	V
@	Rectangle	V
@	Sawtooth	V
@	Sine	V
@	Trapezoid	V
@	Triangle	V
	Device system variables	
	Display variables	
@	Frequency	Hz
@	Offset	V
@	Scale	V
	Local system information	
	Trigger	

Data Browser: FunctionSimulator

The signals can be displayed in the curve window.



Curve window: FunctionSimulator

By changing the Display variables, it is possible to alter the frequency, amplitude and offset during runtime.

Note

- The FunctionSimulator can be used as trigger target action of an imc device.
- Important: The time zone in the imc device must be corresponding to the PC's time zone.

4.6.3 SimplePollDevice

One simple example of Third Party Device is the "SimplePollDevice", which generates a sine signal.

GetAvailableDevices

```
protected override void GetAvailableDevices(SetupNodeList deviceList) {
    // Add the currently available devices here! Example:
    var device = deviceList.GetOrAdd(ESetupNodeType.Device, "SimplePollDevice");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceNickname, value: "SimplePollDevice");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceSN, value: "");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceProduct, value: "");
}
```

GetDeviceDescription

In the device description, the channel information is presented. In this case, the channel is an analog channel named "Poll_Input_001", having a pre-set sampling interval of 0.01 seconds.

```
protected override void GetDeviceDescription(SetupNode device) {
    // Add the full device description here! Channels, modules...
    //
    // Example code (equidistant channel with a float sample every 10ms):
    var channel = device.Children.GetOrAdd(ESetupNodeType.Channel, "Poll_Input_001");
    channel.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "Poll_Input_001");
    channel.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "IN01");
    channel.Parameters.UpdateOrAdd(EClassID.eStatus, value: ParameterValues.eStatus.Active);
    channel.Parameters.UpdateOrAdd(EClassID.eChannelType, value:
ParameterValues.eChannelType.AnalogInput);
    channel.Parameters.UpdateOrAdd(EClassID.eDivisionUnit, value: "V"); // The original (electrical)
measurement unit
    channel.Parameters.UpdateOrAdd(EClassID.eUserUnit, value: "V");

    channel.Parameters.UpdateOrAdd(EClassID.eXFormat, value: ParameterValues.eXFormat.Normal);
    channel.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.IEEEDOUBLE);
    channel.Parameters.UpdateOrAdd(EClassID.eSampleTime, value: 0.01, editorType:
EEditorType.ComboBox); // 100 Hz (default value)
}
```

The data transfer is handled by a special object of a class (*managed*).

```
public override IDataTransferDriverBase CreateDataTransferDriver() {
    return new DataTransferDriver();
}
```

DataTransferDriver.cs

The DataTransferDriver handles the transfer of data from the Third Party Device to imc STUDIO.

For this purpose, the Third Party Device-object must be known to the DataTransferDriver. With the help of this object, it is then possible to control the Third Party Device.

In this example, the corresponding object [DeviceDriver](#)¹⁴⁶ is created in the initialization process.

```
public class DataTransferDriver : AbstractDataTransferDriver {
    // ...
    private DeviceDriver m_deviceDriver = null; // Device-Object

    protected override ILogbookEntryBase[] Initialize(DataTransferEnvironment environment){
        m_deviceDriver = new DeviceDriver(environment); // Initializing device
        return null;
    }
    // ...
}
```

Upon performing a Prepare action, the channel settings are applied and transferred to the device.

```
protected override ILogbookEntryBase[] Prepare(SetupNode configuration, DataTransferNodeList
dataTransferNodes) {
    // ...
    m_deviceDriver.ConfigureChannel(child.Name, FunctionType.Sine, (double)sampleTime.Value); //
Setting channels in the device
    // ...
}
```

When starting and stopping, the dataWriter is started/stopped.

During measurement, the data are continually polled by the device via a timer and exported to imc STUDIO by the dataWriter.

```
public DataTransferDriver() {
    m_processDataTimer = new Timer(OnProcessData, null, Timeout.Infinite, Timeout.Infinite);
}

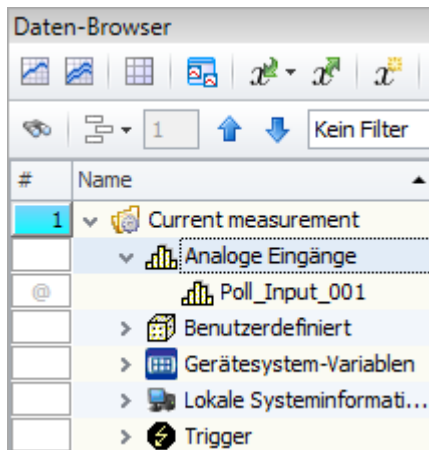
private void OnProcessData(object state) {
    // ...
    double[] data = m_deviceDriver.ReadData(dataTransferNode.Name); // retrieving data from the device
    EResult result = dataWriter.WriteData(data);
    //...
}
```

DeviceDriver.cs

For details, see the chapter [DeviceDriver and PushDeviceDriver](#) ¹⁴⁶.

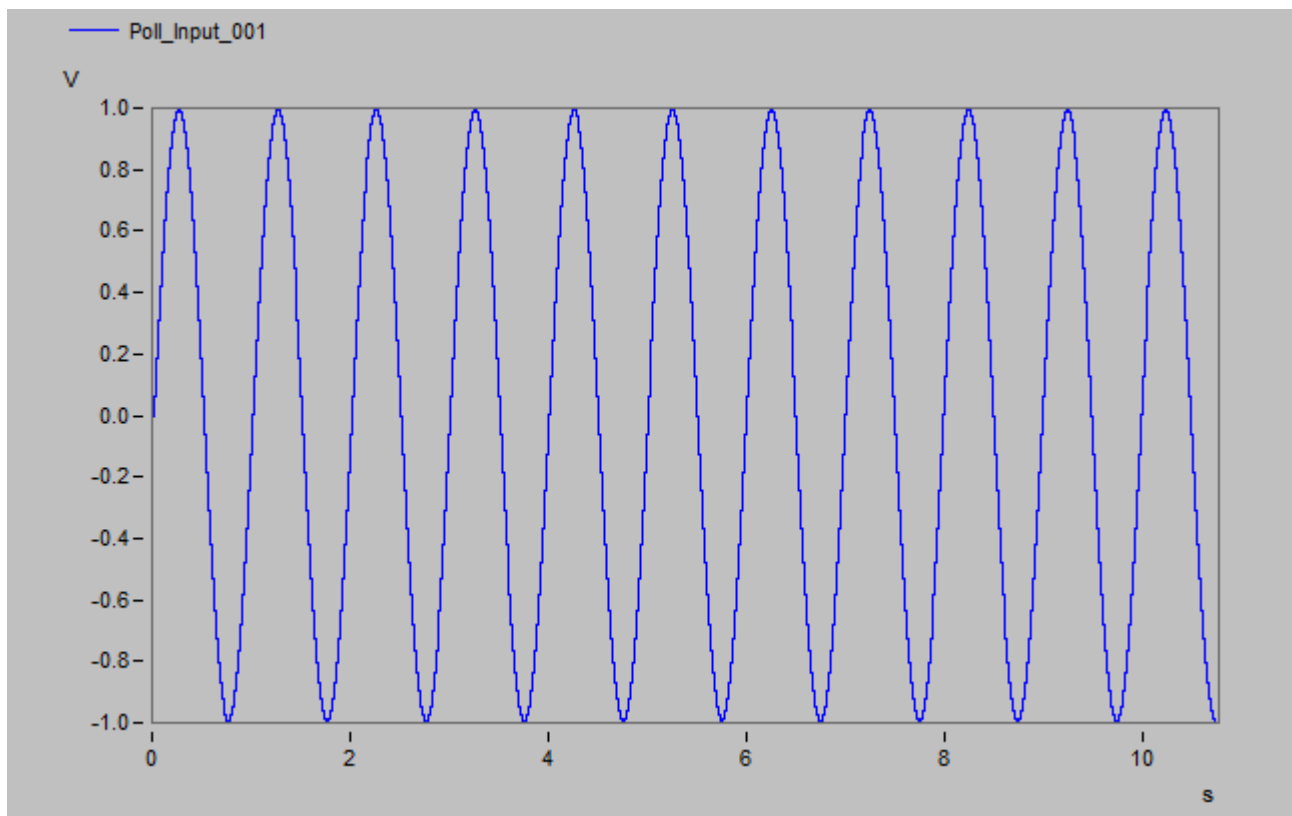
Functionality in imc STUDIO

After selection of the device and its configuration, the analog channel "Poll_Input_001" appears in the Data Browser.



Data Browser: SimplePollDevice

Once the measurement starts, the sine-signal generated can be displayed in a curve window.



Curve window: SimplePollDevice

4.6.4 SimplePushDevice

An additional example of Third Party Device is the "SimplePushDevice", which generates a sine-signal. In the process, the device initiates a callback.

GetAvailableDevices

```
protected override void GetAvailableDevices(SetupNodeList deviceList) {
    // Add the currently available devices here! Example:
    var device = deviceList.GetOrAdd(ESetupNodeType.Device, "SimplePushDevice");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceNickname, value: "SimplePushDevice");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceSN, value: "");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceProduct, value: "");
}
```

GetDeviceDescription

In the device description, the channel information is presented. In this case, the channel is an analog channel named "Input_001" with a pre-set sampling interval of 0.01 seconds.

```
protected override void GetDeviceDescription(SetupNode device) {
    // Add the full device description here! Channels, modules...

    // Example code (equidistant channel with a float sample every 10ms):
    var channel = device.Children.GetOrAdd(ESetupNodeType.Channel, "Push_Input_001");
    channel.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "Push_Input_001");
    channel.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "IN01");
    channel.Parameters.UpdateOrAdd(EClassID.eStatus, value: ParameterValues.eStatus.Active);
    channel.Parameters.UpdateOrAdd(EClassID.eChannelType, value:
    ParameterValues.eChannelType.AnalogInput);
    channel.Parameters.UpdateOrAdd(EClassID.eDivisionUnit, value: "V"); // The original (electrical)
    measurement unit
    channel.Parameters.UpdateOrAdd(EClassID.eUserUnit, value: "V");

    channel.Parameters.UpdateOrAdd(EClassID.eXFormat, value: ParameterValues.eXFormat.Normal);
    channel.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.IEEEDOUBLE);
    channel.Parameters.UpdateOrAdd(EClassID.eSampleTime, value: 0.01); // 100 Hz (default value)
}
```

The data transfer is handled by a special object of a class (*managed*).

```
public override IDataTransferDriverBase CreateDataTransferDriver() {
    return new DataTransferDriver();
}
```

DataTransferDriver.cs

The DataTransferDriver handles the data transfer from Third Party Device to imc STUDIO.

For this purpose, the Third Party Device-object must be known to the DataTransferDriver. With the help of this objects, it is then possible to control to the Third Party Device.

In this example, upon initialization the corresponding object [PushDeviceDriver](#)¹⁴⁷, which is described in the file DeviceDriver.cs, is created. The method OnProcessData is passed to the SimplePushDevice, and this method is called when data are finalized in the Third Party Device and can be transferred (*push*) to imc STUDIO.

```
public class DataTransferDriver : AbstractDataTransferDriver {
    // ...
    private PushDeviceDriver m_deviceDriver = null;

    protected override ILogbookEntryBase[] Initialize(DataTransferEnvironment environment) {
        m_deviceDriver = new PushDeviceDriver(environment, OnProcessData);
        return null;
    }
    // ...
}
```

Upon performing a Prepare action, the channel's settings are applied and transferred to the device.

```
protected override ILogbookEntryBase[] Prepare(SetupNode configuration, DataTransferNodeList
dataTransferNodes) {
// ...
    m_deviceDriver.ConfigureChannel(child.Name, FunctionType.Sine, (double)sampleTime.Value); //
    setting channels in the device
// ...
}
```

When starting and stopping, the dataWriter is started/stopped.

During the measurement, the method `OnProcessedData()` is called, if the Third Party Device itself has concluded it data package and calls this event.

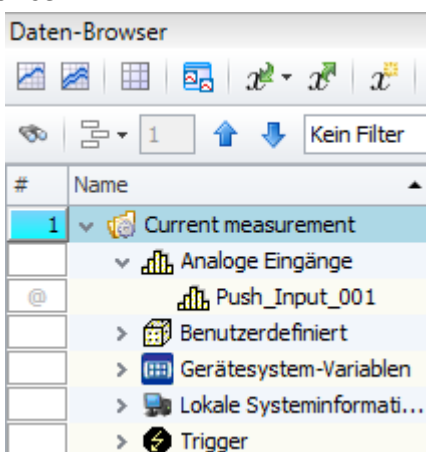
```
private void OnProcessData(string channelName, DataEventArgs e) {
// ...
    switch (e.DataEvent) {
        case DataEvent.Begin:
            dataWriter.Open();
            result = dataWriter.BeginWrite(e.Time.Value);
            // Error-Handling
            break;
        case DataEvent.Data:
            result = dataWriter.WriteData(e.Data);
            // Error-Handling
            break;
        case DataEvent.End:
            result = dataWriter.EndWrite();
            dataWriter.Close();
            // Error-Handling
            break;
        default:
            break;
    }
//...
}
```

DeviceDriver.cs

For more on this topic, see the chapter [DeviceDriver and PushDeviceDriver](#) ¹⁴⁶.

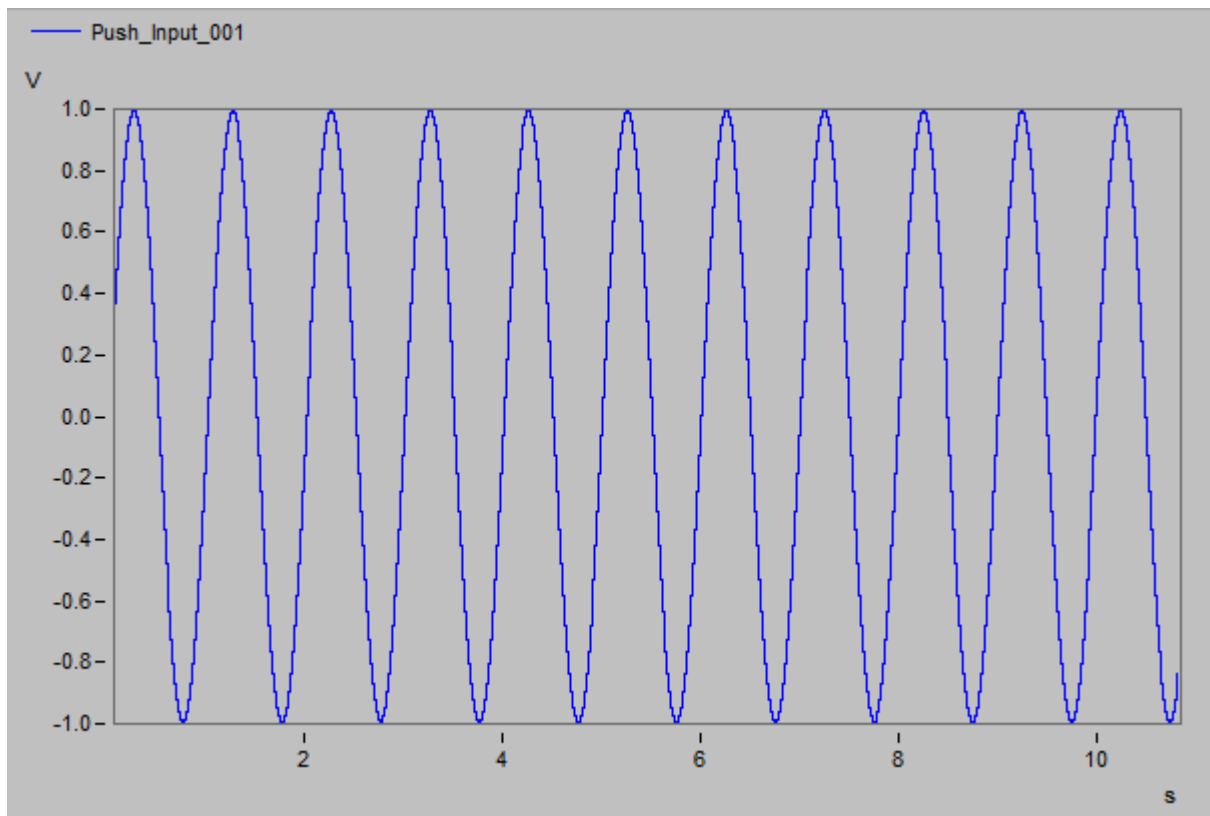
Functionality in imc STUDIO

After selection of the device and its configuration, the analog channel "Poll_Input_001" appears in the Data Browser.



Data Browser: SimplePushDevice

Once the measurement has been started, the sine-signal generated can be displayed in a curve window.



Curve window: SimplePushDevice

4.7 FAQ



FAQ

When importing an older experiment with a third party device, I get the error "System.IO.FileNotFoundException"

Answer: The given third party devices are imported as DLL. There are dependencies to other libraries. In the version 5.2 there are changes causing the following error message:

The Script "ChannelLoader" encountered: System.IO.FileNotFoundException: Could not load file or assembly 'imc.Studio.Interfaces.V2, Version=5.0.2.0, Culture=neutral, PublicKeyToken=68a4b1d388e6c0b6' or one of its dependencies. The system cannot find the file specified.
File name: 'imc.Studio.Interfaces.V2, Version=5.0.2.0, Culture=neutral, PublicKeyToken=68a4b1d388e6c0b6' [...]

To fix this error, please remove the corresponding third party device script (AudioDevice, ChannelLoader) with the help of the 3rd Party Device Management and add it again.

Contact the author of the third party device script if you have further questions.

4.8 Tutorial

Here you will find a list of tutorials:

Tutorial	Beschreibung
Add a channel 	A new channel is added to the SimplePollDevice.
Add a Display variable 	A Display variable for setting the signal frequency is added to the SimplePollDevice.

4.8.1 Extension of the SimplePollDevice

4.8.1.1 Add a channel

The task is to add an analog channel to the existing SimplePollDevice. For this purpose, a new experiment is created, and additionally the SimplePollDevice added to the device list via the 3rd Party Device Management, unless this already happened.



Note

While the changes are made, the SimplePollDevice may not be selected.

GetAvailableDeviceDevices

The device's name is changed to *SimplePollDeviceExtended*.

```
protected override void GetAvailableDevices(SetupNodeList deviceList) {
    // Add the currently available devices here! Example:
    var device = deviceList.GetOrAdd(ESetupNodeType.Device, "SimplePollDeviceExtended");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceNickname, value: "SimplePollDeviceExtended");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceSN, value: "");
    device.Parameters.UpdateOrAdd(EClassID.eDeviceProduct, value: "");
}
```

GetDeviceDescription

An additional channel having the same properties as the first channel is added. This segment can be copied and inserted. The new channel name is altered.

```
protected override void GetDeviceDescription(SetupNode device) {
    // Add the full device description here! Channels, modules... Example:
    var channel = device.Children.GetOrAdd(ESetupNodeType.Channel, "Poll_Input_001");
    channel.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "Poll_Input_001");
    channel.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "IN01");
    // ... code is omitted
    var channel2 = device.Children.GetOrAdd(ESetupNodeType.Channel, "Poll_Input_002");
    channel2.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "Poll_Input_002");
    channel2.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "IN02");
    // ... same settings as Input_001, code is omitted here
}
```

DataTransferDriver.cs

In the class `DataTransferDriver`, the `SimplePollDevice` is configured. In the method `Prepare()`, the channels are configured.

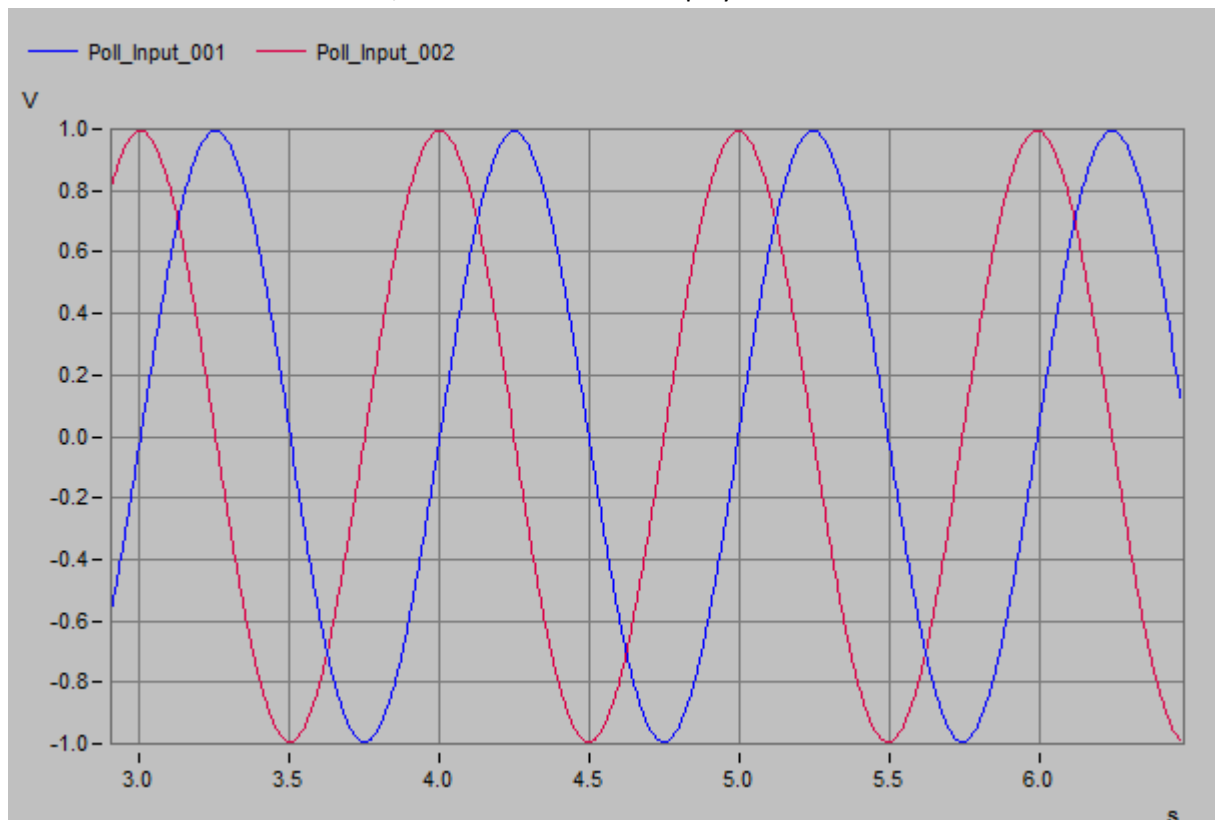
By default, sine-signals are generated for all channels. By polling the channel name, it is possible to set a different signal, e.g. cosine.

```
protected override ILogbookEntryBase[] Prepare(SetupNode configuration, DataTransferNodeList
dataTransferNodes) {
    // ...
    if (sampleTime != null && sampleTime.Value is double) {
        // Channel 1 : Sine
        if (child.Name == "Poll_Input_001"){
            m_deviceDriver.ConfigureChannel(child.Name, FunctionType.Sine, (double)sampleTime.Value);
        }

        // Channel 2 : Cosine
        if (child.Name == "Poll_Input_002"){
            m_deviceDriver.ConfigureChannel(child.Name, FunctionType.Cosine, (double)sampleTime.Value);
        }
    }
    //...
}
```

After selection of the device, the channels `Poll_Input_001` and `Poll_Input_002` appear in the list of analog channels, as well as in the Data Browser.

When a measurement is started, the channels can be displayed in the curve window.



Curve window: Extended SimplePollDevice

4.8.1.2 Add a display variable

A Display variable is to be added, which adjusts the frequency for the analog signal.

GetDeviceDescription

The following segment is added to the `GetDeviceDescription`-method. The Display variable "Frequency" is thus added to the device.

```
var displayVar = device.Children.GetOrAdd(ESetupNodeType.Channel, "Frequency");
displayVar.Parameters.UpdateOrAdd(EClassID.eChannelName, value: "Frequency");
displayVar.Parameters.UpdateOrAdd(EClassID.ePlugInName, value: "Freq");
displayVar.Parameters.UpdateOrAdd(EClassID.eStatus, value: ParameterValues.eStatus.Active);
displayVar.Parameters.UpdateOrAdd(EClassID.eChannelType, value:
ParameterValues.eChannelType.DisplayVar);
displayVar.Parameters.UpdateOrAdd(EClassID.eDivisionUnit, value: "Hz"); // The original (electrical)
measurement unit
displayVar.Parameters.UpdateOrAdd(EClassID.eUserUnit, value: "Hz");
displayVar.Parameters.UpdateOrAdd(EClassID.eDataType, value: ParameterValues.eDataType.IEEDDOUBLE);

displayVar.Parameters.Remove(EClassID.eSampleTime);
displayVar.Parameters.Remove(EClassID.eDuration);
displayVar.Parameters.Remove(EClassID.ePCTransferToHD);
```

DataTransferDriver.cs

In the class `DataTransferDriver`, the following code must be amended/edited in the method `OnProcessData()`.

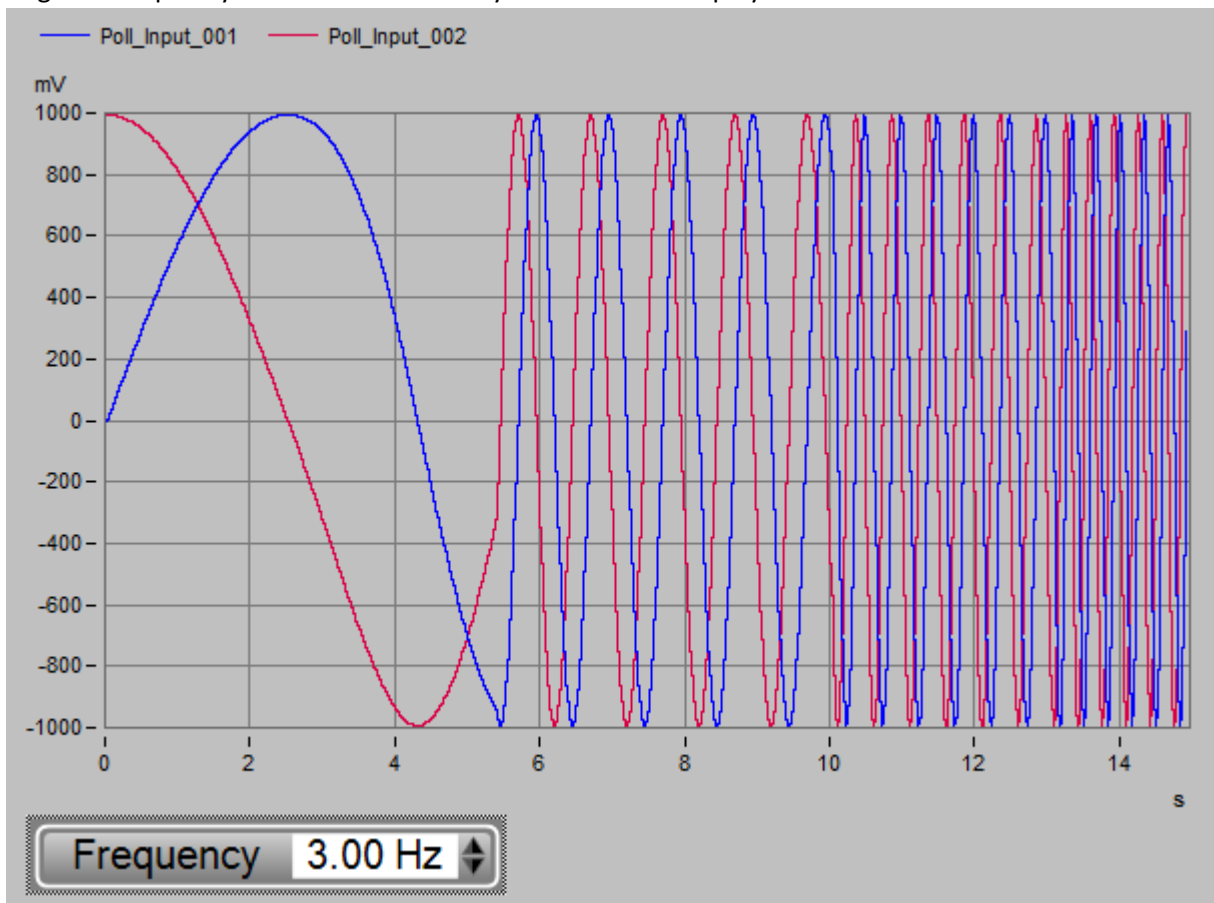
```
if (dataTransferNodes != null) {
    // defining parameters (inserting into existing framework)
    double frequency = 1.0;
    double offset = 0.0;
    double scale = 1.0;
    // ...
}

if (dataTransferNodes != null) {
    // ...
    // importing Display variable (foreach-block to be inserted before other foreach)
    foreach (DataTransferNode dataTransferNode in dataTransferNodes) {
        DataReader dataReader = dataTransferNode.Reader;
        if (dataReader != null) {
            DataWriter dataWriter = dataTransferNode.Writer;
            if (dataWriter != null) {
                if (dataTransferNode.Name == "Frequency") {
                    if (dataReader.ReadData(out frequency) == EResult.Success) {
                        if (frequency <= 0.1) {
                            frequency = 0.1;
                            dataWriter.WriteData(frequency);
                        } else if (frequency >= 10000.0) {
                            frequency = 10000.0;
                            dataWriter.WriteData(frequency);
                        } // frequency
                    } // ReadData
                } // Name = Frequency
            } // dataWriter
        } // dataReader
    } // foreach
    // ...
}

// the existing foreach-block must be edited
foreach (DataTransferNode dataTransferNode in dataTransferNodes) {

    DataWriter dataWriter = dataTransferNode.Writer;
    if (dataWriter != null) {
        // this line must be inserted for setting the frequency
        m_deviceDriver.SetTunableParameters(dataTransferNode.Name, frequency, offset, scale);
        double[] data = m_deviceDriver.ReadData(dataTransferNode.Name);
        // ...
    }
}
```

After closing the Editor and selecting the third-party device, the Display variable is displayed in the Data Browser. The signal's frequency can now be altered by means of the Display variable.



Adjustment of the amplitude and offset can be accomplished by adding more Display variables. On this subject, see also the [FunctionSimulator](#)¹⁴⁷.

5 API

The **imc STUDIO API** (ApplicationInterface) is a programming interface which provides both the ability to freely develop programs and also **fundamental functions** belonging to imc STUDIO, such as:

- reading and writing of device and channel settings (Setup) ,
- access to the Panel and the Widgets,
- access to the Data Browser: creating, reading and writing variables,
- applying of imc FAMOS-functions to channels,
- executing menu actions in the ribbon,
- executing Sequencer commands,
- reaction to events,
- Calculate channels using imc FAMOS functions (requires a valid license).

This manual explains certain introductory points and describes the interfaces with imc STUDIO.

Note

- The **imc STUDIO API** is provided as part of the installation of imc STUDIO *Developer*.
- The programming language is a .NET programming language (e.g. C#, Visual Basic .NET).
- The interface is compatible exclusively with .NET Core. The .NET Framework is no longer supported.

5.1 Licensing

The **imc STUDIO API** is marketed in conjunction with the **imc DATA API** as **imc API**.

To develop a project, the *imc API Developer* license is needed. To run the programs an *imc API Runtime* license is needed.

With **imc DATA API**, you obtain access to the functionalities of imc FAMOS, including of its Kits, the curve window and the data objects.

The *Base-Project* of the **imc STUDIO API** is installed on the computer as part of installation of the *imc STUDIO Developer*.

If functions from the *imc STUDIO Standard* Edition or higher are needed by **API**, a corresponding license must be activated via the License Manager.

Reference

For more information, see:

- [Installation](#)  163
- [Base Interface](#)  165

5.2 Installation

In the process of installing the **imc STUDIO Developer**, the primary project as well as extra sample projects are installed on this path:

"C:\Users\Public\Documents\imc\imc STUDIO\API 2026\" (or corresponding version)

The subfolder "Shared" contains the basic project which is required for making one's own projects.

The subfolder "Projects" contains executable sample projects.

Prerequisites for development based on the imc STUDIO API

- imc STUDIO Installation (Developer Edition)
- imc DATA API Installation (Developer Edition)
- Code editor / IDE for .NET (.NET Core / dotnet) with C# support
- .NET 8.0 (.NET Core / dotnet)
- .NET Framework 4.8 - Required for legacy code.
- Use of an WPF or Windows-Forms project

Prerequisites for using the imc STUDIO API as runtime

- imc STUDIO Installation (corresponding to the functionality)
- imc DATA API Installation (Runtime Edition)



It is recommended that you have only one imc STUDIO version installed at a time in order to avoid version conflicts with the assemblies.



Note

.NET Version

The calling application (*.exe) must be the same or higher .NET Framework version of the imc STUDIO API component.

imc STUDIO 2026 primarily uses **.NET 8.0** *.

The .NET Framework 4.8 is still required for older components (legacy code).



Reference

imc DATA API

You can find the Setup file of the **imc DATA API** in the installation package under \Products\Development\Setup\imc_DATA_API

- Setup_imcDataApiDeveloper.exe
- Setup_imcDataApiRuntime.exe

* It is installed along with imc STUDIO 2026 during installation.

5.3 Collaboration with the imc DATA API



Note

License

To use the imc DATA API, also the **imc API Developer** or **imc API Runtime** license is required.



Reference

You will find more information in the imc API manual.

5.4 Specifying the imc STUDIO version

Warning

In the development process it is recommended that there is only one imc STUDIO version installed to avoid conflicts with the assemblies.

When using the completed application also parallel installation is possible.

The interface attempts to automatically detect the version of imc STUDIO Version and enters it into the configuration file [default.ispc](#) ¹⁶⁴.

The following steps are necessary in order to use a particular version of imc STUDIO in the imc STUDIO API:

In the **default.ispc**-file used, for each entry in the assembly, the corresponding version number must be entered. This number can be obtained from the application path of the corresponding imc STUDIO version.



In the [Initialization](#) ¹⁶⁶ the parameter `correctVersionInISPC` must be set to `false`. Otherwise the `default.ispc` is overwritten.

Product config file: default.ispc

By default, the **default.ispc** for the API is located in the following folder: `C:\Users\Public\Documents\imc\imc STUDIO\API\Projects\AppData`

Notes on default.ispc

This file is loaded when an instance of imc STUDIO is started. The content indicates which components are to be loaded.

Along with the information whether a component is active, the file also contains the version number. This number must match the version number of the imc STUDIO installed; non-matching versions will cause the start to be canceled.

Therefore it is necessary to perform manual verification and, if applicable, alteration of the version:

1. Version of imc STUDIO installed:
 - a. see file properties of **imc.Studio.Version.dll** in the imc STUDIO installation folder
2. Version specification in the file **default.ispc**
 - a. To edit, the file can be opened in an XML- or text editor.
 - b. The installed version of imc STUDIO must be indicated in the XML-tag **Version**: e.g. the plug-in **imc.Studio.PlugIns.Panel.dll**
3. If the versions don't match or if there is no version indicated:
 - a. Change the text of the version within the tag `<Version>xxx</Version>`
 - b. save **default.ispc**
 - c. restart the program

Example of a plug-in component in default.ispc

```
<PlugIn>
  <PlugInType>DotNet</PlugInType>
  <StartupClass>imc.Studio.PlugIns.Panel._ThePlugIn</StartupClass>
  <Culture>neutral</Culture>
  <PublicKeyToken>68a4b1d388e6c0b6</PublicKeyToken>
  <AssemblyName>imc.Studio.PlugIns.Panel</AssemblyName>
  <Active>True</Active>
  <Version>23.0.1.6502</Version>
  <LoadPositionNumber>13</LoadPositionNumber>
</PlugIn>
```

5.5 Base interface

The project **imc.Studio.ApplicationInterface** is considered the Basis-project.

imc STUDIO has a core (imc STUDIO Core) having a framework onto which all functionalities sign on.

These functionalities only become operable in consequence of referencing various assemblies.

In order to make it easier to gain familiarity, the basis-project contains the most important references and provides simplification of these functionalities.

Note

- imc makes the source code of the basis-project available. It is highly recommended **not** to alter it. In future versions, the scope of functionalities will be expanded. Any changes would be lost upon updating and/or would need to be reconstructed.
- What is recommended is to create **one's own** imc STUDIO-class which is derived from `SimplifiedImcStudioComponent`. Here, it is possible to make any enhancements desired.

5.5.1 Initialization

Parameter

For the **initialization** of the imc STUDIO components, the following **parameters** must be provided:

```
imcStudioObject = new SimplifiedImcStudioComponent(<application folder>, <parent window>, <installation folder> )
```

Parameter	Description
Application folder	- In the standard case, imc STUDIO is started from its application folder, e.g. "C:\ProgramData\imc\imc STUDIO\Applications_1". This is where the configuration is saved in default.ispc and loaded. - When setting a new folder for your application, the folder "AppDir" (application settings) from the Demo-projects belonging to imc STUDIO can be used. - For the purpose of modifying the components used, the file default.ispc can be copied from the STUDIO-applications folder (once all all components to be used have been activated in the imc STUDIO user interface). - Alternatively, for testing purposes a complete copy of the folder ".._1\" can be used. - To check which application folder is used by the current installation of imc STUDIO, have the properties of imc STUDIO's desktop link displayed.
Parent window	- Necessary for displaying dialogs from the imc STUDIO component - Necessary for controlling the multi-threading within imc STUDIO. - A <i>Windows.Forms</i> or <i>WPF</i>-window can be transferred.
Installation folder	- To start imc STUDIO, the installation folder must be provided as a parameter. - By default, imc STUDIO installs in "C:\Program Files\imc\imc STUDIO 2026\" (or according to the respective version). - The path can be determined automatically by means of the following function: <code>SimplifiedImcStudioComponent.GetImcStudioInstallPath();</code>

Optional parameters

Parameter	Description
initialize	Initializes the imc STUDIO core during the constructor. If <code>false</code> is specified here, this will have to be rectified manually using the <code>Init()</code> -method.
applicationLoaded	Triggers the "imc STUDIO loaded" event.
correctVersionInISPC	Overwrites the imc STUDIO configuration file (default.ispc), so that all plug-ins specified have the same STUDIO core.
wpfApplicationThemeName	Name of the WPF theme. For the default imc STUDIO theme, use an empty string. For the default WPF theme, use "None".
setImcDevExpressWinFormSkin	Aktiviert bzw. deaktiviert den DevExpress-Skin für WinForms-Anwendungen

**Note****Initialize**

If you want to initialize the component later, the `applicationLoaded` parameter must also be set to `false` in the constructor.

```
ImcStudioComponent = new SimplifiedImcStudioComponent(appDir, this, strImcStudioInstallPath, false, false);
ImcStudioComponent.Init();
ImcStudioComponent.ApplicationLoaded(this);
```

5.5.2 Overview

Here, the main functions are described.

All objects themselves come with notes on their respective functionality and call parameters.

Basis class: ImcStudioComponent	
Actions	Triggers the actions registered by the plugins; see <i>imc.Studio.ApplicationInterface.Constants.ActionNames</i>
Commands	for running Sequencer commands
Documents	Access to applications-, project- and experiment settings and supply of supply of events for loading and saving
Environment	Gets global imc STUDIO variables (e.g. installation path)
Events	Registration of EventHandlers, e.g. for the event "device started", "device stopped"; see <i>imc.Studio.ApplicationInterface.Constants.EventNames</i>
InlineFamos	Configuration of an InlineFamos task (requires appropriate license).
Logbook	Access to logbook entries
Notifications	Internal communication between imc STUDIO components
Panel	Access to Panel pages
Scripts	Access to imc STUDIO scripts
Variables	Access to measured data, or all data from a variety of sources. Generating one's own variables. Access to the IVariablesManager by using "Variables.Base"

Class: SimplifiedImcStudioComponent : ImcStudioComponent	
DevSetup	<i>DevSetupExtended</i> Functionality for, among other things, configuration of devices and channels (enhancement of the Basis-class's DevSetup-functionality)
Devices	<i>DevicesActions</i> Devices actions such as device search, connect, disconnect
Experiment	<i>ExperimentActions</i> Actions affecting the experiment such as Load, Save
Measurement	<i>MeasurementActions</i> Actions associated with the measurement such as Start, Stop
Windows	<i>WindowsExtended</i> Calling of dialogs and views present in imc STUDIO

Reference

Usage and functionality of these classes are identical as in imc STUDIO Scripting. For details and example use cases, please see the documentation on Scripting.

Note

The above classes provide access to frequently used functionalities by encapsulating interfaces. Access to further functionalities is partially possible by manual encapsulation. The approach is based on the following naming pattern using *Core* as an example:

`Core.Base` is of type `ICoreBase`. Therefore you need to get an `API_Core_Vx` interface via `GetAPI()`. The letter **x** stands for the interface's version.

Example code:

```
API_Core_V2 coreApi = Core.Base.GetAPI<API_Core_V2>();
```

5.5.3 Actions

The following actions are available in the Basis-project:

Actions in <code>imc.Studio.ApplicationInterface.Constants</code>	
ActionNames	various menu actions
DeviceActions	Connect, Disconnect ProcessConfiguration, Download, Reconfigure ReleaseAllTriggers ResumeDataStorage, SuspendDataStorage Start, Stop

5.5.4 Misc functions

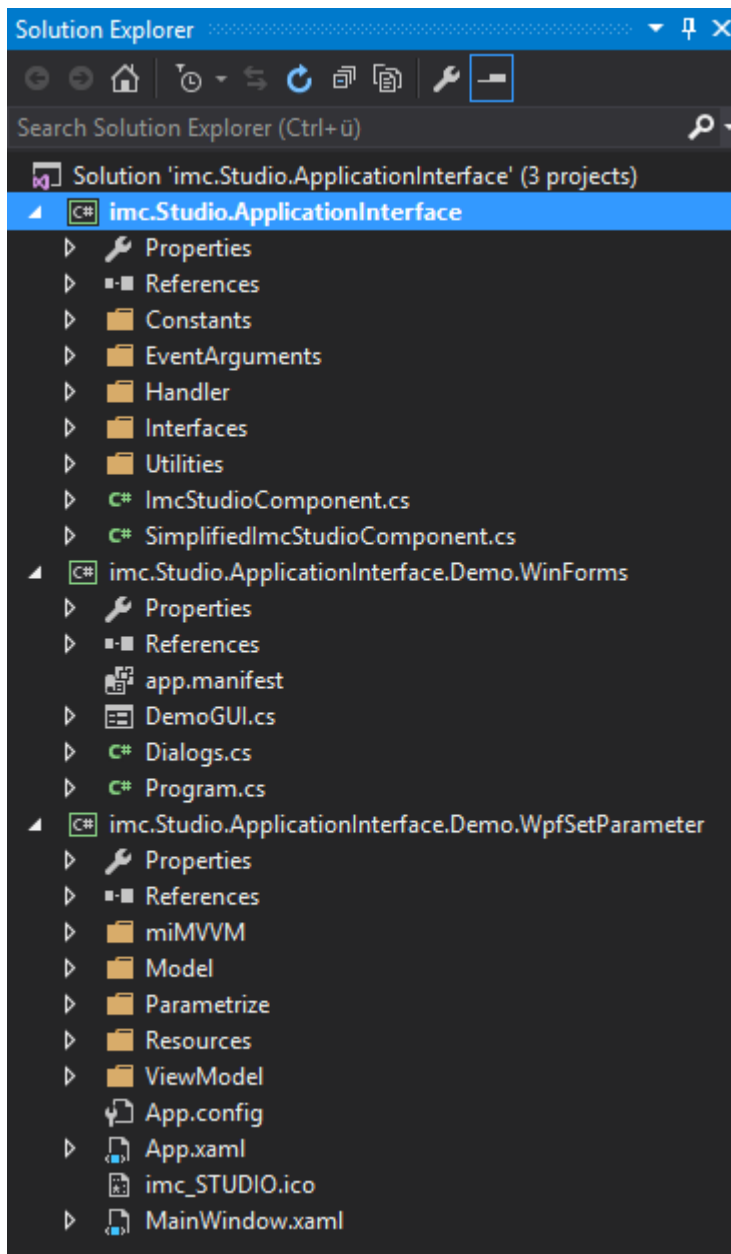
imc.Studio.ApplicationInterface.Constants	
DocumentNames	Collection of constants for accessing documents.
EventNames	Collection of event names.
SharedComponentNames	Collection of shared component names.
WindowNames	Collection of window names.

5.6 Sample projects

The example projects are installed in the following folder:

C:\Users\Public\Documents\imc\imc STUDIO\API 2026\Projects

Projects	Description
WinForms 	Simple GUI (Windows Forms) for using a device and configuring channels as well as displaying the channels in the curve window. Solution: imc.Studio.ApplicationInterface
WpfSetParameter 	Simple GUI (WPF) for using a device. Solution: imc.Studio.ApplicationInterface
WpfCalculate 	Simple GUI (WPF) for using a device and calculation of channels Solution: imc.Studio.ApplicationInterface.Demo.WpfCalculate



Solution Explorer (Visual Studio 2015)

5.6.1 WinForms



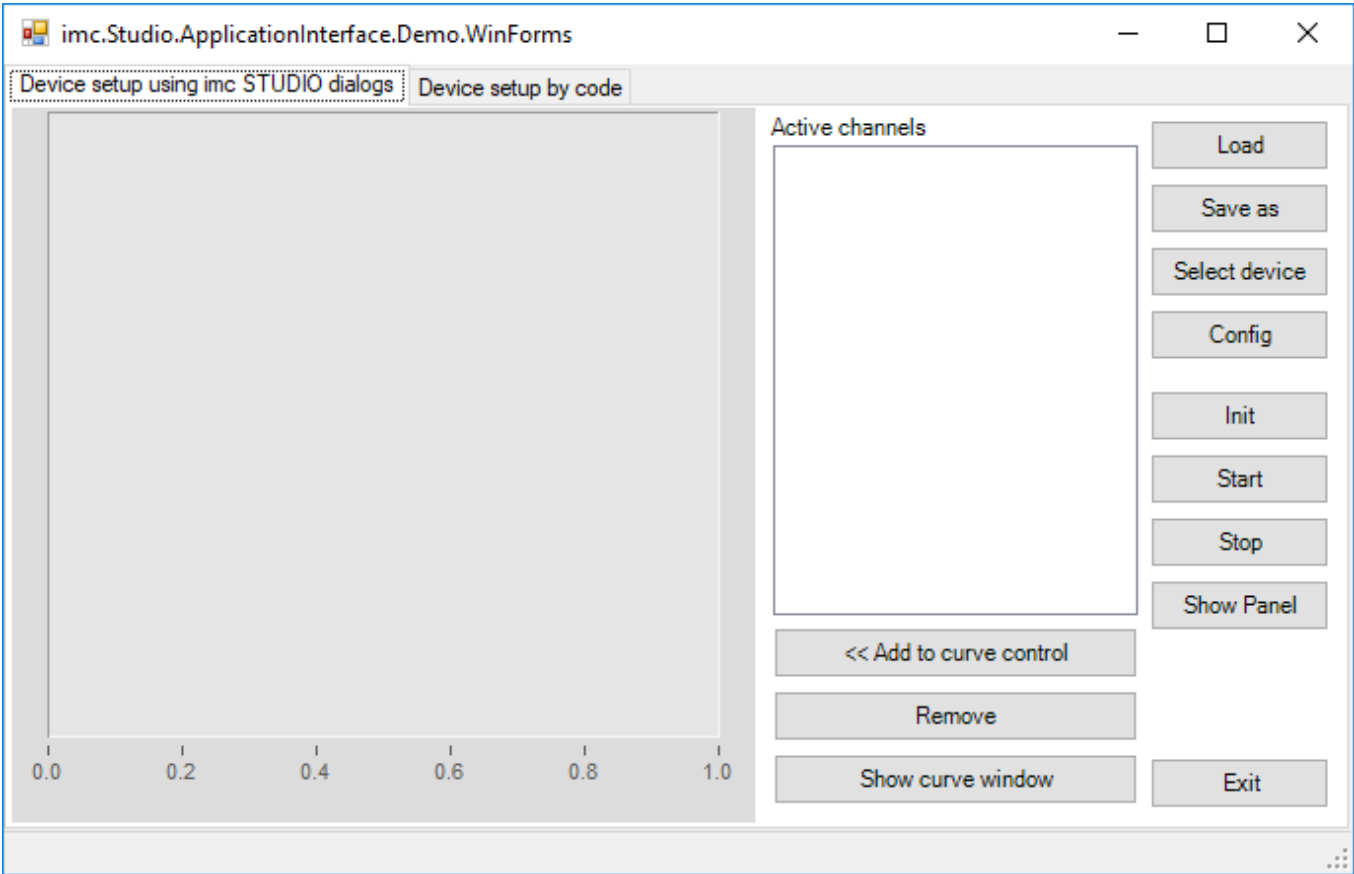
Warning

License required

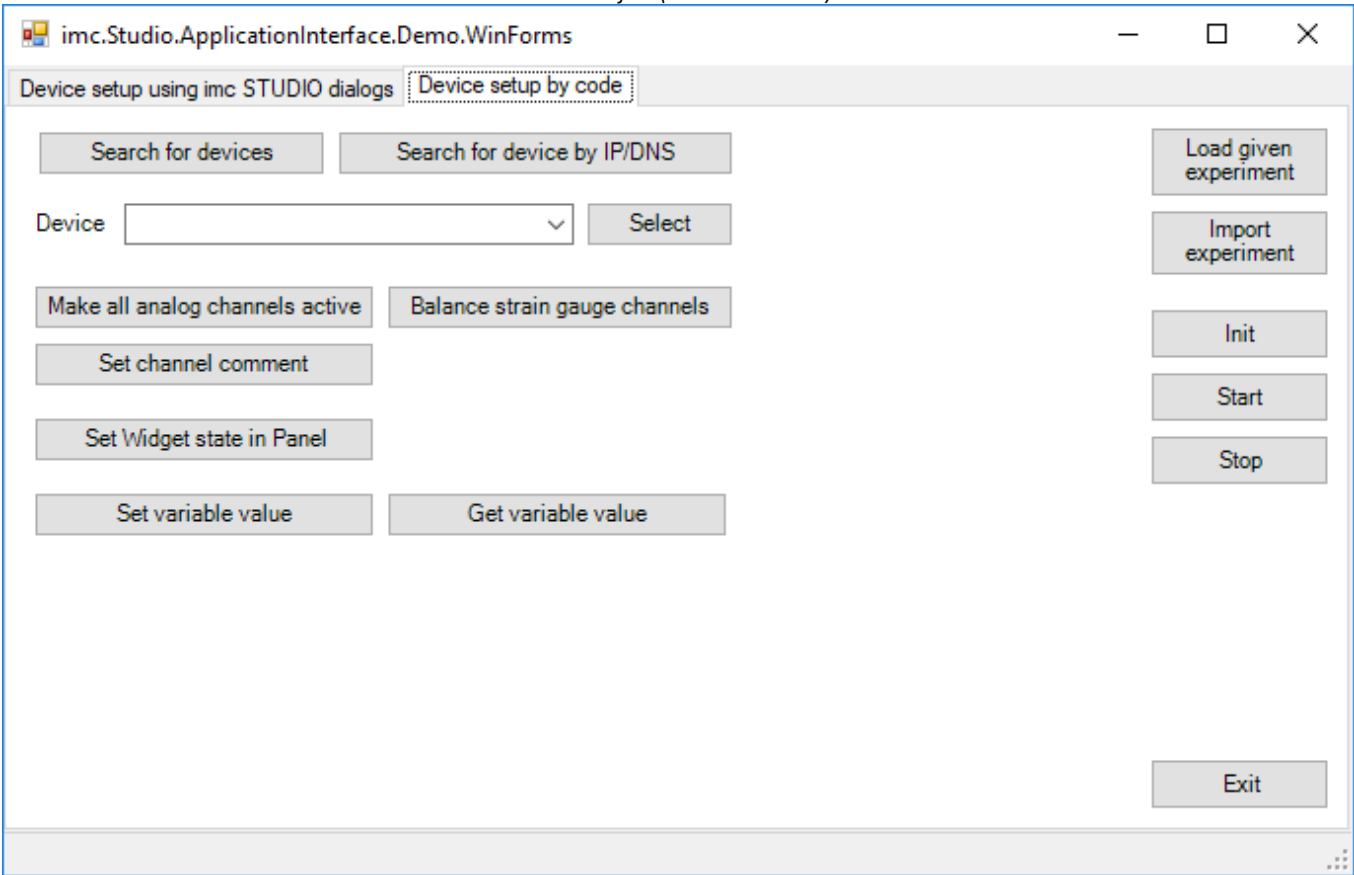
To use the project in its entirety, a **imc API Developer** or **imc API Runtime** license is required.

imc.Studio.ApplicationInterface.Demo.WinForms

- Implementation in C# and WinForms
- Example of small-scope application for controlling a measurement.
- Calling of various dialogs for device searching and selection of devices, as well as the Setup-dialog for setting parameters.
- Start/Stop of measurement and visual output of data.



Demo Project (Windows-Forms)



Demo Project (Windows-Forms)

5.6.2 WpfSetParameter



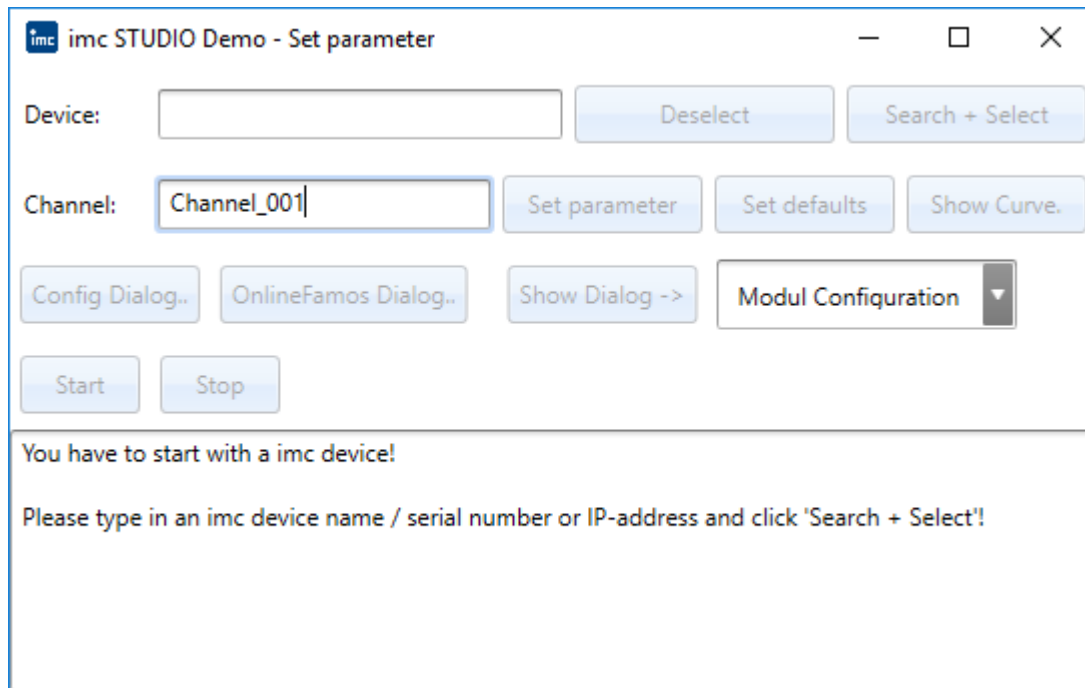
Warning

License required

To use the project in its entirety, a **imc API Developer** or **imc API Runtime** license is required.

imc.Studio.ApplicationInterface.Demo.WpfSetParameter

- Implementation in C# and WPF
- Example of parameterization on command, without manual parameter entry.



Demo Project (WPF)

5.6.3 WpfCalculate



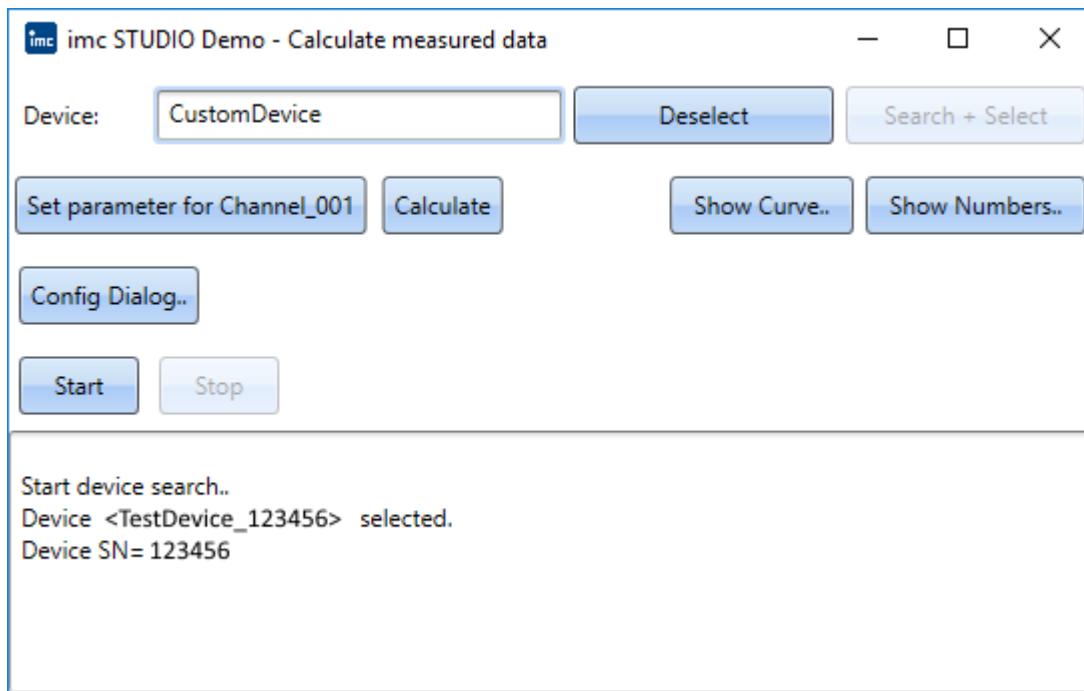
Warning

License required

To use the project in its entirety, a **imc API Developer** or **imc API Runtime** license is required.

imc.Studio.ApplicationInterface.Demo.WpfCalculate

- Implementation in C# and WPF
- Calculation of minimum, maximum and mean value.



Demo-Project (WpfCalculate)

5.7 User-created application

When creating one's own personal project (Windows.Forms or WPF), the following **references** must be incorporated.

In previous versions of imc STUDIO, the references were located in the Global Assembly Cache (GAC). However, since this is no longer supported - or only supported to a limited extent - the references are now located in the imc STUDIO 2026 installation directory.

The most important ones are already pre-populated in the project files (.csproj). You can use the encapsulated registry variable \$(IMCStudioPath) in the .csproj file to add another reference from the installation directory. In this case, the most recently installed version of imc STUDIO (not necessarily the latest) is always used.



Example

Add a reference to the project file

```
<Reference Include="imc.Interfaces">
  <HintPath>$(IMCStudioPath)\imc.Interfaces.dll</HintPath>
  <Private>False</Private>
</Reference>
```

The structure can be viewed in the API's .csproj files. This allows you to manually include a different directory as well.

You need to compile the **Shared-Project** in `C:\Users\Public\Documents\imc\imc STUDIO\API\Shared` for getting the `imc.Studio.ApplicationInterface.dll`.

References	Content
imc.Studio.ApplicationInterface	Entry point for imc STUDIO (<i>in the Shared-Project</i>)
imc.Common.Interfaces	General imc components (Logbook,...)
imc.Interfaces	EClassID parameters for configuring device and channel parameters in DevSetup class.
imc.Studio.Interfaces	General imc STUDIO components
imc.Common.Components.DataObjects	Working with data
imc.Common.Controls.ImcCurves	Display of data

References	Content
PresentationFramework	Required by the SimplifiedImcStudioComponent-class
System.Windows.Forms	For use with WinForms

Namespaces	Content
imc.Studio.Tools.DevSetupGraphGenerator	EClassID (for parameterizing devices)
imc.Common.Components.DataManager imc.Common.Components.DataManager. Interfaces	Data objects such as DmChannel, DmFile, IDmChannel, IDmText.
imc.Studio.ApplicationInterface.Constants	Various actions and constants for parameterizing, e.g. ParameterValues

An example of Windows-Forms or alternatively WPF-project is presented [here](#) ¹⁷⁴.



Reference

For additional information also see the documentation on Scripting.

5.7.1 Example

Initialization example of the imc STUDIO components

In the example code (C#), only the main functions are displayed without any inspection or error handling. There are differences when calling imc STUDIO windows depending on whether the calling program is a WinForms- or a WPF-application.

The main difference is the basis class of the view (System.Windows.Forms or System.Windows.Window).

WPF application

```
using imc.Studio.ApplicationInterface;

public partial class WPFMainWindow : Window
{
    private SimplifiedImcStudioComponent _component;

    public WPFMainWindow ()
    {
        // Application path for this imc STUDIO application
        // imc Studio user interface has its own completely separate path for this!
        string appPath = @"C:\ProgramData\imc\imc STUDIO\Applications\AppDir";

        // Complete path for imc STUDIO installation which is to be used,
        // or zero if imc STUDIO and the main application are located in same folder
        // Example:
        string imcStudioInstallPath = @"C:\Program Files\imc\imc STUDIO 2023";

        // Generating the imc STUDIO components instance
        _component = new SimplifiedImcStudioComponent(appPath, this, imcStudioInstallPath);
    }

    protected override void OnClosing(CancelEventArgs e)
    {
        // Prompt whether to save before closing the application
        _component.ClosingImcComponent(this, e);
        if (!e.Cancel)
        {
            base.OnClosing(e);
        }
    }

    protected override void OnClosed(EventArgs e)
    {
        // Cleanup upon closing application
        base.OnClosed(e);
        _component.Dispose();
        _component = null;
    }
}
```

WinForm-application

```

using imc.Studio.ApplicationInterface;

public partial class WinFormsMainWindow : Form
{
    private SimplifiedImcStudioComponent _component;

    internal WinFormsMainWindow()
    {
        InitializeComponent();

        // Application path for this imc STUDIO application
        // imc Studio user interface has its own completely separate path for this!
        string appPath = @"C:\ProgramData\imc\imc STUDIO\Applications\AppDir";

        // Complete path for imc STUDIO installation which is to be used,
        // or zero if imc STUDIO and the main application are located in same folder
        // Example:
        string imcStudioInstallPath = @"C:\Program Files\imc\imc STUDIO 2023";

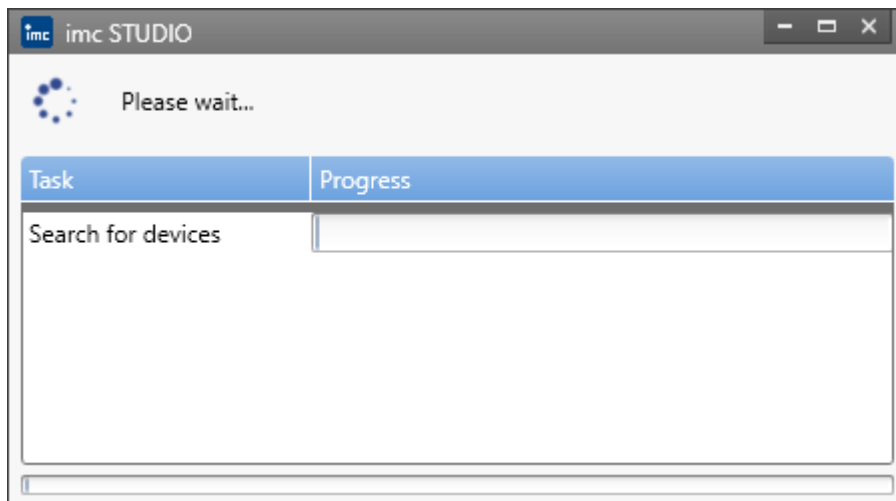
        // Generating the imc STUDIO components instance
        _component = new SimplifiedImcStudioComponent(appPath, this, imcStudioInstallPath);
    }

    protected override void OnClosing(CancelEventArgs e)
    {
        // Prompt whether to save before closing the application
        _component.ClosingImcComponent(this, e);
        if (!e.Cancel)
        {
            base.OnClosing(e);
        }
    }

    protected override void OnClosed(EventArgs e)
    {
        // Cleanup upon closing the application
        base.OnClosed(e);
        _component.Dispose();
        _component = null;
    }
}

```

5.7.2 ProgressVisualizer



Progress-Dialog (ProgressVisualizer)

When you execute actions that exceed a certain duration, a progress dialog called ProgressVisualizer opens. In your own applications, it is therefore useful to hide this and transfer the information to your own status element.

```

using imc.Common.Interfaces.Logbook;
using imc.Studio.ApplicationInterface;
using imc.Studio.Interfaces.Core.Services.ProgressVisualizer;
private SimplifiedImcStudioComponent _component;
private void HideProgressVisualizer()
{
    var services = _component.Core.Services.GetAPI<API_CoreServices_V2>();
    var progressVisualizer = services.ProgressVisualizer.GetAPI<API_CoreServicesProgressVisualizer_V3>();
    progressVisualizer.GUIEnableHideTemporary();

    progressVisualizer.TaskProgress += OnProgressVisualizerTaskProgress;
}
private void OnProgressVisualizerTaskProgress(object sender, ICoreServicesTaskEventArgsBase e)
{
    var taskEventArgs = e.GetAPI<API_CoreServicesTaskEventArgs_V1>();

    ProgressTask(taskEventArgs);
}
private void ProgressTask(API_CoreServicesTaskEventArgs_V1 taskEventArgs)
{
    _component.Logbook.LogEntry(ErrorSender, taskEventArgs.Text, 0, ELogbookEntryCategory.Information);
}

```

5.7.3 Experiments

Activated project management

Experiment Name

```
string experimentName = _component.ProjectManagement.CurrentExperiment.Name;
```

Load experiment

When loading experiments, the path relative to the database must be specified. This includes the project name.

```
var experimentPathInDatabase = @"StandardProject\Experiment_0001";
var result = studioComponents.Experiment.Load(experimentPathInDatabase);
```

Import experiment

When importing an experiment, the full path of the experiment file as well as the project name must be specified.

```
var projectName = "StandardProject";
var experimentFileName = @"c:\temp\Test.imcStudio";
var result = _components.Experiment.Import(projectName, experimentFileName);
```

Save experiment

```
_components.Experiment.Save()
```

Save experiment as

Variant 1:

```
var experimentFileName = @"c:\Users\Public\Documents\DB\StandardProject\Experiment_0001\Experiment_0001.imcStudio";
_component.Experiment.Save(experimentFileName);
```

Variant 2:

To use the ProjectManagement class, the following references must be added to the C# project.

This can also be done in Visual Studio via the tool-tip and "Show potential fixes".

- imc.Common.Interfaces (für IExperiment_V1)
- imc.Studio.Interfaces (für IProjectManagementBase, API_ProjectManagement_V7)

```
var currentExperiment = _component.ProjectManagement.CurrentExperiment;
currentExperiment.Name = "NewExperimentName";
component.ProjectManagement.ExperimentSaveAs(null, currentExperiment, false);
```

Save as dialog

```
_component.Experiment.SaveAs()
```

Without project management

Load experiment

```
var experimentFileName = @"c:\temp\Test.imcExp";  
var result = _component.Experiment.Load(experimentFileName);
```

Save experiment

```
_components.Experiment.Save()
```

Save experiment as

```
var experimentFileName = @"c:\temp\Test.imcExp";  
_component.Experiment.Save(experimentFileName);
```

5.7.4 Curve window



Note

License activation

For using the curve window control in your development environment you need to activate a **imc API Developer** license.

If you wish to use the curve window on one of your user interfaces (Windows.Forms, WPF, ...), perform the following necessary steps.

Adding the curve window to the toolbox

The instructions below demonstrate how to add the curve window to the toolbox on the basis of Visual Studio as an example:

1. Open the toolbox tool window (while your GUI designer is open)
2. From the context menu (opened by right-clicking), select "Choose Items..."
3. Select the tab ".NET Framework Components"
4. Click on the Browse button
5. Select the DLL from the following path:
C:\Users\Public\Documents\imc\imc DATA API\<%API_VERSION%>\<%API_REVISION%>\Assemblies\Release\x64\ imc.Common.Controls.ImcCurves
6. Two new entries will appear in the list: CimcCurveControl and CimcCurveGlobalSettings.
7. Leave the check mark for both of these and close the dialog with OK.
8. Subsequently, the curve window object will be available in the toolbox.

Usage in Windows.Forms projects

In Windows.Forms-projects, it is possible to use Drag&Drop to move the curve window-object in the Designer directly into the GUI.

Usage in WPF projects

1. In WPF-projects, the following references must be added to the project:
 - System.Windows.Forms

- WindowsFormsIntegration
- imc.Common.Controls.ImcCurves

2. In the Designer, the following namespace is used in the xaml-code:

```
xmlns:imc="clr-namespace:imc.Common.Controls.ImcCurves;assembly=imc.Common.Controls.ImcCurves"
```

3. Within the grid object, the WindowsFormsHost object containing the curve window is added:

```
<WindowsFormsHost>
  <imc:CimcCurveControl x:Name="mainCurveWindow" />
</WindowsFormsHost>
```

4. The curve window-object is not displayed visually, but it can be addressed in the code by its associated name.

Show data in a curve window

To display channels in a curve window (mainCurveWindow), the data object of the channel is needed.

The following shows how all Analog Channels are displayed in the curve window:

```
using imc.Studio.ApplicationInterface;
using imc.Common.Components.DataManager.Interfaces;
using imc.Common.Controls.ImcCurves;
using imc.Common.Controls.ImcCurves.Interfaces;
using imc.Common.Controls.ImcCurves;
using imc.Data.Core.Contract;

private SimplifiedImcStudioComponent _component;

public void ShowVariableInCurve(IVariable variable)
{
    var result = variable.QueryInterface<IDmChannel>();

    if (result.IsSuccessful)
    {
        var channel = result.Value;

        mainCurveWindow.Clear();
        mainCurveWindow.AppendChannel(channel, CwAppendConstants.AppendNewLine, 0);
    }
}
```

5.7.5 Commands

Import an OFA file

To import an imc Online FAMOS (OFA) file, the command "Import supplemental file" is used.

```
using imc.Studio.ApplicationInterface;

private SimplifiedImcStudioComponent _component;

/// <summary>
/// Imports an *.ofa file to a given device.
/// </summary>
/// <param name="fileName">OFA filename.</param>
/// <param name="deviceName">Device identifier (eDeviceName); e.g. imcDev__18123456.</param>
private void ImportOfaFile(string fileName, string deviceName)
{
    _component.Commands.Invoke("Die Zusatzdatei 'C:\\\\Training\\\\API_2018-11\\\\Misc\\\\TestCalc.ofa'
wird importiert",
        "<AppPlotFactoryCmd version=\\\"1\\\">" +
        "<common version=\\\"1\\\" cuid=\\\"ImportImcDevicesSatelliteFile\\\" caption=\\\"Zusatzdatei
importieren\\\">" +
        "<command version=\\\"1\\\"><properties version=\\\"1\\\">" +
        "<cmd owner=\\\"imc.Studio.PlugIns.AppPlot.Engine.AppPlotImportSatelliteFile\\\" version=\\\"2\\\"
cuid=\\\"ImportImcDevicesSatelliteFile\\\">" +
        "<filename>" + fileName + "</filename>" +
        "<overwrite>eOverwriteWithoutPrompt</overwrite><targetname />" +
        "<map><devsatimportmap>" +
        "<devicename>" + deviceName + "</devicename> <used>True</used></devsatimportmap>" +
        "</map></cmd></properties></command></common></AppPlotFactoryCmd>");
}
```

5.7.6 Windows (WinForms)

General

The desired window/control *windowControl* is added to the WinForm *form*:

```
using (var form = new Form())
{
    form.Width = windowUi.Width;
    form.Height = windowUi.Height;
    form.Text = "WindowName";
    form.Controls.Add(windowControl); // whereas windowControl is the desired control
    form.Dock = DockStyle.Fill;
    form.ShowDialog();
}
```

Logbook

```
private void ShowLogbook()
{
    var logbookUi = _component.Windows.GetWindowControl("Logbook").AsWinFormsControl();
    logbookUi.Width = 1200;
    logbookUi.Height = 500;
    logbookUi.Dock = DockStyle.Fill;

    using (var logbookForm = new Form())
    {
        logbookForm.Width = logbookUi.Width;
        logbookForm.Height = logbookUi.Height;
        logbookForm.Text = "Logbook";
        logbookForm.Controls.Add(logbookUi);
        logbookForm.Dock = DockStyle.Fill;
        logbookForm.ShowDialog();
    }
}
```

Setup pages: Devices

```
private void ShowDevicesPage()
{
    var devicesPageName = "Devices";
    var setupControl = _component.Windows.CreateSetupControl(devicesPageName);

    using (var devicesForm = new Form())
    {
        devicesForm.Width = 1000;
        devicesForm.Height = 800;
        devicesForm.Text = devicesPageName;
        devicesForm.Controls.Add(setupControl);
        setupControl.Dock = DockStyle.Fill;
        devicesForm.ShowDialog();
    }
}
```

Panel

```
private void ShowPanel()
{
    var panelControl = _component.Windows.Panel.AsWinFormsControl();

    if (panelControl == null)
    {
        MessageBox.Show(@"Panel was not found. Make sure that the Panel plug-in is properly loaded!");
        return;
    }

    using (var panelForm = new Form())
    {
        devicesForm.Width = 1000;
        devicesForm.Height = 800;
        devicesForm.Text = @"Panel";
        devicesForm.Controls.Add(panelControl);
        setupControl.Dock = DockStyle.Fill;
    }
}
```

5.7.7 Windows (WPF)

Logbook

```
private void AddLogbookToWindow()
{
    var logbookElement = CreateStudioLogbook(_studioComponent);
    Logbooks.Children.Add(logbookElement); // Logbooks is a StackPanel element
}

private static FrameworkElement CreateStudioLogbook(SimplifiedImcStudioComponent studioComponent)
{
    var logbookAsWpf = _cComponent.Windows.GetWindowControl("Logbook").AsWPFFControl();

    var logbookElement = (logbookAsWpf as FrameworkElement);
    logbookElement.LayoutTransform = new ScaleTransform(0.8, 0.8);
    logbookElement.MaxWidth = 1024;
    logbookElement.MaxHeight = 800;
    return logbookElement;
}
```

Setup Seiten: Devices

```
private void ShowSetupPage()
{
    var devicesPageName = "Devices";
    var deviceUi = _component.Windows.CreateSetupControl(devicesPageName).AsWPFFControl();
    deviceUi.Width = 1400;
    deviceUi.Height = 700;

    var window = new Window()
    {
        Width = deviceUi.Width,
        Height = deviceUi.Height,
        HorizontalContentAlignment = System.Windows.HorizontalAlignment.Stretch,
        VerticalContentAlignment = VerticalAlignment.Stretch,
        Content = deviceUi
    };

    window.ShowDialog();
}
```

Panel

```
private void ShowPanelAsWpf()
{
    var panelControl = _component.Windows.Panel.AsWPFFControl();

    if (panelControl == null)
    {
        MessageBox.Show(@"Panel was not found. Make sure that the Panel plug-in is properly loaded!");
        return;
    }

    Window w = new Window
    {
        Width = 1000,
        Height = 800,
        Content = panelControl
    };

    w.ShowDialog();
}
```

5.8 Distribution

In order to transfer the project you have developed, the following prerequisites must be met by the target computer:

- imc STUDIO is installed
- depending on the functional scope, activation of the appropriate licenses
- Activation of imc API Runtime license
- Deployment of the desired project
- Application folder (including default.ispc)

5.9 Notes

Tips

It is recommended to use imc STUDIO to make the project settings (such as views, Setup-pages, ...) by using the API-database path.

Curve window

For using the imc curve window, an activated license for **imc API Developer** is required.

FileNotFoundException: imc.Common.Components.DataObjects.dll

If you start the examples of imc API directly after the installation of imc STUDIO, a FileNotFoundException may occur.

In this case please restart the PC and try again.

6 ME'Scope

6.1 Overview

The ME'Scope software can be used to display the movement of structures in a model. Data in the time domain or frequency domain can be used as measured variables.

6.2 Supported imc hardware

imc modules, that support the IEPE coupling:

- ICPU-6 (imc ARGUS*fit*),
- ICPU2-8 (imc CRONOS*flex*, imc CRONOS*compact*),
- AUDIO2-4 (imc CRONOS*flex*, imc CRONOS*compact*),
- QI-4 (imc CRONOS*flex*)

6.3 Licensing

In addition to the license for ME'Scope, a license for the "imc API Runtime" is required.

6.4 Installation

Prerequisites for using ME'Scope

- Installation of ME'Scope version 23.0.12.13
Please contact us for other versions.
- Installation of imc STUDIO 2024 R2 or higher
- Installation of the imc DATA API Runtime
installed under Products\Development\Setup\imc_DATA_API\

Reference

Please find further information in the imc DATA API manual.

- Import of the MEScopeFiles.zip
 - Execute the "CopyApplication.cmd" file
Alternatively: copy manually the "Application" directory to "c:\ProgramData\imc\imc STUDIO\"
 - Copy the directories located in Acquisition-Folder to ME'scope installation directory
e.g. "c:\Program Files\MEScope\Acquisition"
- Further details on configuration can be found in the readme file in the *.zip file.

6.5 Configuration

- If there is only one device in the network, this device will be used.
- Otherwise, please enter the serial number of the device or main module in the following file:
"C:\ProgramData\imc\imc STUDIO\Applications\me_scope_config.cfg"



Warning

It is recommended that only one imc STUDIO version be installed during software development in order to avoid version conflicts with the assemblies.

When operating the finished application, multiple versions can also be installed in parallel.

6.6 Notes

Log files

Log files are located in the following paths:

C:\ProgramData\imc\imc STUDIO\Applications\MEscope\log

C:\ProgramData\imc\imc STUDIO\Applications\MEscope\logFrontEnd

Index

3

- 3PDI
 - AudioDevice 140
 - ChannelLoader 141
 - fos4x 142
 - FunctionSimulator 144
- 3rd Party Device Management 137

A

- adjustment 7
- API
 - Actions 168
 - Aktionen 168
 - Basis Interface 165
 - Beispiele 174
 - Classes 167
 - Curve window 163, 182
 - default.ispc 166
 - Demo 182
 - DLL 173
 - Examples 169, 170, 172
 - Functionality 167
 - getting started 163
 - Hints 182
 - imc Managed Class Libraries 163
 - imc DATA API 163
 - imc FAMOS 163
 - Initialisierung 166
 - Logbook 180, 181
 - prerequisites 163
 - PresentationFramework 173
 - Problems 182
 - References 173
 - Version 164
 - Version festlegen 164
 - Windows.Forms 170
 - WPF 172

- AudioDevice 140

B

- Balance action
 - execute via Scripting 99
- Breakpoint
 - Scripting 20

C

- calibration 7
- CE Certification 7
- Certificates 7

- Change requests 7
- Channel
 - time stamped 134
- Channel action
 - execute via Scripting 99
- Channel parameters
 - get/set via Scripting 31, 32, 99
- Channel playback 141
- Channel properties
 - get/set via Scripting 99
- ChannelLoader 141
- Channels
 - add to SimplePollDevice 158
 - add to Third Party Device 132
 - Parameters (Scripting) 31, 32
 - Properties (Scripting) 31, 32
 - Saving (Scripting) 31, 32
- Command name 75
- Commands
 - execute via Scripting 75
- Compiler
 - 1607 124
 - 3270 124
 - CS0029 124
 - CS0234 124
 - CS0525 124
 - CS1607 124
 - CS1729 124
 - Error CS0029 124
 - Error CS0234 124
 - Error CS0525 124
 - Error CS1729 124
 - MSB3270 124
 - Warning CS1607 124
 - Warning MSB3270 124
- COM-Port 122, 128
- Connection status
 - get/set via Scripting 97
- Context Script
 - Activity Scope 89
 - create 12, 13
 - Parameter changed (Event) 89
 - Status changed (Event) 89
 - Value changed (Event) 89
- Curve window 178
 - Add channel 46
 - Axes 46
 - Scripting 46
 - y-Axis 46
- Customer support
 - Tech support 6

D

- Data transfer 137
- Debug
 - Breakpoint 20
 - Scripting 20
- Design-Mode
 - activate via Scripting 75
 - deactivate via Scripting 75
- Device actions
 - execute via Scripting 75, 97
- Device description 129
- Device parameters
 - get/set via Scripting 97
- Device properties
 - get/set via Scripting 97
- Devices
 - add to Third Party Devices 131
 - Parameter (Scripting) 22, 23
 - Parameters (Scripting) 97
 - Properties (Scripting) 22, 23, 97
- Dialog response 79
- DIN-EN-ISO-9001 7
- Diskstart
 - execute via Scripting 22, 27

E

- eRange 136
- Event Script
 - create 12, 13
 - Measurement storage area 95
- Experiment parameters
 - get/set via Scripting 106
- Export
 - Scripting 17

F

- FAQ
 - Compiler error 124
 - Scripting 124
- FFT 133
- File access
 - Scripting 83
- FileNotFoundException 157
- fos4x 142
 - Balance 143
 - Balancing 143
 - Boundary 143
 - Characteristic curve 143
 - Compensation 143
 - Connetor 143
 - License 142

- lower Wavelength 143
- nSens 143
- Reference temperature 143
- Reference wavelength 143
- Sensitivity 143
- Sensor 143
- Signal quality 143
- Tare set point 143
- Time 143
- Unit 143
- upper Wavelength 143
- Wavelength 143

FunctionSimulator 144, 147

G

- General terms and conditions 7
- Guarantee 7

H

- Hotline
 - Tech support 6

I

- imc FAMOS
 - Access to functions in Scripting 70
- Import
 - Scripting 17
- Inline FAMOS
 - private Task 74
 - Scripting 74
- ISO-9001 7

L

- Liability restrictions 7
- License 178
- Limited Warranty 7
- Logbook

- Access via Scripting 65, 67
 - Entries 65
 - Export 65
 - OnError 65
 - OnWarning 65

M

- maintenance 7
- Measurement
 - start/stop via Scripting 22, 23, 75
- Measurement status
 - get via Scripting 97
- Menu actions
 - execute via Scripting 75

O

OFA
 Scripting 28
Online FAMOS
 Scripting 28
Options
 Dialog response 79

P

Panel
 Access via Scripting 39, 40, 42, 44
 Add Windows.Forms 39, 42
 Tables 39, 44
Panel Script 85
 create 12
 Data Browser 85
 New Widgets 85
 Variables 85
 Widgets 85
Parameters
 Channels (Scripting) 99
 Devices (Scripting) 97
 Experiment (Scripting) 106
Product improvement 7
Progress dialog 176
ProgressVisualizer 176
PushDeviceDriver 146

Q

Quality Management 7

R

Range 136
repair 7
RS232 122, 128

S

Script 12, 13, 84
 create 12, 13
 execute 84
 run 84
Script Editor 12, 13
 Language 12, 13
Script options 113
Script Scope 12, 13
Scripting 9
 1607 124
 3270 124
 abstract base class 19
 AbstractScript.cs 124
 Activity Scope 14, 89

Advanced code completion 113
Axes 46
Balance 31, 32
base class 19
Breakpoint 20
Changes 81
Channel calculation 119
Channel number 31, 32
Channel parameters 31, 32, 115
Channel properties 31, 32
Channel state 31, 32
Classes 21
Clipboard 80
Clock state 22, 29
Clock type 22, 29
Commands via Scripting 75
COM-Port 128
Connector 31, 32
Context menu 10
Context Script 13, 89
Coupling 31, 32
CS0029 124
CS0234 124
CS0525 124
CS1607 124
CS1729 124
Curve window 46
Datapool 21
Debug 20
Device actions 22, 23
Device parameters 22, 23
Device properties 22, 23
Device selection 22, 23
DevSetup 21
Dialog response 79
Diskstart 22, 27, 97
Duration 31, 32
Edit 15
Editor 20
Enumerated channel number 31, 32
Error 65, 124
Error CS0029 124
Error CS0234 124
Error CS0525 124
Error CS1729 124
Event Script 13, 95
Events 68
Execute 16
Export 17
FAQ 124
File access 83
Filtering 31, 32

Scripting 9
 get/set Device properties 22, 23
 imc FAMOS functions 70
 imc FAMOS Kit 19
 imc FAMOS Math 19
 Import 17
 Import 4.0R1 124
 Inline FAMOS 74
 Input range 31, 32
 IntelliSense 113
 Introduction 12, 13
 Logbook 65, 67
 Measurement storage area 95
 Menu 10
 Menu actions via Scripting 75
 Meta data 31, 32
 Metadata 115
 Module type 31, 32
 MSB3270 124
 NTP 29
 OFA 28
 Offset 31, 32
 OnError 65
 Online FAMOS 28
 OnWarning 65
 Open 15
 Options 14, 113
 Panel 21, 39, 40, 42, 44
 Panel Script 12, 13
 Parallel-Installation 124
 Parameter changed (Event) 89
 Parameters 108
 ParameterValues 22, 23, 31, 32
 Preprocessing 31, 32
 private Task (IFA) 74
 Provider 19
 Proxy class 19, 124
 Range 31, 32
 Regenerate proxies 19
 Restore 81, 113
 RS232 128
 Run 16
 Sample time 31, 32
 Scale 31, 32
 Scaling 31, 32
 Script 12, 13, 84
 Script clipboard 123
 Script Editor 15, 20
 Script Scope 12, 13
 Script storage location 113
 Sensitivity 31, 32
 Separate proxy class library 14, 80

SerialPort 128
Setting values 108
start/stop Measurement 22, 23
Status changed (Event) 89
Storage 26
Storage scope 14
Strain gauge 31, 32
Subversion 113
Supplemental files 28
SVN 113
Synchronization 29
Tables 39, 44
TED 31, 32
Third Party Device Script 126
Timed start 29
Tool window 10
Tutorial 115, 119
Type Library Script 13, 93
Unit 31, 32
Value changed (Event) 89
Warning CS1607 124
Warning MSB3270 124
Widget 39, 42, 44
Working copy 113
y-Axis 46
Sequencer
 Commands via Scripting 75
 create scripts 12, 13
 start/stop via Scripting 75
Serial Interface 122
SerialPort 128
Service
 Tech support 6
service and maintenance 7
service check 7
SimplePollDevice 152
 add channel 158
 add Display variable 160
 Extend 158, 160
SimplePushDevice 155
Storage 22, 26
Supplemental files
 Scripting 28
Synchronisation
 get/set via Scripting 97
System.IO.FileNotFoundException 157

Tech support 6
Telephone numbers
 Tech support 6
Third Party Device 146

- Third Party Device 146
 - add Channel 132
 - Data transfer 137
 - FunctionSimulator 147
 - Logbook 137
 - SimplePollDevice 152
 - SimplePushDevice 155
 - Simulator 146
 - virtual 146
- Third Party Device Script 96
 - create Script 126
- Third Party Devices
 - add Device 131
 - eRange 136
 - Range 136
- Third Party Device Interface 126
- time stamped 134
- Type Library Script 93
 - create 12, 13

V

- Variables
 - Access via Scripting 85
- virtual device 146

W

- Warranty 7
- Widget
 - Access via Scripting 39, 42, 44, 85
- WinForms 178
- WPF 178



An Axiometrix Solutions Brand

Contact imc

Address

imc Test & Measurement GmbH
Voltastr. 5
13355 Berlin

Phone: (Germany): +49 30 467090-0

E-Mail: info@imc-tm.de

Internet: <https://www.imc-tm.com>

Tech support

If you have problems or questions, please contact our tech support:

Phone: (Germany): +49 30 467090-26

E-Mail: hotline@imc-tm.de

Internet: <https://www.imc-tm.com/service-training/>

Service and maintenance

Our service team is at your disposal for service and maintenance inquiries:

Phone: (Germany): +49 30 629396-333

E-Mail: imc-service@axiomatrixsolutions.com

Internet: <https://www.imc-tm.com/service>

imc ACADEMY - Training center

The safe handling of measurement devices requires a good knowledge of the system. At our training center, experienced specialists are here to share their knowledge.

E-Mail: schulung@imc-tm.de

Internet: <https://www.imc-tm.com/service-training/imc-academy>

International partners

You will find the contact person responsible for you in our overview list of imc partners:

Internet: <https://www.imc-tm.com/imc-worldwide/>

imc @ Social Media

<https://www.linkedin.com/company/imc-test-&-measurement-gmbh>

<https://www.linkedin.com/company/famos-test-measurement-data-analysis>

<https://www.youtube.com/c/imcTestMeasurementGmbH>