

OpenXiLEnv User Guide

for V0.8.15

Authors: Eric Bieber, Udo Gillich, Horst Kiebler, Michael Matthäi

Attention: Full compability to lower OpenXiLEnv-versions is not given in all parts.

1. Table of content

- 1. [Table of content](#)
- 2. [What is OpenXiLEnv](#)
 - 2.1. [Use case scenarios](#)
 - 2.2. [Platforms](#)
 - 2.3. [Prime advantages](#)
- 3. [Installation](#)
 - 3.1. [General information for installation](#)
 - 3.2. [Installing OpenXiLEnv](#)
 - 3.3. [Setup a new project in OpenXiLEnv](#)
 - 3.4. [Installing OpenXiLEnv HIL option](#)
 - 3.5. [Default setting](#)
 - 3.6. [Hotkeys](#)
 - 3.7. [System requirements](#)
- 4. [Concept](#)
 - 4.1. [Concept](#)
- 5. [Control Panel](#)
 - 5.1. [General information of the control panel](#)
 - 5.2. [Run-Control](#)
 - 5.2.1. [Breakpoint](#)
 - 5.3. [Process](#)
 - 5.4. [Blackboard](#)
 - 5.4.1. [Conversion formula](#)
 - 5.4.2. [Text replacement](#)
 - 5.4.3. [Supported data types](#)
 - 5.4.4. [Fix registered variables](#)
 - 5.4.5. [System variables / environment variables](#)
 - 5.5. [Internal Process Control](#)
- 6. [Display of variables](#)
 - 6.1. [General information for the display of variables](#)
 - 6.2. [Text window](#)
 - 6.3. [Oscilloscope-window](#)
 - 6.3.1. [Default settings](#)
 - 6.3.2. [Choice of variables](#)
 - 6.3.3. [Deactivating chosen variables](#)
 - 6.3.4. [Zoom](#)
 - 6.3.5. [Scroll](#)
 - 6.3.6. [Print](#)

- 6.4. [Tachometer](#)
 - 6.5. [Knob](#)
 - 6.6. [Slider \(Slide control\)](#)
 - 6.7. [Bargraph](#)
 - 6.8. [Control lamps](#)
 - 6.8.1. [Control lamps configuration-dialogue](#)
 - 6.9. [Enums](#)
- 7. [Calibration tree](#)
 - 7.1. [General](#)
 - 7.2. [Data display](#)
- 8. [Processes](#)
 - 8.1. [General information about processes](#)
 - 8.2. [Internal processes](#)
 - 8.2.1. [Signal generator](#)
 - 8.2.2. [Formula calculator](#)
 - 8.2.3. [Stimuli-Player](#)
 - 8.2.4. [Trace-Recorder](#)
 - 8.2.5. [Script](#)
 - 8.2.6. [TimeProcess](#)
 - 8.3. [External processes](#)
 - 8.3.1. [Visual-C++ .Net](#)
 - 8.3.2. [Interface](#)
 - 8.3.3. [Value range control for variable](#)
- 9. [Script language](#)
 - 9.1. [Introduction](#)
 - 9.2. [Instruction set](#)
 - 9.2.1. [Basic instructions\[\]](#)
 - 9.2.2. [Only HiL instructions](#)
 - 9.2.3. [Environment variable within script-instructions](#)
 - 9.3. [Notes on the instructions](#)
 - 9.4. [Script debug window](#)
 - 9.5. [Other](#)
 - 9.5.1. [The 'Script_Status_Flag'](#)
 - 9.5.2. [Error messages](#)
 - 9.5.3. [Notes](#)
 - 9.5.4. [Tips & Tricks](#)
 - 9.5.5. [Examples](#)
- 10. [Remote Control](#)
 - 10.1. [Connection buildup and others](#)
 - 10.1.1. [XiLEnv_ConnectTo](#)
 - 10.1.2. [XiLEnv_ConnectToInstance](#)
 - 10.1.3. [XiLEnv_DisconnectFrom](#)
 - 10.1.4. [XiLEnv_DisconnectAndClose](#)
 - 10.1.5. [XiLEnv_IsConnectedTo](#)
 - 10.1.6. [XiLEnv_GetVersion](#)
 - 10.1.7. [XiLEnv_GetAPIVersion](#)

- 10.1.8. [XiEnv_GetAPIModulePath](#)
- 10.1.9. [XiEnv_CreateFileWithContent](#)
- 10.1.10. [XiEnv_SetEnvironVar](#)
- 10.1.11. [XiEnv_GetEnvironVar](#)
- 10.1.12. [XiEnv_ChangeSettings](#)
- 10.1.13. [XiEnv_TextOut](#)
- 10.2. [XiEnv_ErrorTextOut](#)
- 10.3. [Scheduler](#)
 - 10.3.1. [XiEnv_StopScheduler](#)
 - 10.3.2. [XiEnv_ContinueScheduler](#)
 - 10.3.3. [XiEnv_IsSchedulerRunning](#)
 - 10.3.4. [XiEnv_DoNextCycles](#)
 - 10.3.5. [XiEnv_DoNextCyclesAndWait](#)
 - 10.3.6. [XiEnv_StartProcess](#)
 - 10.3.7. [XiEnv_StartProcessAndLoadSvl](#)
- 10.4. [XiEnv_StartProcessEx](#)
 - 10.4.1. [XiEnv_StartProcessEx2](#)
 - 10.4.2. [XiEnv_StopProcess](#)
 - 10.4.3. [XiEnv_GetNextProcess](#)
 - 10.4.4. [XiEnv_GetProcessState](#)
 - 10.4.5. [XiEnv_AddBeforeProcessEquationFromFile](#)
- 10.5. [XiEnv_AddBehindProcessEquationFromFile](#)
 - 10.5.1. [XiEnv_DelBeforeProcessEquations](#)
 - 10.5.2. [XiEnv_DelBehindProcessEquations](#)
 - 10.5.3. [XiEnv_WaitUntil](#)
- 10.6. [Control internal processes](#)
 - 10.6.1. [XiEnv_StartScript](#)
 - 10.6.2. [XiEnv_StopScript](#)
 - 10.6.3. [XiEnv_StartRecorder](#)
 - 10.6.4. [XiEnv_StopRecorder](#)
 - 10.6.5. [XiEnv_StartPlayer](#)
 - 10.6.6. [XiEnv_StopPlayer](#)
 - 10.6.7. [XiEnv_StartEquations](#)
 - 10.6.8. [XiEnv_StopEquations](#)
 - 10.6.9. [XiEnv_StartGenerator](#)
 - 10.6.10. [XiEnv_StopGenerator](#)
- 10.7. [Control the user interface](#)
 - 10.7.1. [XiEnv_LoadDsktop](#)
 - 10.7.2. [XiEnv_SaveDsktop](#)
 - 10.7.3. [XiEnv_CreateDialog](#)
 - 10.7.4. [XiEnv_AddDialogItem](#)
 - 10.7.5. [XiEnv_ShowDialog](#)
 - 10.7.6. [XiEnv_IsDialogClosed](#)
 - 10.7.7. [XiEnv_SelectSheet](#)
 - 10.7.8. [XiEnv_AddSheet](#)
 - 10.7.9. [XiEnv_DeleteSheet](#)

- 10.7.10. [XiEnv_RenameSheet](#)
- 10.7.11. [XiEnv_OpenWindow](#)
- 10.7.12. [XiEnv_CloseWindow](#)
- 10.7.13. [XiEnv_DeleteWindow](#)
- 10.7.14. [XiEnv_ImportWindow](#)
- 10.7.15. [XiEnv_ExportWindow](#)
- 10.8. [Blackboard](#)
 - 10.8.1. [XiEnv_AddVari](#)
 - 10.8.2. [XiEnv_RemoveVari](#)
 - 10.8.3. [XiEnv_AttachVari](#)
 - 10.8.4. [XiEnv_Get](#)
 - 10.8.5. [XiEnv_GetPhys](#)
 - 10.8.6. [XiEnv_Set](#)
 - 10.8.7. [XiEnv_SetPhys](#)
 - 10.8.8. [XiEnv_Equ](#)
 - 10.8.9. [XiEnv_WrVariEnable](#)
 - 10.8.10. [XiEnv_WrVariDisable](#)
 - 10.8.11. [XiEnv_IsWrVariEnable](#)
 - 10.8.12. [XiEnv_LoadRefList](#)
 - 10.8.13. [XiEnv_SaveRefList](#)
 - 10.8.14. [XiEnv_ExportRobFile](#)
 - 10.8.15. [XiEnv_GetVariConversionType](#)
 - 10.8.16. [XiEnv_GetVariConversionString](#)
 - 10.8.17. [XiEnv_SetVariConversion](#)
 - 10.8.18. [XiEnv_GetVariType](#)
 - 10.8.19. [XiEnv_GetVariUnit](#)
 - 10.8.20. [XiEnv_SetVariUnit](#)
 - 10.8.21. [XiEnv_GetVariMin](#)
 - 10.8.22. [XiEnv_GetVariMax](#)
- 10.9. [XiEnv_SetVariMin](#)
 - 10.9.1. [XiEnv_SetVariMax](#)
 - 10.9.2. [XiEnv_GetNextVari](#)
 - 10.9.3. [XiEnv_GetNextVariEx](#)
 - 10.9.4. [XiEnv_GetVariEnum](#)
 - 10.9.5. [XiEnv_WriteFrame](#)
 - 10.9.6. [XiEnv_GetFrame](#)
 - 10.9.7. [XiEnv_WriteFrameWaitReadFrame](#)
 - 10.9.8. [XiEnv_ImportVariProperties](#)
 - 10.9.9. [XiEnv_EnableRangeControl](#)
 - 10.9.10. [XiEnv_DisableRangeControl](#)
 - 10.9.11. [XiEnv_GetRaw](#)
 - 10.9.12. [XiEnv_SetRaw](#)
- 10.10. [Calibration](#)
 - 10.10.1. [XiEnv_LoadSvl](#)
 - 10.10.2. [XiEnv_SaveSvl](#)
 - 10.10.3. [XiEnv_ReferenceSymbol](#)

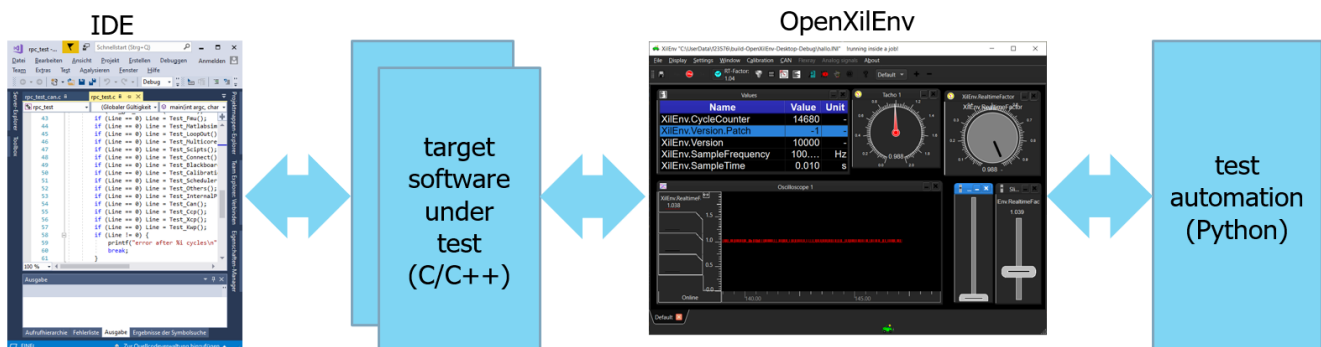
- 10.10.4. [XilEnv_DereferenceSymbol](#)
- 10.10.5. [XilEnv_GetSymbolRaw](#)
- 10.10.6. [XilEnv_SetSymbolRaw](#)
- 10.11. [Calibration with A2L file](#)
 - 10.11.1. [XilEnv_SetupLinkToExternProcess](#)
 - 10.11.2. [XilEnv_GetLinkToExternProcess](#)
 - 10.11.3. [XilEnv_GetIndexFromLink](#)
 - 10.11.4. [XilEnv_GetNextSymbolFromLink](#)
 - 10.11.5. [XilEnv_GetDataFromLink](#)
 - 10.11.6. [XilEnv_SetDataToLink](#)
 - 10.11.7. [XilEnv_ReferenceMeasurementToBlackboard](#)
 - 10.11.8. [XilEnv_DereferenceMeasurementFromBlackboard](#)
 - 10.11.9. [XilEnv_GetLinkDataType](#)
 - 10.11.10. [XilEnv_GetLinkDataArrayCount](#)
 - 10.11.11. [XilEnv_GetLinkDataArraySize](#)
 - 10.11.12. [XilEnv_CopyLinkData](#)
 - 10.11.13. [XilEnv_FreeLinkData](#)
 - 10.11.14. [XilEnv_GetLinkSingleValueDataType](#)
 - 10.11.15. [XilEnv_GetLinkSingleValueTargetDataType](#)
 - 10.11.16. [XilEnv_GetLinkSingleValueFlags](#)
 - 10.11.17. [XilEnv_GetLinkSingleValueAddress](#)
 - 10.11.18. [XilEnv_GetLinkSingleValueDimensionCount](#)
 - 10.11.19. [XilEnv_GetLinkSingleValueDimension](#)
 - 10.11.20. [XilEnv_GetLinkSingleValueData\[Double/Int/UInt\]](#)
 - 10.11.21. [XilEnv_SetLinkSingleValueData\[Double/Int/UInt\]](#)
 - 10.11.22. [XilEnv_GetLinkSingleValueDataString](#)
 - 10.11.23. [XilEnv_GetLinkSingleValueDataStringPtr](#)
 - 10.11.24. [XilEnv_SetLinkSingleValueDataString](#)
 - 10.11.25. [XilEnv_GetLinkSingleValueUnit](#)
 - 10.11.26. [XilEnv_GetLinkSingleValueUnitPtr](#)
 - 10.11.27. [XilEnv_GetLinkArrayValueDataType](#)
 - 10.11.28. [XilEnv_GetLinkArrayValueTargetDataType](#)
 - 10.11.29. [XilEnv_GetLinkArrayValueFlags](#)
 - 10.11.30. [XilEnv_GetLinkArrayValueAddress](#)
 - 10.11.31. [XilEnv_GetLinkArrayValueDimensionCount](#)
 - 10.11.32. [XilEnv_GetLinkArrayValueDimension](#)
 - 10.11.33. [XilEnv_GetLinkArrayValueData\[Double/Int/UInt\]](#)
 - 10.11.34. [XilEnv_SetLinkArrayValueData\[Double/Int/UInt\]](#)
 - 10.11.35. [XilEnv_GetLinkArrayValueDataString](#)
 - 10.11.36. [XilEnv_GetLinkArrayValueDataStringPtr](#)
 - 10.11.37. [XilEnv_SetLinkArrayValueDataString](#)
 - 10.11.38. [XilEnv_GetLinkArrayValueUnit](#)
 - 10.11.39. [XilEnv_GetLinkArrayValueUnitPtr](#)
- 10.12. [OpenXilEnv for HiL](#)
- 10.13. [CAN](#)
 - 10.13.1. [XilEnv_LoadCanVariante](#)

- 10.13.2. [XiEnv_LoadAndSelCanVariante](#)
 - 10.13.3. [XiEnv_DellAllCanVariants](#)
 - 10.13.4. [XiEnv_TransmitCAN](#)
 - 10.13.5. [XiEnv_SetCanErr](#)
 - 10.13.6. [XiEnv_SetCanErrSignalName](#)
 - 10.13.7. [XiEnv_ClearCanErr](#)
 - 10.13.8. [XiEnv_SetCanSignalConversion](#)
 - 10.13.9. [XiEnv_ResetCanSignalConversion](#)
 - 10.13.10. [XiEnv_ResetAllCanSignalConversion](#)
- 10.14. [CCP-Protocol](#)
 - 10.14.1. [XiEnv_LoadCCPConfig](#)
 - 10.14.2. [XiEnv_StartCCPBegin](#)
 - 10.14.3. [XiEnv_StartCCPAddVar](#)
 - 10.14.4. [XiEnv_StartCCPEnd](#)
 - 10.14.5. [XiEnv_StopCCP](#)
 - 10.14.6. [XiEnv_StartCCPCalBegin](#)
 - 10.14.7. [XiEnv_StartCCPCalAddVar](#)
 - 10.14.8. [XiEnv_StartCCPCalEnd](#)
 - 10.14.9. [XiEnv_StopCCPCal](#)
- 10.15. [XCP-Protocol](#)
 - 10.15.1. [XiEnv_LoadXCPConfig](#)
 - 10.15.2. [XiEnv_StartXCPBegin](#)
 - 10.15.3. [XiEnv_StartXCPCalAddVar](#)
 - 10.15.4. [XiEnv_StartXCPEnd](#)
 - 10.15.5. [XiEnv_StopXCP](#)
 - 10.15.6. [XiEnv_StartXCPCalBegin](#)
 - 10.15.7. [XiEnv_StartXCPCalAddVar](#)
 - 10.15.8. [XiEnv_StartXCPCalEnd](#)
 - 10.15.9. [XiEnv_StopXCPCal](#)
- 11. [Appendix](#)
 - 11.1. [Transfer parameter](#)
 - 11.1.1. [-ini](#)
 - 11.1.2. [-script](#)
 - 11.1.3. [-Instance](#)
 - 11.1.4. [-err2msg](#)
 - 11.1.5. [--ScriptErrExit](#)
 - 11.1.6. [--exit_on_script_syntax_error](#)
 - 11.1.7. [--exit_on_script_execution_error](#)
 - 11.1.8. [-icon](#)
 - 11.1.9. [-nogui](#)
 - 11.1.10. [-Port](#)
 - 11.1.11. [-sys \(remote master only\)](#)
 - 11.1.12. [Transfer parameter of an external process](#)
 - 11.2. [Environment variables](#)
 - 11.2.1. [ExternalProcess_CallFrom](#)
 - 11.2.2. [ExternalProcess_Instance](#)

- 11.2.3. [XiLEnv_NoGui](#)
 - 11.2.4. [XILENV_NO_XCP](#)
 - 11.2.5. [FMU_EXTRACTOR_CMD](#)
- 11.3. [Buildin-functions in formula](#)
 - 11.3.1. [Mathematical buildin-functions](#)
 - 11.3.2. [Buildin-function for bitoperations](#)
 - 11.3.3. [CAN Buildin-functions](#)
 - 11.3.4. [other buildin-functions](#)
- 11.4. [INI-/DSK-Files](#)
- 11.5. [CAN files](#)
- 12. [OpenXiL-Env for HiL](#)
 - 12.1. [General information of OpenXiLEnv for HiL](#)
 - 12.2. [Differences between OpenXiLEnv and OpenXiLEnv for HiL](#)
 - 12.3. [Hardware-Requirements](#)
 - 12.3.1. [OpenXiLEnv for HiL](#)
 - 12.4. [CAN-Bus](#)
 - 12.4.1. [CAN-configuration](#)
 - 12.4.2. [Insert data-error on the CAN-bus](#)
 - 12.4.3. [CCP-application](#)
 - 12.5. [Integration of a model](#)

2. What is OpenXiLEnv

OpenXiLEnv is primarily a Software in the loop environment for testing embedded software without target hardware. It has a graphical user interface. It is automatable, repeatable and freezable.



2.1. Use case scenarios

1. Software in the loop environment (SiL)
2. Model in the loop environment (MiL)
3. Open loop environment
4. Module test environment
5. Complete digital twin test
6. Manuel tests

7. Automated tests:

1. Simple integrated script language
2. RPC interface (Python, Basic, C++, EXAM,...)

8. Rapid prototyping

9. Offline calibration

10. Mini Hardware in the Loop environment (HiL)

2.2. Platforms

1. Host operating systems:

1. Windows 10
2. Linux

2. Integrated Development Environment (IDE):

1. Eclipse (GCC)
2. Visual Studio
3. Matlab Simulink (Co-Simulation)

3. Targets:

1. 64 bit
2. 32 bit
3. Functional Mock-up Unit (FMU)
4. QEMU (planned)

2.3. Prime advantages

1. No target hardware needed
2. No target development environment needed
3. FMI interface for FMU support
4. Memory protection against each other
5. Multicore support
6. Virtual network CAN FD with rest bus simulation
7. XCP over ethernet support (INCA, CANape)
8. Important for cloud environment:

1. GUI less variant
2. Unlimited instances
3. No licence needed
4. No installation necessary

9. HIL options:

1. Mini HIL with CAN(FD) interfaces

3. Installation

3.1. General information for installation

The installation files of OpenXilEnv can be found in the compile directory

The directories contain one or more of the following programme packages OpenXilEnv, the remote control API (remote_control), the surrounding tools and the the deprecated OpenXil-Env for HiL. OpenXil-Env for HiL have a hardware-connection for a lab car. Therefore a special hardware is required. For detail see Section [OpenXil-Env for HiL](#) below.

3.2. Installing OpenXilEnv

You don't need to install OpenXilEnv you can start it directly from anywhere

3.3. Setup a new project in OpenXilEnv

1. All files from the subdirectory ... TODO. The files have the following meaning:

File	Usage
XilEnvGui.exe	OpenXilEnv application itself (it is a 64bit application regardless its name)
XilEnv.exe	OpenXilEnv application with no graphical user interface. it has the same functionality as XilEnvGui.exe. This is usefull for automation background testing witout user interaction.
Open_XilEnv_userguide_en.pdf	This document
include	Directory with header for own projects
include\XilEnvExtProc.h	OpenXilEnv-header for external processes
include\XilEnvExtProcCan.h	Declaration of the virtual CAN-interface
include\XilEnvExtProcMain.c	This file include a main() function implementation for an user defined external process using the DLL version of the interface to OpenXilEnv (XilEnvExtProc64/32.dll).
XilEnvExtProc32.dll	Communication DLL for 32 bit executable
XilEnvExtProc64.dll	Communication DLL for 64 bit executable

File	Usage
XilEnvRpc.dll	Rmot procedure call DLL
ExtProc_FMUExtract.exe	Extern process needed to extract a FMU file
ExtProc_FMULoader32.exe	Extern process needed to execute a 32 bit FMU
ExtProc_FMULoader64.exe	Extern process needed to execute a 64 bit FMU
Samples64/32	Directory with some sample projects

- 2. Start XilEnvGui without parameter
- 3. Choose generate a new INI file
- 4. Start Visual Studio or respectively Eclipse
- 5. Open a sample project from the Sample subdirectory
- 6. Rebuild the project

3.4. Installing OpenXiLEnv HIL option

Todo

3.5. Default setting

Through the menu item **Settings** => **Basic Settings** the default setting can be adjusted.

Basic settings

?

×

Main settings

Display settings

INI-File settings

Script settings

Remote procedure call

Remote master

Project name:

Default Title

1

Work directory:

%TEMP%\.

2

3.1

Scheduling period:

3

0.01

is (resolution are 1ns)

☐

Sync with flexray channel 0

Blackboard elements:

4

100000

Editor:

NOTEPAD.EXE

5

Priority:

PRIORITY_IDLE

6

☒ remember manual referenced labels

7

☒ not faster than realtime

8

☐ no case senisitive filters

9

☐ replace :: with ._ in labelname (ASAP compatible)

10

☐ replace all other non ASAP combatible chars (not a-z, A-Z, 0-9, [,], _) with

11

☐ ".[xx]" xx are the hexadecimal value of the character

11.1

☐ "._[xx]._ " xx are the hexadecimal value of the character

11.2

☐ use relative paths

12

☐ store reference lists only none persistence

13

☒ stop scheduler while a dialog is open

14

☒ allow external process communication over socket

port

1800

15

OK

Cancel

1. Titel bar of the project, this has no effect.

```
[BasicSettings]
...
Default Title=This is a sample project
...
```

2. Workspace where OpenXilEnv switch at start up

```
[BasicSettings]
...
Work Dir=c:\user\myhome\work
...
```

3. Defines the sample time of the whole system. This must be the smallest possible cycle time of any process.

```
[Scheduler]
...
Period=0.01
...
```

4. Max. number of possible blackboard variables

```
[BasicSettings]
...
BlackBoardSize=10000
...
```

5. used editor

```
[BasicSettings]
...
Editor=Notepad.exe
...
```

6. Which priority does OpenXilEnv have under Windows (ignored by HiL). Best setting is „PRIORITY_IDLE". This means every other program have priority. Possible values are:

Prorotity name	Description
PRIORITY_NORMAL_NO_SLEEP	OpenXilEnv have the same prorotity as all other applications. If it have to wait (not faster than realtime) it will not call a sleep function it will wait inside a occupation loop. This will be result in thebest realtime behaviour.
PRIORITY_NORMAL	OpenXilEnv have the same prorotity as all other applications. If it have to wait (not faster than realtime) it will call a sleep function.
PRIORITY_BELOW_NORMAL	OpenXilEnv have a lower prorotity as all other applications.
PRIORITY_LOWEST	OpenXilEnv have a lowest prorotity.
PRIORITY_IDLE	OpenXilEnv will only run if no other application want to be executed. This is the default behaviour.

```
[BasicSettings]
...
```

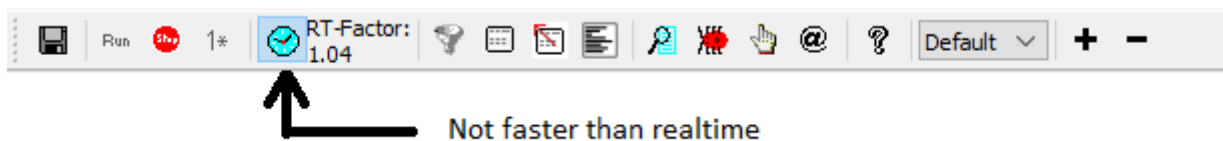
```
Priority=PRIORITY_IDLE
...
```

7. If this is checked all manual references take place inside an calibration window will be saved inside the configuration file (INI). And will be referenced after restart the external process or OpenXilEnv. Default is 1.

```
[BasicSettings]
...
remember manual referenced labels=1
...
```

8. If this is checked OpenXilEnv will not run faster than realtime. It will wait if it executed a cycle to fast. This will be ignored inside OpenXilEnv for HiL.

```
[BasicSettings]
...
not faster than real time=1
...
```



9. If this is checked all filter inside OpenXilEnv are not case sensitive. Default is 0.

```
[BasicSettings]
...
no case sensitive filters=1
...
```

10. If this is checked all references will be checked for A2L confirm labels name. If the label name include a ":" this will be replaced by "._". Default is 0.

```
[BasicSettings]
...
ASAP compatible labelnames=1
...
```

11. If this is checked all references will be checked for A2L confirm labels name. All none A2L conform characters would be replaced with it hexadecimal value. You can select between ".[xx]" (1) or "._[xx]._" (2)

sequence. Default is 0.

```
[BasicSettings]
...
ReplaceAllNoneAsapCompatibleChars=2
...
```

12. If this is checked OpenXilEnv try to truncate all selected paths to a relative path (if possible). Default is 0.

```
[BasicSettings]
...
use relative paths=1
...
```

13. Will be ignored see 7.

```
[BasicSettings]
...
StoreReferenceListsOnlynonePersistence=yes
...
```

14. If this is checked the simulation time would be stopped until a dialog is open. This should not be activate if realtime behaviour is expected. Default is 1.

```
[BasicSettings]
...
stop scheduler while a dialog is open=1
...
```

15. If this is checked external processes can connect to OpenXilEnv also over sockets. You have to define a port number. OpenXilEnv will listen on this port to incoming connections. Default is 0. This can be overloaded by the transfer parameter -port.

```
[BasicSettings]
...
ExternProcessLoginSocketPort=1800
...
```

Protect process list against changes

16. Protect complete INI-file against changes 8.1 when quit the program demand if changes should be discard
17. Automatic referencing after restart of manually labels referenced in the application window
18. OpenXiLEnv should not run faster than real time. OpenXiLEnv tries to comply with the scheduling period set in point 3 (ignored by by the HIL option).
19. Show physical units for variables without conversion dez=>phys.
20. Activate the green car in the status-bar with a set or variable speed. A formula for the calculation of the speed can be specified here.
21. Filter for variable-selection-dialogue is not case sensitive.
22. Within the ASAP-file no „:“ are allowed, so they are replaced through „_“.
23. Speeds up the read of the debug-information for a Boorland-project, because the data is not transferred though a file but a pipe.
24. Show modul static variables in the application window. Attention: as a result there could be unclear labels. Modul static labels can be within serveral modules with the same name (can be prevented with option 16.1) 16.1 Use the modul name as prefix to get clearly names.
25. Sort the sections of the INI-file alphabetically when saving.
26. Create a backup of the old INI-file "xxx.ini.bak" before saving it.
27. Eliminate the creation of a script-debug-file. For durantion test it would be very long. / Unterdrecks das Erstellen eines Script-Debug-Files. Bei Dauerlauf-Tests würde dieses sonst sehr lang werden

The settings "Project name", "Scheduling period" and "Blackb. elements" will be applied when OpenXiLEnv was restarted. 2KByte of memory will be reservated for every Blackboard-element. So there should not be too much unnecessary elements reservated.

The tab "display settings" configured the the behaviour of the main window and some child windows.

Basic settings

Main settings | Display settings | INI-File settings | Script settings | Remote procedure call | Remote master

☐ show units for none physical values **1**

☒ show status car **2** speed:

☒ display static object (caution about double symbols) **3**

☐ to avoid double symbols add source file name to symbol name (only for Visual C) **4**

☐ supress none existing values **5**

☒ select startup application style: **6**

New displays window default settings

7

Text | Oscilloscope

☒ show unit column

☒ show display type column

☐ default font name and size:

1. Show physical units for variables without conversion `dez=>phys`. Default is 0.

```
[BasicSettings]
...
display unit for none physical values=1
...
```

2. Activate the green car in the status-bar with a set or variable speed. A formula for the calculation of the speed can be specified here. Default is 1.

```
[BasicSettings]
...
StatusbarCar=1
StatusbarCarSpeed=n_ab/1000
...
```

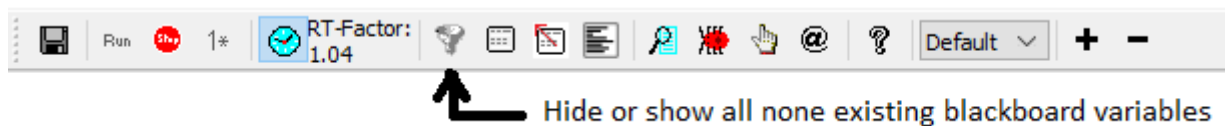

3. If this is checked all module static symbols inside an external processes are visible. They would be displayed inside the calibration tree and inside SVL files. Remark. static symbols can be existing more than once with the same name inside an executable. You can select 4. to avoid double symbols. Default is "no"

```
[BasicSettings]
...
ReadModuleStaticLabels=yes
...
```

4. If this is checked each static symbol would be extended with the module file name as prefix. Default is "no". you have to select 4. before.

```
[BasicSettings]
...
ExtendStaticLabelsWithFilename=yes
...
```

5. If this is checked all none existing blackboard variables would be hide inside all text and oscilloscope windows. Default is 0.

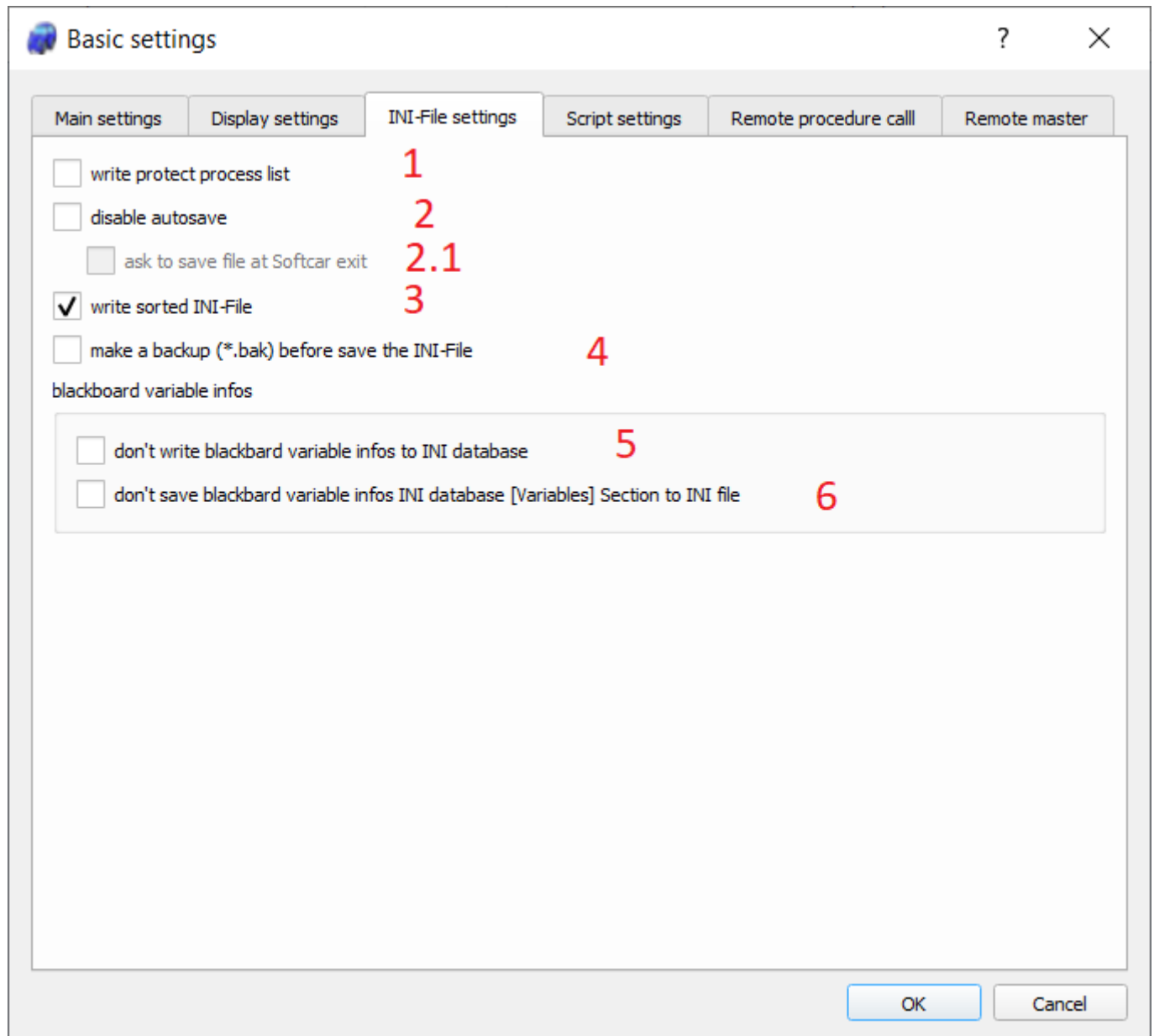


```
[BasicSettings]
...
suppress the display of non existing values=1
...
```

6. With this item you can select the basic look of OpenXilEnv. Possible values are.
7. The default values if a new window would be created (at the moment only some parameters of text or oscilloscope windows are available).

```
[BasicSettings]
...
OscilloscopeDefaultBufferDepth=1000000
OscilloscopeDefaultFont=Courier, 12
TextDefaultFont=MS Serif, 14
TextDefaultShowDisplayTypeColumn=no
TextDefaultShowUnitColumn=yes
...
```

The tab "INI file settings" configured the behaviour of the INI file handling.



1. Protect process list against changes. If set the process list inside the INi configuration file will not change if you add or remove a process. Default is 0.

```
[start processes]
...
write protect=1
...
```

2. Protect complete INI-file against changes. If you close OpenXilEnv the INI file will not saved automatically. But you can save it with the menu entry save or save as. 2.1 when quit OpenXilEnv demand if changes should be discard.

```
[BasicSettings]
...
switch off the automatic save of the INI-File=1
ask to save the INI-File at OpenXilEnv exit=0
...
```

3. If this is checked the INI configuration file is saved alphabetically sorted. So you can better diff changes inside the INI configuration file. Default is "no".

```
[BasicSettings]
...
SaveSortedINIFiles=yes
...
```

4. Create a backup of the old INI-file "xxx.ini.bak" before saving it. Default value is "no".

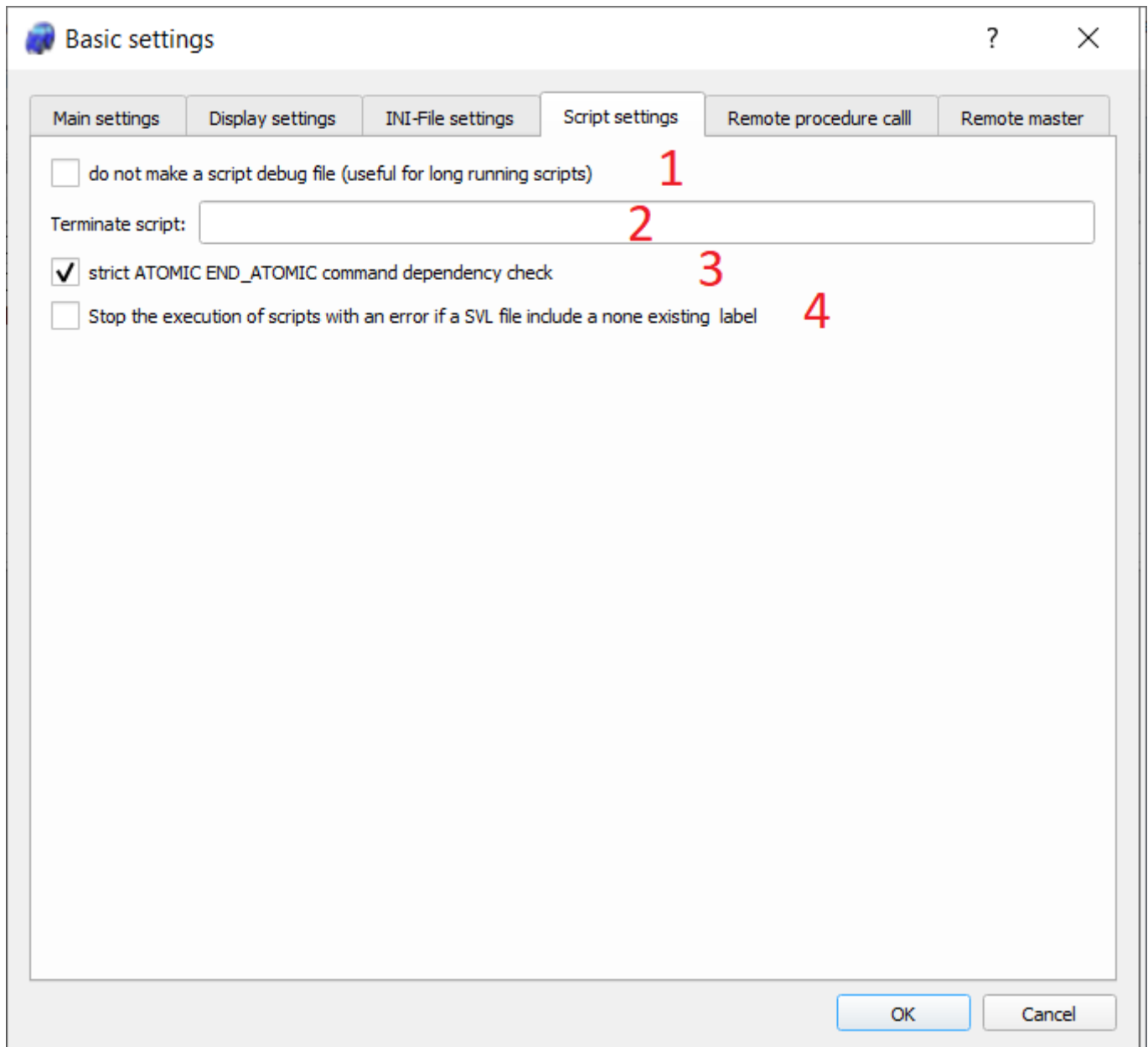
```
[BasicSettings]
...
MakeBackupBeforeSaveINIFiles=yes
...
```

5. If this is checked the property of blackboard variables will not stored inside the [Variables] section of the INI configuration file. Default value is 0.

```
[BasicSettings]
...
DontWriteBlackbardVariableInfosToIni=yes
...
```

6. [BasicSettings] ... DontSaveBlackbardVariableInfosIniSection=yes ...

The tab "script settings" includes all settings for the script:



1. Eliminate the creation of a script-debug-file. For duration test it can be accumulate to a very large file. Default value is "no".

```
[BasicSettings]
...
MakeBackupBeforeSaveINIFiles=no
...
```

2. Here can be defined a termination script. If OpenXiLEnv will be closed this script will be executed. The termination would be postpone till the termination script is finished. The termination script should not be long to avoid convulsion.

```
[BasicSettings]
...
TerminaterScript=path-to-script\terminate.scr
...
```

3. If this is checked the script parser make sure that all ATOMIC command will be have a corresponding END_ATOMIC command. This option is only usefull for old scripts. Normaly this option should stay checked. Default value is "yes".

```
[BasicSettings]
...
  SctrictAtomicEndAtomicCheck=yes
...
```

4. If this is checked scripts will be stopped with an error if a SVL loaded with the LOAD_SVL command has none exiting labels or other error. Default value is "no".

```
[BasicSettings]
...
  StopScriptIfLabelDosNotExistInsideSVL=yes
...
```

The tab "remote procedure call" configured the RPC interface of OpenXilEnv. If you change someting inside this tab you have to restart OpenXilEnv to take place the changes:

Basic settings

Main settings Display settings INI-File settings Script settings Remote procedure call Remote master

You have to restart softcar to take place your changes

Connection type

☒ Named pipes (only windows) 1

☐ Sockets Port: 5810

Debugging (log file) 2

☐ Write command name to log file

☐ Write parameter as hex to log file

☒ Write parameter decoded to log file

☐ Write return value as hex to log file

☒ Write return value decoded to log file

☒ Flush log file for each witten command

☐ Attach to existing log file

☐ Write all connections to one log file

OK Cancel

1. Here you can select if OpenXilEnv should be accessible through named pipes or sockets. Only sockets can be used to control OpenXilEnv over network. If sockets are used a port number on which OpenXilEnv should listen must be defined. Default value is "NamedPipe"

```
[BasicSettings]
...
RpcOverSocketOrNamedPipe=Socket
RpcSocketPort=5810
...
```

2. If this is checked OpenXilEnv will write a logging file where all executed RPC commands are logged. All following checkboxes are selecting which information of the commands should be written to the logging file. Remark: this fill can be increase fast. Default value is 0.

```
[BasicSettings]
...
```

```
RpcDebugLoggingFlags=1049
```

```
...
```

The value is a bit mask and has the following definition:

```
#define DEBUG_PRINT_COMMAND_NAME          1
#define DEBUG_PRINT_COMMAND_PARAMETER_AS_HEX  2
#define DEBUG_PRINT_COMMAND_RETURN_AS_HEX    4
#define DEBUG_PRINT_COMMAND_PARAMETER_DECODED 8
#define DEBUG_PRINT_COMMAND_RETURN_DECODED  16
#define DEBUG_PRINT_COMMAND_FFLUSH          1024
#define DEBUG_PRINT_COMMAND_ATTACH          2048
#define DEBUG_PRINT_COMMAND_ONE_FILE        4096
```

The tab "remote master" is only usefull for the HIL option. Here you can configured the connection to the second linux realtime PC. If you change someting inside this tab you have to restart OpenXilEnv to take place the changes:

The screenshot shows the 'Basic settings' dialog box with the 'Remote master' tab selected. Red numbers 1 through 7 highlight the following elements:

- 1: ☒ Enable remote master
- 2: Remote master name (address): 192.168.1.3
- 3: Remote master port number: 8888
- 4: ☒ Copy and start remote executable
- 5: Remote master executable: %SC_EXE_DIR%\linux\LinuxRemoteMaster.out
- 6: Copied/Loaded executable (-sys):
- 7: Copy remote master executable to: /tmp/LinuxRemoteMaster.out

Buttons at the bottom: OK, Cancel.

1. If this is checked OpenXilEnv will try to connect to the second linux PC. otherwise it will start as a normal OpenXilEnv instance without any hardware connections. Default value is "no".

```
[BasicSettings]
...
ConnectToRemoteMaster=yes
...
```

2. Here you should add the local ip address or name of the second realtime linux PC.

```
[BasicSettings]
...
RemoteMasterName=192.168.1.3
...
```


3. Here you should set the port number on which the second realtime linux PC is listening. This should be always 8888. this is hard coded inside the installation image of lthe linux PC.

```
[BasicSettings]
...
RemoteMasterPort=8888
...
```

4. If this is checked OpenXilEnv will copy and start the executable during startup each time. For debugging reason you can uncheck this and start the executable directly on the linux PC before OpenXilEnv started. Default value is "yes".

```
[BasicSettings]
...
CopyAndStartRemoteMaster=yes
...
```

5. If you have build an own executable with a model included you have to define the location of this image here. OpenXilEnv will copy this image during each start to the linux PC before it will execute it.

```
[BasicSettings]
...
RemoteMasterExecutable=%XILENV_EXE_DIR%\linux\LinuxRemoteMaster.out
...
```

6. This item is only for information you cannot change this. It will be display the model executable which is defined with the transfer parameter -sys. This will overload the definition inside item 4.
7. This defines the remote location where to copy the executable. This should be always "/tmp/LinuxRemoteMaster.out"

```
[BasicSettings]
...
RemoteMasterCopyTo=/tmp/LinuxRemoteMaster.out
...
```

If "switch off automatic save of INI-File" is activated, the INI-file will not be saved automatically at the end of program (all changes will be lost). Afterwards the files can be saved manually through the menu **File => Save** or **Save as** only.

3.6. Hotkeys

By using the menu item **Settings => Other Hotkeys** every letter-/number-key, in combination with the Alt-/Ctrl-keys, can get a function. By using the menu item **Settings => Function Hotkeys** the same allocation is possible for control keys. The Hotkey-configuration can be saved **Settings => Export Hotkeys** and loaded **Settings => Import Hotkeys** (Exchange of different INI-files).

The following functions are available:

- **"single shot equation"**: A calculation can be deposit here, which is executed for every press of a key.
E.g.: $\text{signal} = \text{signal} + 1$
- **"start script"**: Deposit a script-file here, that is executed when pressing the corresponding key.
- **"stop script"**: Stops running scripts when the corresponding key was pressed.
- **"change sheet"**: Toggle between several sheets.
- **"run control continue"**: Continue the scheduler.
- **"run control stop"**: Stop the scheduler.
- **"run control next"**: Continue the scheduler for the number of cycles defined in the control-panel.
- **"run control next on"**: Continue the scheduler for one cycle.

3.7. System requirements

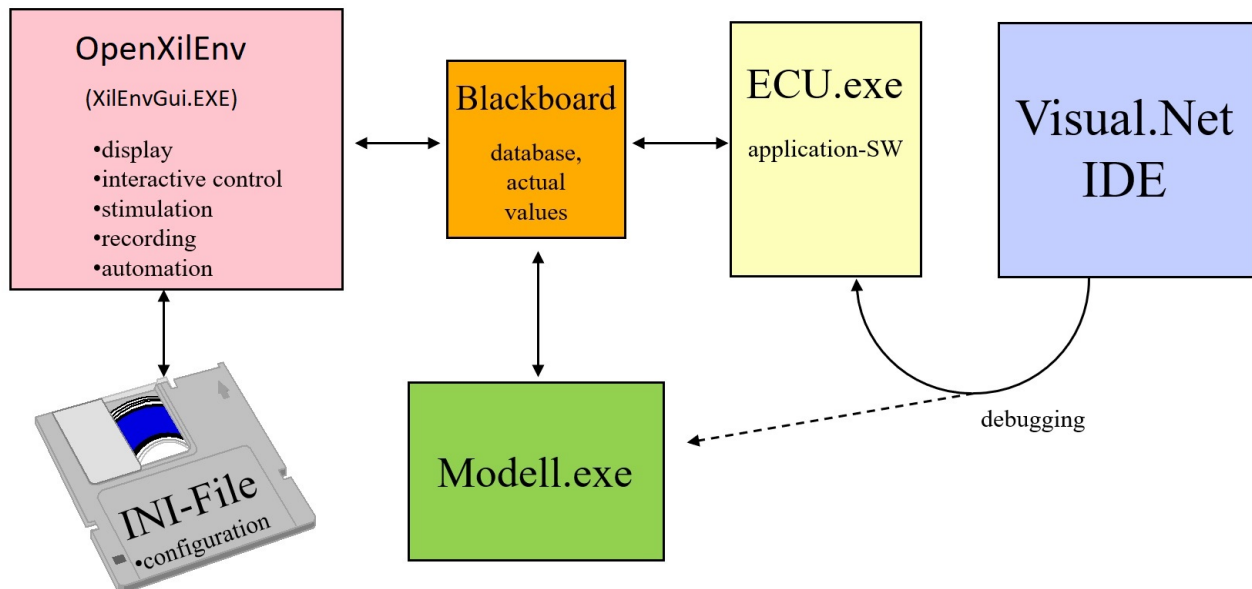
OpenXilEnv:

Todo

4. Concept

4.1. Concept

The test objects, model and OpenXilEnv are separated into own executable. The test object(s), model(s) will be calles external processes. Parts living inside OpenXilEnv will be called internal processes. The seperation into own executable have the advantage of memory protection agains each other. The external processes will be communicate with OpenXilEnv over named pipes or sockets. A library or a DLL is available that executes the access to the blackboard as well as register at OpenXilEnv. The use of the DLL or shared library is preferred to avoid compiler dependency.

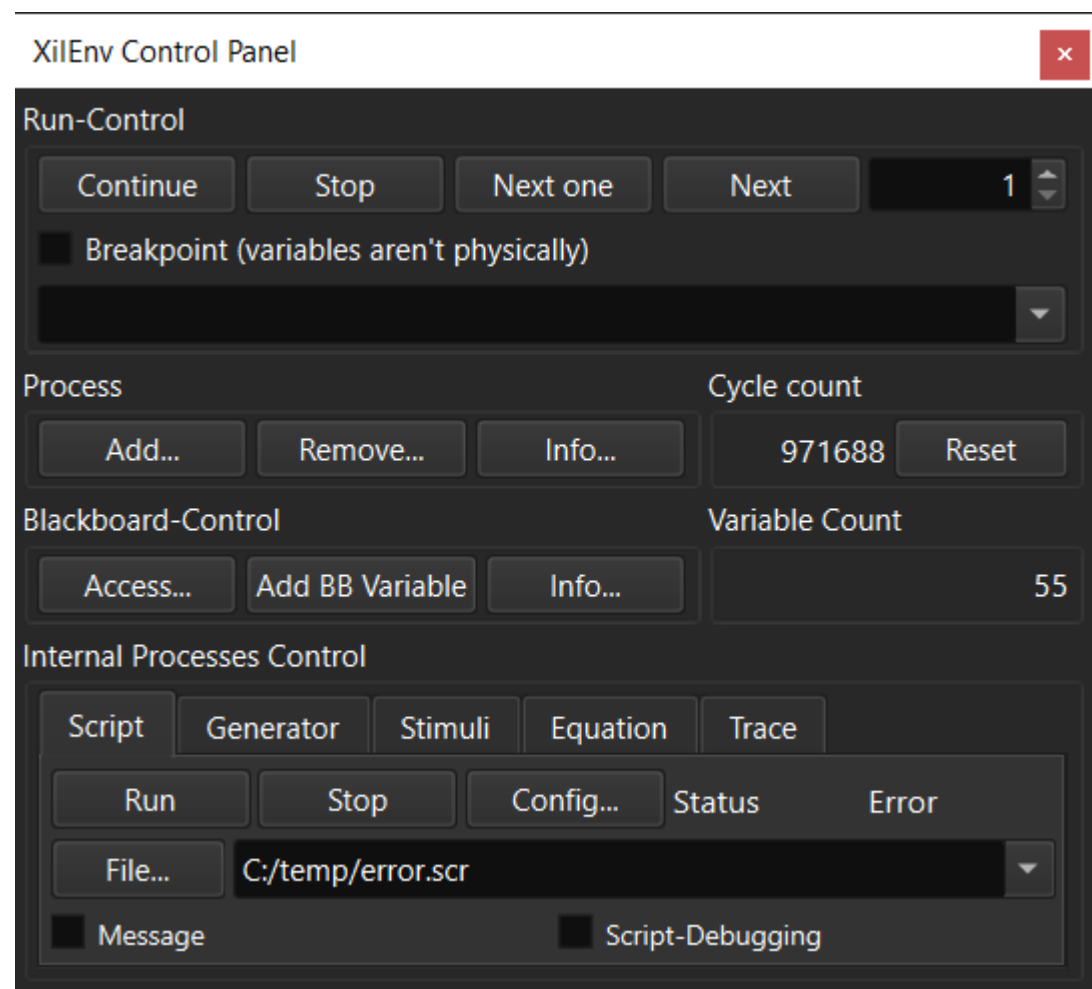


OpenXilEnv concept

Advantages of OpenXilEnv:

- Code size unlimited
- Data size unlimited
- Debugging in the IDE of VC.Net or Eclipse
- less name conflicts between OpenXilEnv modules and ECU- respectively model-modules
- OpenXilEnv stays active (display, controlling) when ECU-SW is stopped in the debugger.
- Critical errors (general violation of protection...) can be matched quicker.
- only 1 config-file

OpenXilEnv is separated into two windows: the main window for displaying and changing values and the control-panel for controlling the execution and the blackboard variable settings. Also the internal components (script, recorder, player, generator and the equation calculator) can be controlled there.



A screenshot of the main window with a text, enum, oscilloscope, calibration map, sliders, bargraphs, tachos, calibration tree and CAN message window:



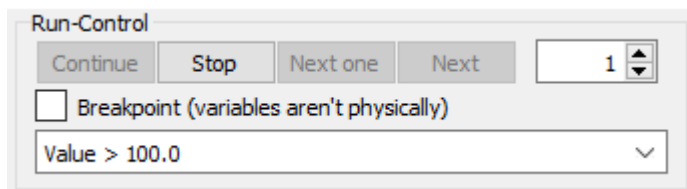
5. Control Panel

5.1. General information of the control panel

The internal functions are controlled through the control panel. It is always in the foreground and divided into four sections.

5.2. Run-Control

The execution of all processes can be interrupted manually or with a breakpoint in this section of the panel.



Continue -key continue execution of all processes (internal and external)

Stop -key stops execution of all processes (internal and external)

Next one -key will execute one cycle of all processes (internal and external)

Next -key will execute a number of cycles defined inside the right input box.

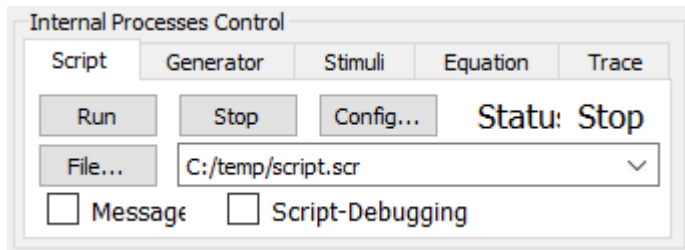
5.2.1. Breakpoint

If **Breakpoint** is activated there is a verification on the breakpoint condition at the end of a cycle. In this case the execution of every process will be stopped. Single step with **Next one-** or XX-step with **Next-key**.

Start-key continue all processes (internal and external)

"Breakpoints" can be set with STOP_OpenXiLEnv-command in the script-program. Also see in chapter „*Script debug window*".

5.3. Process



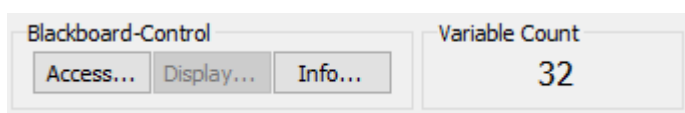
Process administration **Add...** offers a variety of all not yet started internal processes (should be empty in most cases). The button ext. process provides a dialogue box for file selection to start external processes (exe-files) manually at any time.

(Within the HIL option there is possibility to start external processes)

Remove... offers the possibility to shut down processes. There is no separation between internal and external processes. Retrice the current status of processes by the use of the ****Info...-**key**.

5.4. Blackboard

The blackboard is a kind of database of all current values and their setups.



There are the follwoing settings: name, unit, datatype, min-, max-value, format specification, step size, conversion formula or text replacement, color.

The settings can be found through blackboard-info-dialogue box (OpenXiLEnv control panel --> Blackboard --> Info..). All settings are saved as INI-file. The minimum and maximum values do not limit the values range of a variable but these values are used to display on the oscilloscope or tacho/knob.

The label has 511 characters, the unit has 63 characters and the conversion is limited to 130000 characters.

5.4.1. Conversion formula

When a conversion formula should be used for the display of a variable set the dialogue item 'Conversion type' to physical. Do not forget or OpenXiLEnv will not use the conversion formula!

Syntax:

The conversion formula is a standard formula, whereby the #-character is comply with the variable name. Alternatively a \$-character can be used. The following operations are allowed:

+, -, *, /, &, |, >, <, >=, <=, ==, !=, &&, ||, !, &, |, ^, >>, <<, (,), sin(), cos(), tan(), asin(), acos(), atan(), exp(), pow(), sqrt(), log(), log10(), and(in,bits), or(in,bits), xor(in,bits), invert(in), getbits(in,bitoffset,bitsize), setbits(in,bitoffset,bitsize,bits), andbits(in,bitoffset,bitsize,bits), orbits(in,bitoffset,bitsize,bits), xorbits(in,bitoffset,bitsize,bits), round(v), round_up(v), round_down(v), ...

Remaks: The data type of the variable is used. Constant can have three dirence types. First is a signed 64 bit integer second is a unsigned 64 bit integer and third is a 64 bit floatingpoint. A postfix can added to the constant value to clarify the data type of the constant. If an 'u' is added the constant is an unsigned integer value, if a 'i' is added the constant is a signed integer. If a ". 0" is added the constant is a floatingpoint value.

The data type of an operation result will be choose as follow: If one parameter is a floatingpoint value the result will be also a flotingpoint value. If all parameters of an operation will be an integer the result will be also an integer.

Examples:

```
50.0/100 = 0.5
50/100.0 = 0.5
50.0/100.0 = 0.5
50/100 = 0
```

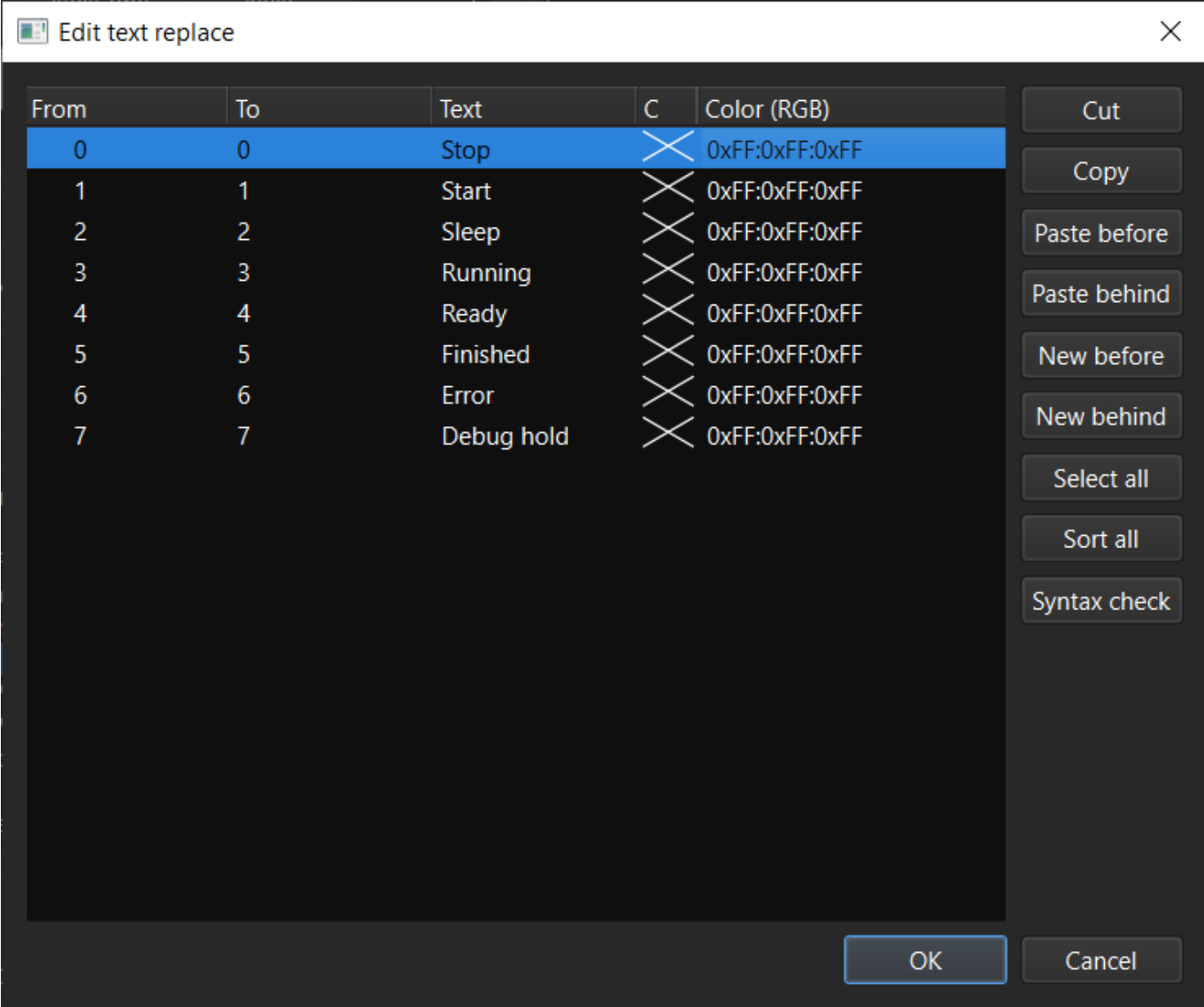
There will be happen no overflow if the result of an operation not fits in the min/max range of the resulting data type. It will be limited to the min/max range.

Example:

```
2.0 \* \# +1000.0
```

5.4.2. Text replacement

When a text replacement should be used for the display of a variable, set the dialogue item 'Conversion type' to "text replace". Do not forget or OpenXilEnv will not use the text replacement! Text replace "status" can get color information. In addition to that an additional dialogue (button "Edit Textrepl.") was created for "Blackboard-Info" wihtin the control panel. Currently the color information of enum-windows, tachometers, knobs, sliders, bargraphs and oscilloscopes can be displayed. This dialogue box looks as follows:



Additional to that the input can be made in a text line

Syntax:

The description of the text replacement consists of single blocks, that must be separated with semicolons. A block consists of a start- and end-value and a text replacement string. If the value of the variable \geq start value and \leq end value this block is evaluated as text replacement. Watch out for the ascending order of start- and end values. (start value block 1 \leq end value block 1 $<$ start value block 2 \leq end value block 2 ...). The text replacement string has to be put in quotation marks e.g.: "text" and additional to that it can contain color information: e.g.: "RGB(2e5:0x7F:0)Text". There must be RGB-macro **within** the quotation marks! It is recommended to use the configuration dialogue.

If a * is used as start value of the first block, it means -infinity. If a * is used as end value of the last block, it means +infinity. When no block is true, the text replacement is always "out of range".

Example: 0 0 "off"; 1 1 "on"; 2 * "undaf";

5.4.3. Supported data types

Name	Encoding	Resolution, Value range
------	----------	-------------------------

Name	Encoding	Resolution, Value range
BB_BYTE	0	8bit (-128...127)
BB_UBYTE	1	8bit (0...255)
BB_WORD	2	16bit (-32768...32767)
BB_UWORD	3	16bit (0...65535)
BB_DWORD	4	32bit (-2147483648...2147483647)
BB_UDWORD	5	32bit (0...4294967295)
BB_QWORD	34	64bit (-9223372036854775808...9223372036854775807)
BB_UQWORD	35	64bit (0...18446744073709551615)
BB_FLOAT	6	32bit-floating point
BB_DOUBLE	7	64bit-floating point

5.4.4. Fix registered variables

Name	Data type	Unit	Meaning
XilEnv.SampleFrequency	BB_DOUBLE	Hz	Sampling frequency
XilEnv.SampleTime	BB_DOUBLE	s	Sampling periode
XilEnv.Version	BB_UWORD	-	OpenXilEnv-version
XilEnv.Realtime	BB_UWORD	-	If the HIL options is active it is always 1 else it is 0
XilEnv.exit	BB_UWORD	-	When != 0 then exitOpenXilEnv
XilEnv.StimulusPlayer	BB_UBYTE	-	Status of the Stimuli-player (see 5.1.3)
XilEnv.StimulusPlayer.suspend	BB_UBYTE	-	Stops the player (when != 0)
XilEnv.TraceRecorder	BB_UBYTE	-	Status of the trace-recorders (see 5.1.4)
XilEnv.Generator	BB_UBYTE	-	Status of the signal generator (see 5.1.1)
XilEnv.EquationCalculator	BB_UBYTE	-	Status of the formula calculator (see 5.1.2)
XilEnv.Script	BB_UBYTE	-	Status of the script-file (see 5.1.5)
XilEnv.ExitCode	BB_UWORD	-	Exit code of the script-command called by START_EXE

5.4.5. System variables / environment variables

Environment variables of the Windows system can be used by %NAME%. Additional to that the OpenXilEnv-system occupies the following variables automatically which can be used as placeholder arbitrary (e.g. for relative path).

Name	Meaning
%XILENV_SCRIPT_NAME%	Name of the current executed script file
%XILENV_SCRIPT_DIR%	Directory of the current active script file
%XILENV_WORK_DIR%	Workspace from basic-settings
%XILENV_EXE_DIR%	Directory of OpenXiEnv-EXE
%XILENV_VARIABLE:Blackboardvariablenname%	Value of a blackboard-variable
%XILENV_PROCESS_DIR:Prozeßname%	Directory of an external process
%XILENV_CURRENT_DIR%	Current directory
%XILENV_CCP_SWSTAND%	Name of the software status read through the CCP. Only valid for activated CCP-connection otherwise it returns "error". Name of the software-status read through the CCP-connection 0 to 3.
%XILENV_CCP0_SWSTAND%	
%XILENV_CCP1_SWSTAND%	
%XILENV_CCP2_SWSTAND%	
%XILENV_CCP3_SWSTAND%	
%XILENV_XCP0_SWSTAND%	Name of the software-status read through XCP-connection 0 to 3. Only valid for activated XCP-connection otherwise it returns "error".
%XILENV_XCP1_SWSTAND%	
%XILENV_XCP2_SWSTAND%	
%XILENV_XCP3_SWSTAND%	
%XILENV_RANDOM%	Random number
%XILENV_TIME_STRING%	Time as string hh:mm:ss
%XILENV_TIME2_STRING%	Time as string hh.mm.ss no : but as separator
%XILENV_DATE_STRING%	Date as string
%XILENV_WINDOWS_TICK_COUNTER%	Returns time in milliseconds since start of the Windows system (overflow after around 49,7 days).

5.5. Internal Process Control

Is used to controll the internal processes. The corresponding process is chosen in the upper list. The single internal processes are described in (TODO:chapter 5.1).

Key	Meaning
Run	loads the chosen file into the corresponding process
Stop	stops the execution of the chosen file
Config...	for script, generator and equation an editor is opened to edit the chosen file, for trace and stimuli open corresponding dialogues
File...	opens a dialogue box for file selection
Message	opens a script output window as soon as an output via MESSAGE-command occurs (see 5.1.5).

6. Display of variables

6.1. General information for the display of variables

To display variables there are two types of representation available, the text window and the oscilloscope window.

6.2. Text window

New text windows can be created through the menu **Display --> New --> Text**. New text windows get the name Blackboard(1..n) automatically. Once created and closed windows can be reopened through **Display --> Open --> Text** .



The screenshot shows a window titled "Electric motor" with a table of variables. The table has three columns: Name, Value, and Unit. The variables listed are MotorCurrent, MotorErrorState, MotorSpeed, MotorTorque, and MotorVoltage. MotorErrorState is highlighted in green and shows the value "OK".

Name	Value	Unit
MotorCurrent	0.00	A
MotorErrorState	OK	-
MotorSpeed	0.00	rpm
MotorTorque	0.00	Nm
MotorVoltage	0.00	V

Adding further variables is always possible by double clicking with the left mous button onto the headpiece (name value) or through the menu **Display --> EditVars...** .

For manual modificaion of variables there ware two possibilities:

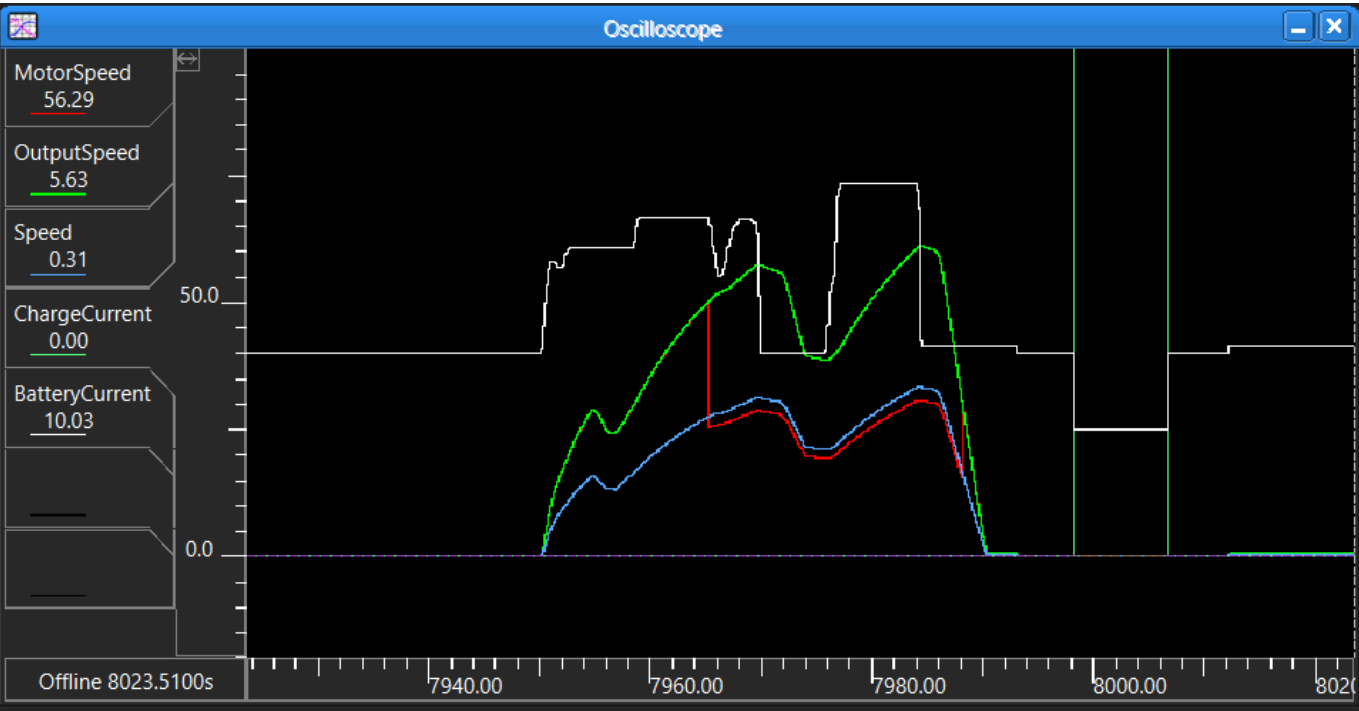
- 1. Click on the variable and increase (+ button) or decrease (- button)the value of the variable. By clicking the "Ctrl"- or "Shift"-key the increment or decrement can be multiplied by 10 or 100.
- 2. Double click the variable and type in the value directly (only possible for decimal and without conversion).

Form of representation:

Type	Description
decimal	Display of the variable without conversion into decimal
hexadecimal	Display of the variable without conversion into hexadecimal
binary	Display of the variable without conversion into binary
physical	Display of the variable value, built through conversion formula respectively display of the text found in the text replacement description

The form of representation is set by the "change variablevalue"-dialogue. So double click on the corresponding variable.

6.3. Oscilloscope-window



6.3.1. Default settings

To change the default settings do a right-click into the oscilloscope-window or open the "oscilloscope settings"-dialogue through the menu **Display --> Config**.

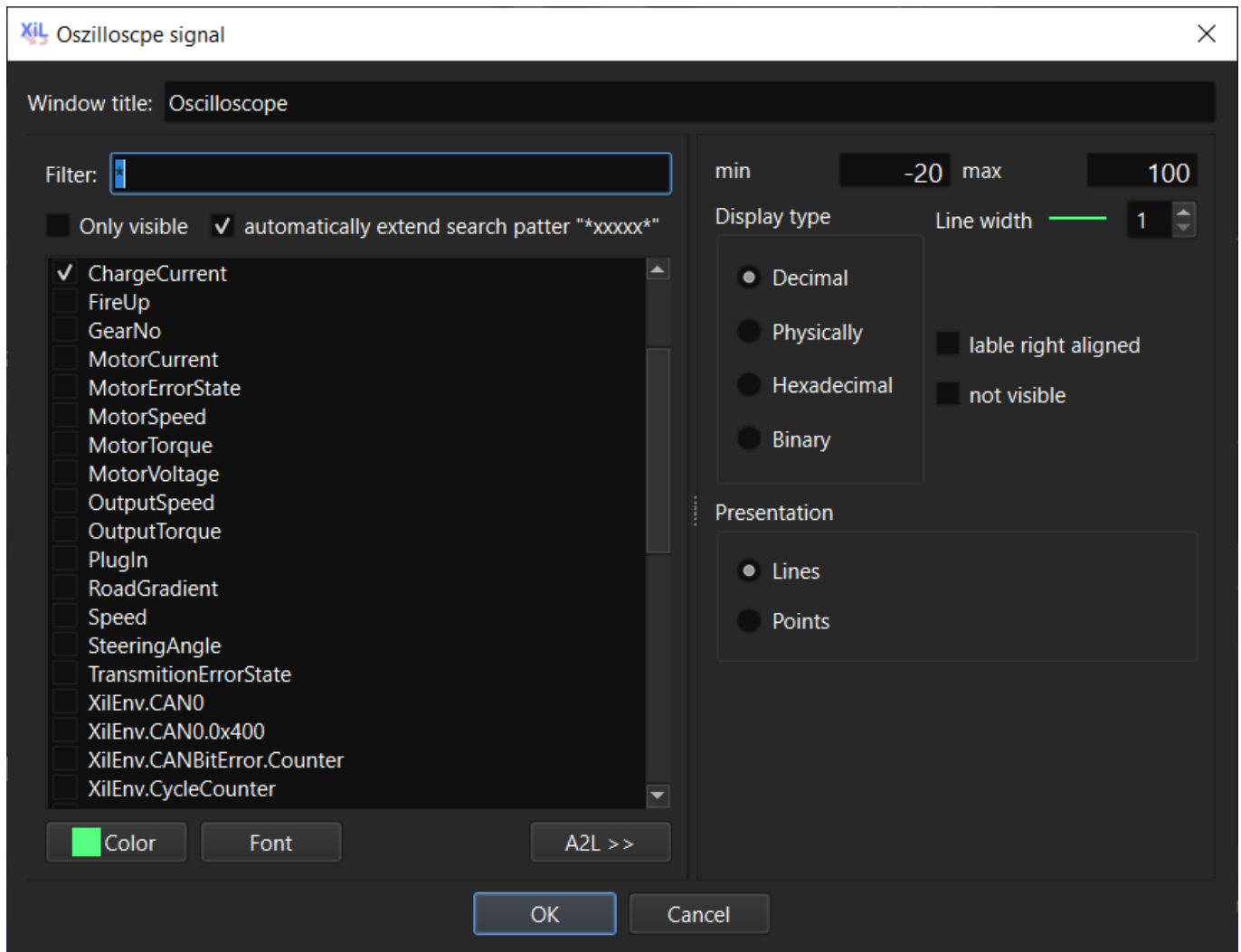
Window name	Window name the INI-file was saved as
Windowstepping	Step size in percent which the windows scrolls. When a graph touch the border of the window, it will be moved left by the chosen percentage.
Buffer depth	How many mesh points are saved in a ringbuffer (looking back to the past). min. 1024 and max. 1000000.
display left axis	show left axis
display right axis	show right axis
Trigger	possible triggering: variable > value that means rising slope or variable < value that means falling slope. If the variable is chosen as "@trigger not active" the trigger function is inactive. If "Single shot" is activated the internal buffer will be filled up with measured values after triggering and go to offline-mode automatically. By "pre trigger" the time period before triggering can be set.

6.3.2. Choice of variables

By a left-click on an axis description a dialogue box for choice of variables is opened.

Either typing in a variable name directly or a term with wildcard (*.?). By the use of the filter-key all applied variables are listed and can be selected. When a new variable was choosen the settings of color, minimum and maximum are applied from the blackboard but they can be changed afterwards. The choice of notation

(decimal or physical) is set to decimal by default. Physical notification is only possible when a conversion was entered for this variable within the blackboard before.



6.3.3. Deactivating chosen variables

Already chosen variables can be deactivated temporary by clicking on the description of the axis while holding the Shift-key.

In addition to that all variables can be affected simultaneously:

- "Ctrl" activates the variable that was clicked on and deactivates all others.
- "Shift-Ctrl" reactivates all.

This process is not only valid for the current OpenXilEnv-session, it is also saved in the INI-file. This means that all variables are in their old state automatically when starting OpenXilEnv the next time.

6.3.4. Zoom

To display an area zoomed, press the Ctrl-key and mark the upper left corner as well as lower right corner by left mouse button. Then release the Ctrl-key. Now choose the part to zoom: Y-axis, time-axis, both axes or no zoom.

OpenXilEnv memorizes the last 10 zoom levels. Run through these 10 zoom levels by using "Zoom in" or "Zoom out". The "Zoom History" displays the last 10 zoom levels textual.

6.3.5. Scroll

Scrolling in Y-direction is possible by pulling a Y-axis up- or downwards while holding the left mouse button at any time. Scrolling of the time-axis is only "offline" possible by pulling the time-axis to the left or right while holding the left mouse button.

6.3.6. Print

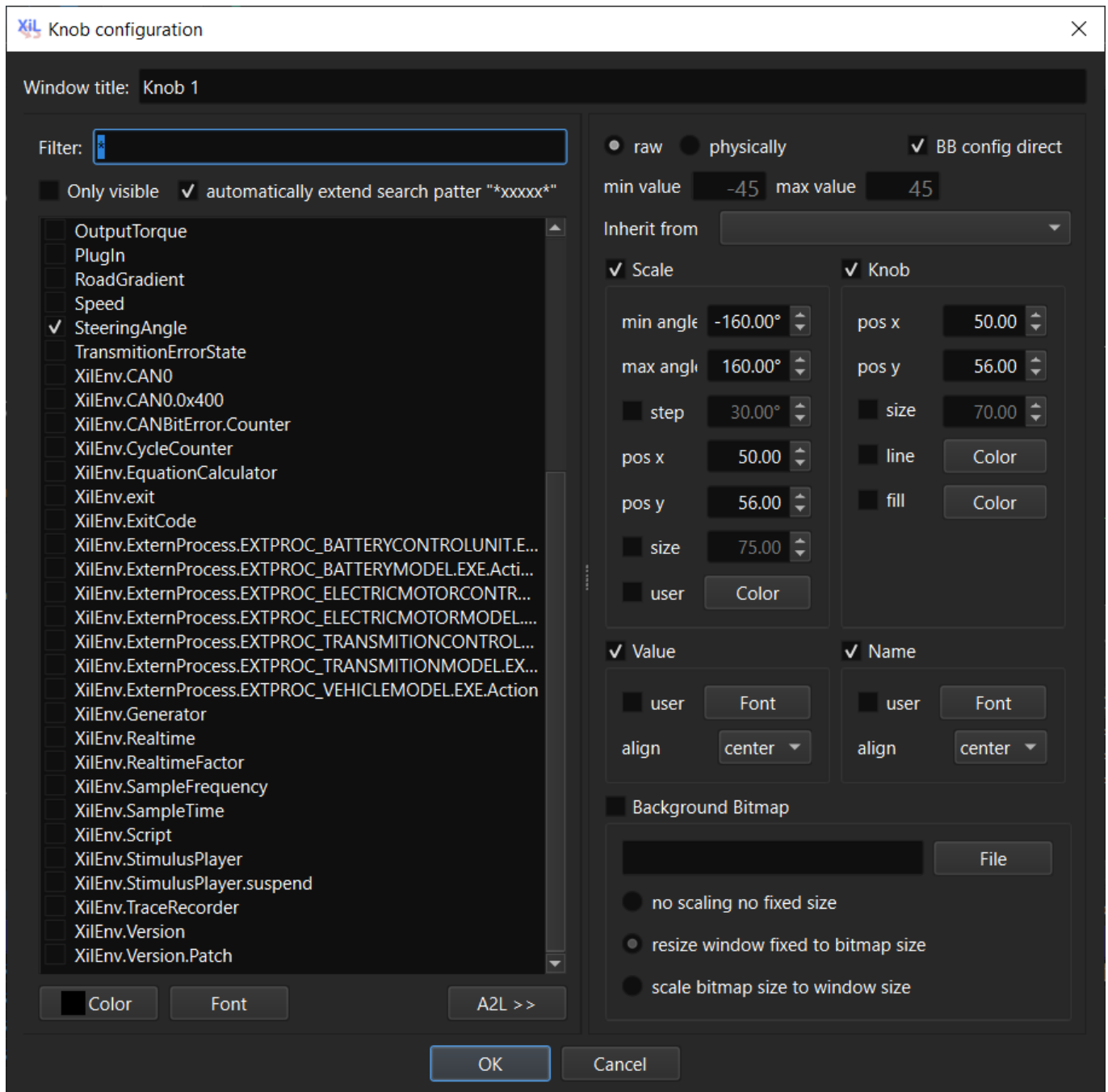
OpenXilEnv has no direct printing function. But it has an interface to the clipboard which can be used for printing through another application (Word, Power Point, ...).

1. right-click into oscilloscope image
2. place image into clipboard by using the menu item "Metafile Clipboard"
3. open/switch to Word or Power Point
4. load metafile from clipboard by key combination "SHIFT-Insert"
5. rework and print

6.4. Tachometer

Right-click (right mouse button), double-click (left mouse button) or **Display --> Config** opens the configuration-dialogue for the active tachometer:

The dialogue shows the selection of variable that should be displayed in the tachometer. Additional to that selection the minimum and maximum values of the variable must be specified. Tachometers can have several styles which can be applied by other tachometers or adjusted with the "Edit"-button:



„**Min angle**” / „**max angle**”: Rotation-area of the tachometer needle in degree

„**Scala**”: when enabled displaying the default scale as in the leftmost image.

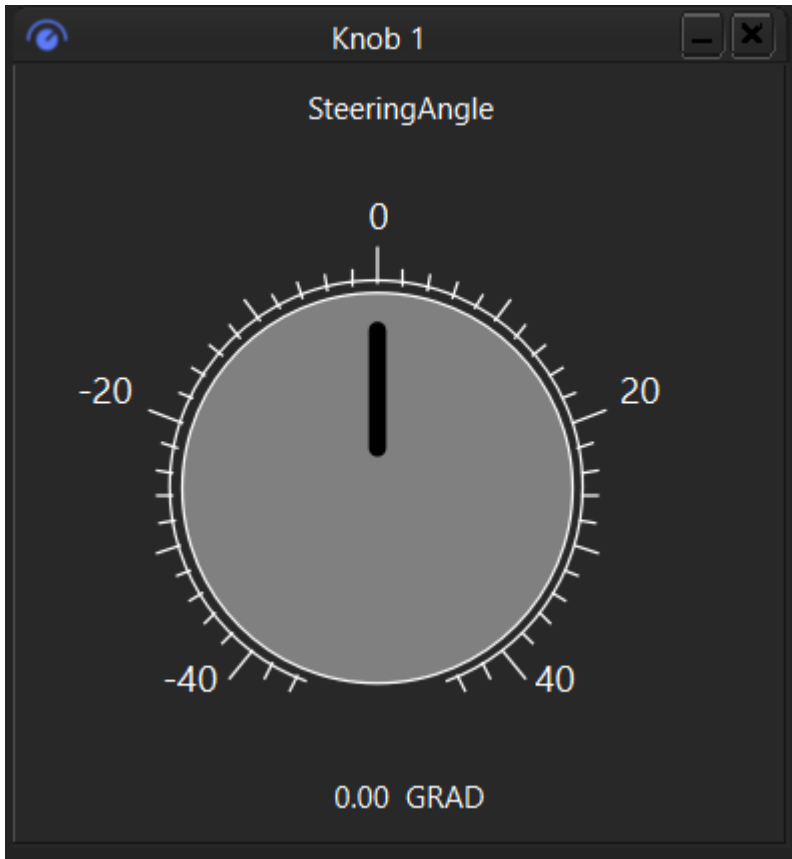
„**Needle**”: When enabled a red needle is used with the defined center and length in percent.

PosX=0%/PosY=0% means top left, PosX=100%/PosY=100% means bottom right

„**Text**”: When enabled a textual output is displayed at the center. PosX=0%/PosY=0% means top left, PosX=100%/PosY=100% means bottom right

„**Background bitmap**”: When enabled a background-file is loaded (see tachometer on left and middle). A tachometer with activated Bitmap has a fixed, non-changeable dimension that corresponds to the Bitmap.

6.5. Knob

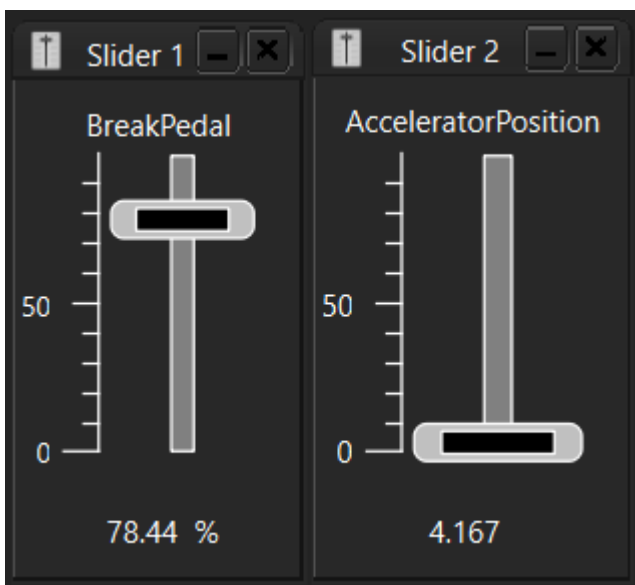


A knob is a tachometer that can be "twisted" with the mouse.

Therefore press and hold down the left mouse button within the knob-window. To get a finer resolution of the angle, move the cursor out of the window while holding the left mouse button.

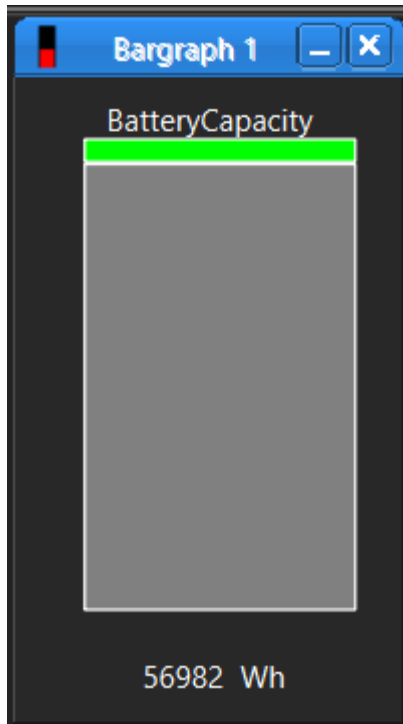
The configuration of the knob is equal to the configuration of the tachometer

6.6. Slider (Slide control)



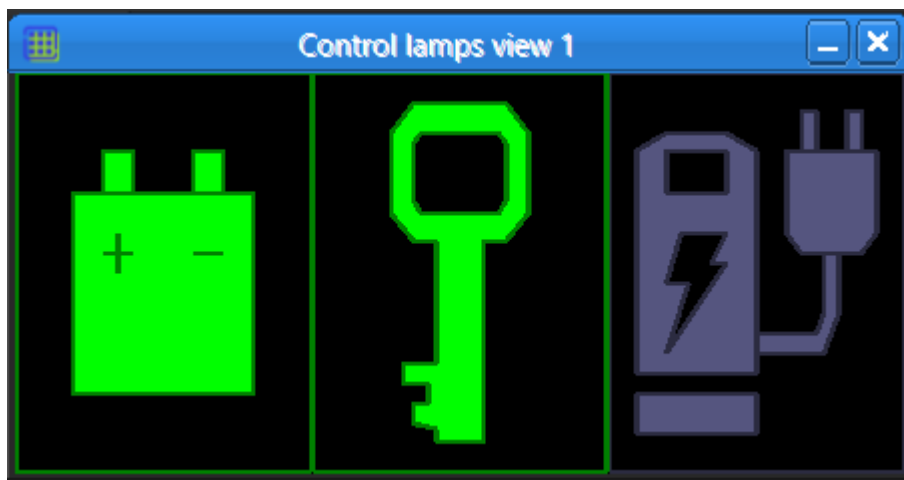
Right-click, double-click (left mouse button) or **Display --> Config** opens the configuration-dialogue for the active slider. The dialogue looks the same as the dialogue of the oscilloscope.

6.7. Bargraph



Right-click, double-click (left mouse button) or **Display --> Config** opens the configuration-dialogue for the active slider. The dialogue looks the same as the dialogue of the oscilloscope.

6.8. Control lamps



Control lamps are elements for coloured display of states (example indicator). For each „control lamp“-window there can be several lamps placed. The lamps are formed and coloured according to the value of the variable. The use of "text replace" is a condition for the conversion of the variable as well as defined colour value for each Enum.

6.8.1. Control lamps configuration-dialogue

The screenshot shows the 'Select control lamps' dialog box with the following components and annotations:

- 1**: Window name input field containing 'Blinker'.
- 2**: Dimensions section with 'Columns' set to 2 and 'Rows' set to 1.
- 3**: Background color selection area, currently showing black with a 'Change color' button.
- 4**: Selected sub window dropdown menu showing 'signal1'.
- 5**: Select Variable list box containing a list of variables, with 'signal1' selected.
- 6**: Position section with 'Pos column' (0), 'Pos row' (0), 'Dim column' (1), and 'Dim row' (1).
- 7**: Stencil list box containing predefined shapes like 'predef oil can', 'predef plus', 'predef minus', 'predef key 1', 'predef key 2', 'predef cycle', 'predef arrow right', and 'predef arrow left'. The 'predef arrow left' is selected, and its preview is shown as a red arrow pointing left.

Buttons at the bottom include 'Add', 'Delete', 'Change', and 'Close'.

1. Window name
2. Dimension as grid where single lamps can be positioned.
3. Background colour
4. Choice of the lamp
5. Assigned variable name of the choosen lamp
6. Position and size of the lamps within the grid
7. Choice of lamp-form. There are predefined forms but it is also possible to define a new one with the "Add"-button. Syntax: The outlines are described by a list of points. A list starts up with a bracket, followed by a plus or minus. Plus means a polygon filled with the foreground colour. Minus means a polygon filled with the background colour. Plus or minus is followed by the corners of the polygon $y_0/x_0;y_1/x_1;x_2/x_2;$. The value range of x and y is 0.0 to 1.0. The definition of the polygon is concluded by an ending-bracket. The polygons are handled from left/top to right/bottom.

Example:

```
(+0.2/0.2;0.8/0.5;0.2/0.8;)(-0.4/0.4;0.6/0.5;0.4/0.6;)
```

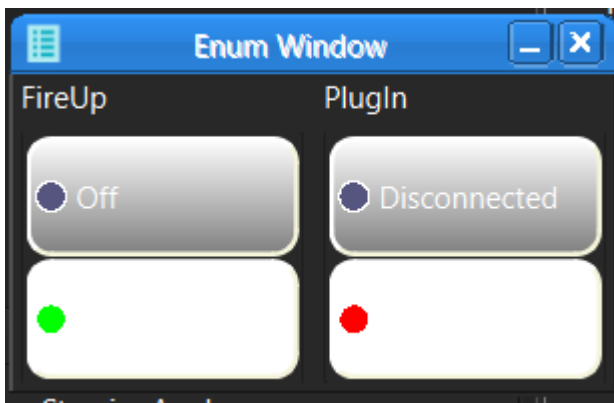


6.9. Enums

An enum-window contains the display of one or more status-variables. Only variables with conversion type "text replace" can be displayed.

Example: conversion of EnumVar:

```
0 0 \"RGB(0xC0:0xC0:0xC0)Off\";1 1 \"RGB(0xFF:0x00:0x00)On\";2 \*
\"RGB(0xFF:0xFF:0x00)undef\"; Conversion of pos:0 0 \"P\";1 1 \"R\";2 2 \"N\";3 3
\"D\";
```



7. Calibration tree

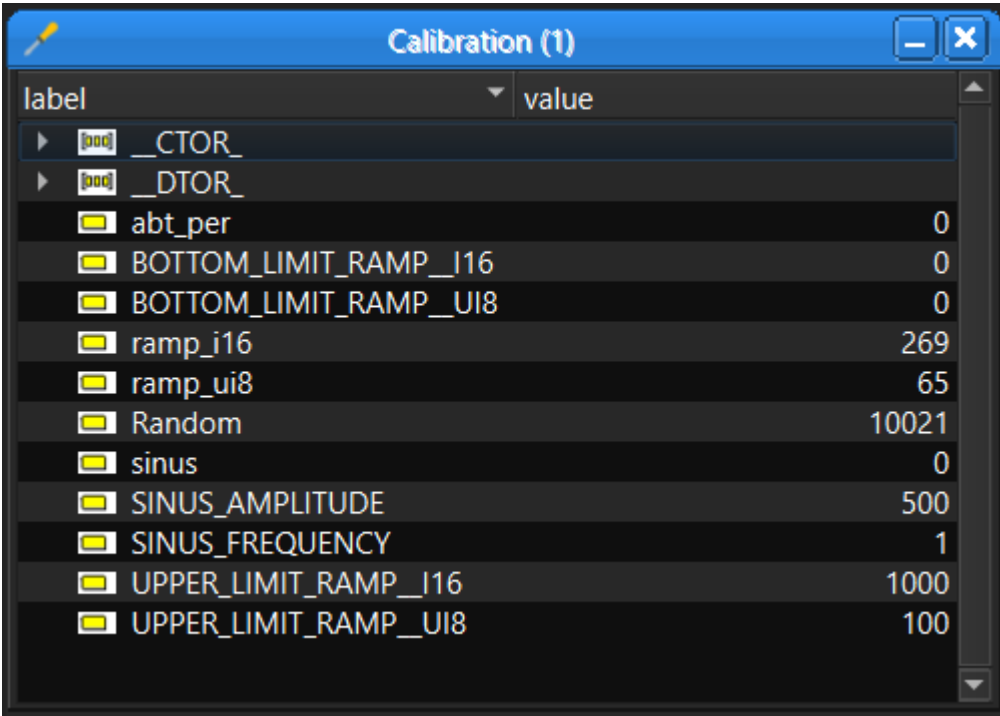
7.1. General

The calibration tree is the possibility under OpenXilEnv to change values and data.







There is differenced between the possibility of influence single values (also structures, fields etc.) and complete value ranges.

Important note: To use the application functionality, the external process that is relevant for the application has to be compiled with all debug-information and WITHOUT incremental linker (from BC 5b02)!

7.2. Data display



The symbols have the following meaning:

Symbol	Meaning
	Basic data type (char, short, long, float, double, ...)
	Pointer
	Array
	Structure, union or class
	Basic class
	Unknown data type (Function pointer and bit error are not

By double clicking with the left mouse button or pressing the insert-key, a variable or structure-/array-/pointer-element can be referenced or modified (applied) later on.

By using the right mouse button or **Display --> Config** the applied process and a label mask can be choosen. (*)-Wildcards are allowed within the mask. E.g.: KL_* displays all charateristic lines.

8. Processes

8.1. General information about processes

All OpenXilEnv-functions are implemented into two groups of processes:

- internal processes, which are the same over all projects (main functionality of OpenXilEnv) and
- external processes for project specific functions (model, ECU-SW)ECU
- maximum of 31 processes which are a number of 10 internal processes.So there is a maximum of 21 external processes possible.

8.2. Internal processes

Process name	Description
SignalGeneratorCompiler	This process compiles generator-files
SignalGeneratorCalculator	This process executes generator files
EquationCompiler	This process compiles formula-files
EquationCalculator	calculates formula-files
StimuliPlayer	This process can stimulate signal by a measurment file (*.DAT, *.MDF)
StimuliQueue	This is required by StimuliPlayer
TraceRecorder	This is needed for recording measurments
TraceQueue	This is required by TraceRecorder and Oscilloscope
Script	THis is the buildin script language interpreter
Oscilloscope	This is needed for oscilloscope windows
TimeProcess	This will provide time signals inside blackboard: "OpenXilEnv.Time.*"
CANServer	This will provide a virtual/real (remote master) CAN connection. The server will receive/transmit messages to virtual or real (remote master) CAN busses. Signals inside the messages will be read/write from/to the blackboard.
CCP_Control	It will provide a CCP master to connect an ECU over virtual CAN or real CAN (remote master). This process will be started automatically if needed.
XCP_Control	It will provide a XCP master to connect an ECU over virtual CAN or real CAN (remote master). This process will be started automatically if needed.
RemoteMasterControl	This process is needed for a remote master connection. This process will be started automatically if needed.
RPC_Control	This process is needed for Remote Procedure Call connection. This process will be started automatically if needed.
XCP_over_eth	It will provide a XCP over ethernet connecion to an ECU. It is only needed if you doen't want to use the driver inside the external process library.

8.2.1. Signal generator

The signal generator uncrosses sample points (value/time-pairs).

Hand-created curves that can be described through sample points (value/time-pairs), e.g.: when a variable should increase from 20 to 100 within 5 seconds and afterwards decrease to 50 within 10 seconds.

The description of the specifications occur in a so called generator-file (*.gen).

Syntax:

A generator file consists of a trigger-condition and one or more ramp definitions. The trigger-condition is introduced by the key word TRIGGER. It is followed by the observed variable, than condition and level. Possible conditions are > or <. The triggering occurs as an oscilloscope. So the level must be at least once overrun and once underrun to start the triggering (Condition:>).

A triggering starts all ramps at same time. If no trigger condition is specified (line TRIGGER ... missing), all ramps are started directly.

Example:

```
TRIGGER Gen_Start > 0.5
```

A ramp definition is always introduced by the keyword "RAMPE" followed by the variable which should be stimulated. Then the sample points (value/time-pairs), separated through whitespaces, tabs or return are following. The first value/time-pair is the first sample point, so all following value/time-pairs must have a higher or the same time value. If the gap between two consecutive value/time-pairs is equal or lower than the set time slice, it is incremented to the next higher one.

Example:

```
RAMPE Signal 800/0 2000/2.3 4000/2.7
4000/3 800/3.5 800/10
```

A value/time-pair can be created variable within certain thresholds. A variable sample point has to be specified by brackets for value-respectively time-information. Within these brackets there can be a calculation that includes constances (+-[0-9]e+-[0-9]), all variables and the following operators: +, -, *, /, &, |, >, <, >=, <=, ==, !=, &&, ||, !, &, |, ^, >>, <<, (,), sin(), cos(), tan(), asin(), acos(), atan(), exp(), pow(), sqrt(), log(), log10(), and(in,bits), or(in,bits), xor(in,bits), invert(in), getbits(in,bitoffset,bitsize), setbits(in,bitoffset,bitsize,bits), andbits(in,bitoffset,bitsize,bits), orbits(in,bitoffset,bitsize,bits), xoebits(in,bitoffset,bitsize,bits), round(v), round_up(v), round_down(v), ...

The evaluation of variable sample points occurs one sample point earlier and is only calculated once. If there are any variables within the equation that are changed there are no consequences on the right passed ramp.

Example:

```
RAMPE Signal (Signal)/0
e 50/2.5
100/(2.5 + peek_time) ;for variable specification
50/(2.5 + 2 * peek_time) ;the brackets are required!!
50/10
```

If the variable that should be stimulated available at the first sample point, a soft trapping of a variable is possible (no jump for unknow start value).

Comments start with a semicolon (👉) and ends with the end of the line.

The processes "SignalGeneratorCompiler" and "SignalGeneratorCalculator" are the signal generator. The "SignalGeneratorCompiler"-process compiles generator-files (*.gen) and the "SignalGeneratorCalculator"-process executes the included ramps.

Generator-files can be used only when both processes are active.

Within the variable generator the status of the signal generator is shown: 0: inactive; 1: *.gen-file was loaded, waiting for trigger condition is true, 2nd generator runs.

8.2.2. Formula calculator

Calculates "simple" equations that are specified in a *.tri-file.

Syntax:

Per line in an equation is allowed which is a maximum of 65535 characters long and ends with a semicolon. Within the TRI-file comments are allowed. They start with a semicolon ; and end at the end of the line. But the semicolon MUST stand at the beginning of the line. The single equations are executed from bottom to top.

Example:

```
SignalB = SignalA * 1.2 + 100;
SignalC= SignalA * SignalB;
; This is a comment
```

An equation can include constants (+-[0-9]e+-[0-9]), all variables and the following operations:

+, -, *, /, &, |, >, <, >=, <=, ==, !=, &&, ||, !, &, |, ^, >>, <<, (,), sin(), ...

The processes "EquationCompiler" and "EquationCalculator" are building the signal generator. The "EquationCompiler"-process compiles formula-files (*.tri) and the "EquationCalculator"-process calculates the included formulas once per cycle.

Only when both processes are active formula-files can be used.

Within the variable "Trigger" the status of the formula calculator is displayed: 0 inactive; 1 active

Note: Within an equation it will be always calculated with the data type Double, a conversion into the corresponding data type is not done until the values were written into the blackboard. Thereby the value is limited to the maximum and minimum thresholds of the corresponding data type ([no]{underline} overflow).

e.g.: Blackboard-variable a has the data type BB_UBYTE and b has the data type BB_UWORD

```
b = 200;
a = b + 100;

result: a = 255 and b = 200
```

```
b = a = 500;

result: a =255 and b = 500

a = 500;
b = a;

result: a =255 and b = 255
```

8.2.3. Stimuli-Player

The player can be used to bring in measuring data. It expects the OpenXilEnv ASCII-format (also called TE-format) or MDF with version 3.3 format. The player is initiated through a configuration-file. This can be created with an editor or the trace configuration-dialogue (**OpenXilEnv Control Panel --> Internal Process Control(Stimuli) --> Config...**) .

Example of a player configuration file

```
; Config-File for HD-Player
HDPLAYER_CONFIG_FILE

; Trace-File
HDPLAYER_FILE filename.dat

; Start condition
TRIGGER Gen_Start > 0.5

; which variables should be recorded
STARTVARLIST
VARIABLE Signal1
VARIABLE Signal2
VARIABLE Signal3
ENDVARLIST
```

These stand for:

Name	Meaning
HDPLAYER_CONFIG_FILE	Key word whereby the player config-file must
HDPLAYER_FILE	Name of the stimuli-file (drive- and path specifications are allowed)

Name	Meaning
TRIGGER	Specifies the start condition of the recording. The condition consists of the variable name und the referece value. The start condition is true, when start value was falling below the reference value and then exeeded the reference value at least once.
STARTVARLIST	Key word for the beginning of the list of variables that should be simulated. If the list is missing all singals included in the stimuli-file are used.
VARIABLE	Specification of a variable that should be recorded has to stand between the key words STARTVARLIST and ENDVARLIST.
ENDVARLIST	Key word for the end of the list of variables.

The state of the stimuli-player is displayed within the variable "HD_Play":

- 0 inactive
- 1 *.cfg-file was loaded, waiting for trigger condition to be true
- 2 Player runs

8.2.4. Trace-Recorder

The recorder can be used for recording, for offline evaluation by courtesy of MCS or for stimulating at a later time. The following 2 formates are supported: MCS-Rec-format and OpenXilEnv ASCII-format . The recorder is initialized through a configuration file. This can be created with an editor or through the trace configuration-dialogue (**OpenXilEnv Control Panel --> Internal Process Control(Trace) --> Config...**) .

Example of a recorder-configuration file

```
; Config-File for HD-Recorder
HdRECORDER_CONFIG_FILE

; Trace-file
HdRECORDER_FILE Recording.dat

; Recording rate in samples
HdRECORDER_SAMPLERATE = 1

; Generate MCS record files
DAT_FORMAT
DESCRIPTION_FILE Desc.a2l
AUTOGEN_DESCRIPTION_FILE

; Length of the recording in samples
```

```
HDRECORDER_SAMPLELENGTH = 10000000

; Start condition
TRIGGER Gen_Start > 0.5

; which variable should be recorded
STARIVARLIST
VARIABLE Signal1
VARIABLE Signal2
VARIABLE Signal3
; additional all variables of the process TEST.EXE
ALL_LABEL_OF_PROCESS TEST.EXE
ENDVARLIST
```

These stand for:

Name	Meaning
HDRECORDER_CONFIG_FILE	Key word the recorder config file has to start with
HDRECORDER_FILE	Name of the trace file (drive- and path specifications are allowed)
HDRECORDER_SAMPLERATE	Step size of the recording rate as an even factor of the system sample time 10ms)
MCSREC_FORMAT	When key word is defined it will be recorded in the MCS-record-format. A corresponding ROB-file is required which contains all variables of the recording (or more); hereby addresses are no issue. The ROB-file must always be named OpenXiLEnv.ROB. From version 4.004 the possibility of generating a ROB-file from OpenXiLEnv exists. At the point of creation, old referenced variables are written into the ROB-file as PROVARI2-items. In this proecess the following settings are assumed: label, text replacement or conversion formula, unit and decimal places. The address details are set to \$0 for all variables. Attention: The ROB-file is only useful for an offline-evaluation !!!
ETASASCII_FORMAT	When the key word is defined, it will be recorded in the ETAS-ASCII-format.
MDF_FORMAT	When the key word is defined, it will be recorded in the MDF-format. MDF-files also contain the conversion forms of the signals, so a physical display is also possible. Attention: If the MDF-file should be evaluated later by the ETAS measurement-analyzer, it must be regared that only linear conversions are standing in the blackboard (e.g. 2*#+ 100).
OpenXiLEnvSTIMULI_FORMAT	When the key word is defined, it will be recorded in the OpenXiLEnv-stimuli-format. This is the standard-format and it will also be used when no format is specified.

Name	Meaning
PHYSICAL	When this key word is defined, all variables are recorded that hold a physical conversion.
DESCRIPTION_FILE	Just in conjunction with MCSREC_FORMAT. Nur in Verbindung mit MCSREC_FORMAT, spezifiziert dtn Dateinamen der Beschreibungsdatei auf die REC-Datei referenziert. Diese muß dann im Gredi geladen sein. The specification is optionally, if nothing is specified, it will be referenced to the description file OpenXiLEnv.ROB.
AUTOGEN_DESCRIPTION_FILE	When this key word is defined, the description file, defined be the DESCRIPTION_FILE will be rewritten for each start of the recorder. If the filename ends with *.rob a ROB-file is written and if it ends with *.a21 an ASAP-file is written.
HDRECORDER_SAMPLELENGTH	Number of samples that should be recorded. An interrupt is possible at any time via user interface or script files.
TRIGGER	Specifies the start condition of the recording. The condition consists of a variable name and a referenced value. The start condition is complied when the variable value has fall below and than exceed the referenced value at least once.
STARTVARLIST	Key word for the beginning of the list for all variables to be recorded.
VARIABLE	Specifiation of a recorded variable has to stand between the key words STARTVARLIST and ENDVARLIST.
ALL_LABEL_OF_PROCESS	All variables of a particular process, this can occur several times between the key words STARTVARLIST and ENDVARLIST. Attention: this option is not available through the configuration-dialogue of the control-panel, but only by editing the *.CFG-file directly.
ENDVARLIST	Key word for the end of the variable list.

In the variable "HD_Rec" the state of the trace-recorder is displayed:

- 0 inactive
- 1 *.cfg-file was loaded, waiting for trigger condition to be true
- 2 recording runs

8.2.5. Script

The internal process "Script" provides the possiblity to control the OpenXiLEnv-user interface and other processes by remote!

8.2.6. TimeProcess

The TimeProcess allocates the current time information in the blackboard. OpenXilEnv uses the Windows-system time at start of the TimeProcess, afterwards the time runs synchronous to the simulated time. When the TimeProcess is started the system time will be used as start time, but afterwards it will use the simulated time and will not be synchronized with the system time. For the HIL option the simulated time is equal to the system time.

Variable name	Data type	Value range
OpenXilEnv.Time.Year	BB_UWORD	[1601 ...65535]
OpenXilEnv.Time.Month	BB_UBYTE	[1..12]
OpenXilEnv.Time.Day	BB_UBYTE	[1..31]
OpenXilEnv.Time.Hour	BB_UBYTE	[0..23]
OpenXilEnv.Time.Minute	BB_UBYTE	[0..59]
OpenXilEnv.Time.Second	BB_UBYTE	[0..59]
OpenXilEnv.Time.Minute	BB_UBYTE	[0..6] == [Sunday..Saturday]

8.3. External processes

The software to be tested must be tied together as so called external processes (some Windows-programs). The external processes are project-specific. Such a test project can be a gearbox-SW, a model, etc.

Hereby the following rule is valid:

Window name == process name == file name

To create an external process you will need a C/C++ development environment. OpenXilEnv supports Eclipse with gcc 4.5 and 4.8 or Visual Studio 2003, 2010, 2012, 2015 and 2017.

Following files are required:

ExtpLibVc7.lib for Visual Studio 2003 or ExtpLibVc7mt.lib if your project should be multithreaded.

ExtpLib_vc2010.lib for Visual Studio 2010

ExtpLib_vc2012.lib for Visual Studio 2012

ExtpLib_vc2012.lib for Visual Studio 2012

ExtpLib_vc2015.lib for Visual Studio 2015

ExtpLib_vc2017.lib for Visual Studio 2017

libextplibgcc.a for Eclipse with gcc as compiler

The Header XilEnvExtProc.h there is defined the whole interface to OpenXilEnv.

and optional extp.rc for a nice icon (you can use or not)

The best way to set up a new project is to copy on of the sample project out of the Samples\ExternProcesses directory and adapt it to your need.

The following command line arguments are possible:

`<extproc>.exe [-qn|wn] [-cn[:m]] [[OpenXilEnv-EXE with path[OpenXilEnv parameter]]`

-qn: The process tries to connect to a already running OpenXilEnv. If it was not successful, OpenXilEnv is started without demanding (quiet). Therefore the executeable file "OpenXilEnv-EXE with path" is called and all following arguments are passed with.

-wn: The process tries to connect to a already running OpenXilEnv within n seconds. If it was not successful, it is demanded if OpenXilEnv should be started. Therefore the executeable file "OpenXilEnv-EXE with path" is called and all following arguments are passed with.

-cn:m: External process is only called each n cycles with m cycles delay.

Example:

```
hn1.exe -q1 -c5:2 (hn1.exe tries to connect to SC within 1 second and is only
called each 5 SC-cycles with a delay of 2 SC-cycles)
```

8.3.1. Visual-C++ .Net

A new Visual C++ .Net project can be set up as follows. Alternatively the example-project extpt_vc7.vcproj from OpenXilEnv-delivery directory can be used.

Create new project

Todo

8.3.2. Interface

All interfaces to OpenXilEnv are defined in the header XilEnvExtProc.h and XilEnvExtProcCan.h.

XilEnvExtProcCan.h is only required when the virtual CAN is used by OpenXilEnv. The following functions must be provided in the test-SW:

reference_varis

```
void reference_varis (void)
```

All referencing should be made here (publish the addresses of variables). For referencing the following macros are defined in XilEnvExtProc.h, thereby XXX stands for the particular datatype: BYTE, UBYTE, WORD, UWORD, DWORD, UDWORD, QWORD, UQWORD, FLOAT, DOUBLE, SI8, UI8, SI16, UI16, SI32, UI32:

REFERENCE_XXX_VAR (ptr, name)

Ptr: Symbol address is built from

Name: Name in the blackboard as string

When name and symbol should be identical (most common case) the following macro is useful:

REF_XXX_VAR(name)

Name: Symbol and name in the blackboard

REFERENCE_XXX_VAR_UNIT (ptr, name, unit)

Ptr: Symbol where address is built from

Name: Name in the blackboard as string

Unit: Unit in the blackboard as string

REF_XXX_VAR_AI(name, unit, convtype, conversion, pmin, pmax, width, prec, rgb_color, steptype, step)

Name: Name in the blackboard as string

Unit: Unit in the blackboard ass string

Convtype: defines the conversion type of the display for this variable see table:

Conversion: Conversion string contains a conversion formula, an enum-definition or an empty string depending on convtype see table:

Define	Value	Description
CONVTYPE_NOTHING	0	No conversion defined; conversion = „"
CONVTYPE_EQUATION	1	A formula as conversion; conversion = „#*10.0"
CONVTYPE_TEXTREPLACE	2	An Enum; conversion = "0 0 "off\"; 1 1 \"on\";P
CONVTYPE_UNDEFINED	255	The conversion stays unmodified like it is listed in the INI-file

Pmin: Sets the maximum value range of the variable. For FP_NAN as transfer argument the maximum value range stays unmodified as it is in the INI-file.

Pmax: Sets the minimum value range of the variable. For FP_NAN as transfer argument the minimum value range stays unmodified as it is in the INI-file.

Width, Prec: Set the form of display. Width are the maximum displayed characters, Prec sets the displayed decimal places (Example: width = 10, prec = 3, output: 120.921). For WIDTH_UNDEFINED and PREC_UNDEFINED UNDEFINED as transfer arguments the form of display stays unmodified as it is in the INI-file.

Rgb_color: Assigns a default-color to the variable. There is the macro SC_RGB(r,g,b) available herefore (r=0...255 sets the red, g=0...255 sets the green and b=0...255 sets the blue component of the color). For COLOR_UNDEFINED as transfer argument the default color stays unmodified as it is in the INI-file.

Steptype, step: Sets increment/decrement of a variable. If steptype=0 a linear increment/decrement ($x=x+step$) is used, if steptype=1 a percentage increment/decrement ($x=x*step$) is used. For eTEPS_UNDEFINED as transfer argument the settings stay unmodified as it is in the INI-file.

REF_DIR_XXX_VAR_AI(dir,name, unit, convtype, conversion, pmin, pmax, width, prec, rgb_color, steptype, step)

Dir: There are three possibilitys

Possibility	Description
READ_ONLY_REFERENCE	The variable will only copied fromblackboard to the extern process before the cyclic method, but not back
WRITE_ONLY_REFERENCE	The variable will not copied from before the cyclic method but copiedafter it to the blackboard.
READ_WRITE_REFERENCE	The variable will copied fromblackboard and after the cyclicmethod back.

Name: Name in the blackboard as string

Unit: Unit in the blackboard ass string

Convtype: defines the conversion type of the display for this variable see table:

Conversion: Conversion string contains a conversion formula, an enum-definition or an empty string depending on convtype see table:

Define	Value	Description
CONVTYPE_NOTHING	0	No conversion defined; conversion = „"
CONVTYPE_EQUATION	1	A formula as conversion; conversion = „#*10.0"
CONVTYPE_TEXTREPLACE	2	An Enum; conversion = "0 0 "off\"; 1 1 \"on\";P
CONVTYPE_UNDEFINED	255	The conversion stays unmodified like it is listed in the INI-file

Pmin: Sets the maximum value range of the variable. For FP_NAN as transfer argument the maximum value range stays unmodified as it is in the INI-file.

Pmax: Sets the minimum value range of the variable. For FP_NAN as transfer argument the minimum value range stays unmodified as it is in the INI-file.

Width, Prec: Set the form of display. Width are the maximum displayed characters, Prec sets the displayed decimal places (Example: width = 10, prec = 3, output: 120.921). For WIDTH_UNDEFINED and PREC_UNDEFINED UNDEFINED as transfer arguments the form of display stays unmodified as it is in the INI-file.

Rgb_color: Assigns a default-color to the variable. There is the macro SC_RGB(r,g,b) available herefore (r=0...255 sets the red, g=0...255 sets the green and b=0...255 sets the blue component of the color). For COLOR_UNDEFINED as transfer argument the default color stays unmodified as it is in the INI-file.

Steptype, step: Sets increment/decrement of a variable. If steptype=0 a linear increment/decrement ($x=x+step$) is used, if steptype=1 a percentage increment/decrement ($x=x*step$) is used. For eTEPS_UNDEFINED as transfer argument the settings stay unmodified as it is in the INI-file.

For C++ projects there is the class cReferenceWrapper with the following static methods, that are overloaded for the datatypes named above:

```
void RefVar(XXX *arg_ptr_Var, char *arg_ptr_Name)
```

```
void RefVarUnit(XXX *arg_ptr_Var, char *arg_ptr_Name, char *arg_ptr_Unit)
```

```
void RefVarAllInfos(XXX *arg_ptr_Var, char *arg_ptr_Name, char *arg_ptr_Unit, int arg_Convtype, char *arg_ptr_Conversion, double arg_Min, double arg_Max, int arg_Width, int arg_Prec, unsigned long arg_RGBColor, int arg_StepType, double arg_step)
```

init_test_object

```
int init_test_object (void)
```

The initialization of the ECU should be carried out here. When no error occurs, return 0. If a value unequal 0 is returned, the process is shut down immediately.

cyclic_test_object

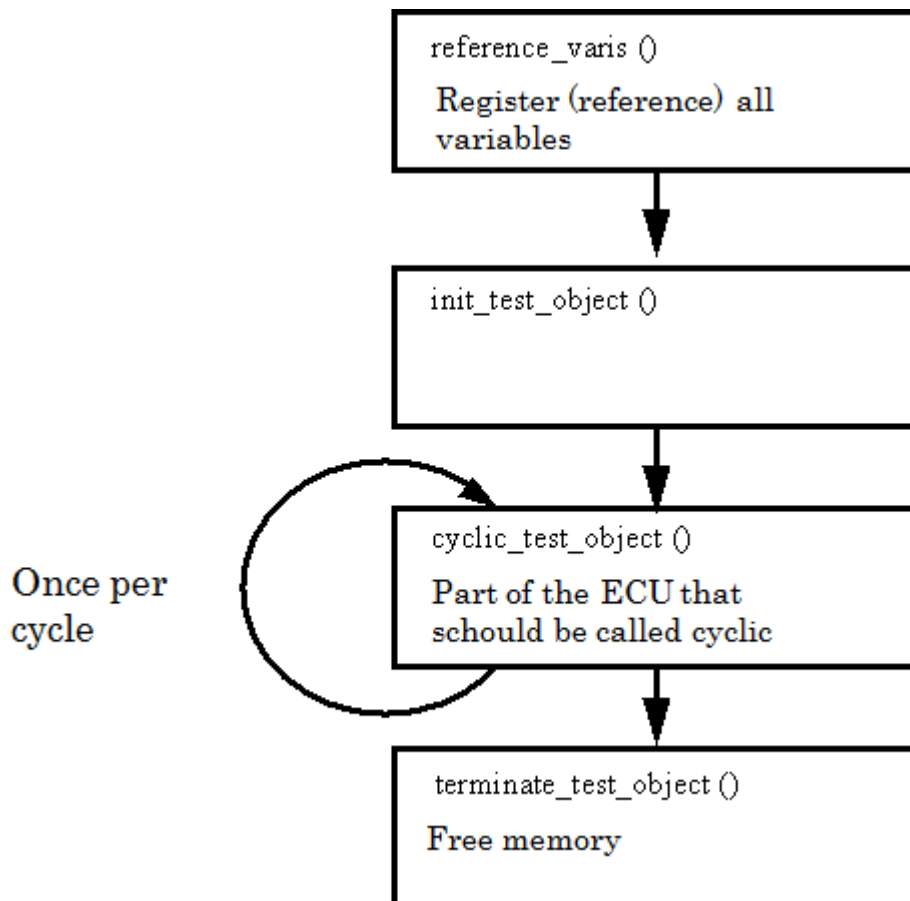
```
void cyclic_test_object (void)
```

This function is called cyclical, as long as the external process (ECU) runs.

terminate_test_object

```
void terminate_test_object (void)
```

This function is called when the external process is ending.



Examples

```

#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define XILENV_INTERFACE_TYPE XILENV_DLL_INTERFACE_TYPE
#include "XilEnvExtProc.h"
#include "XilEnvExtProcMain.c"

/* Global variable of the external process */
short ramp_i16;
unsigned char ramp_ui8;
short sinus;
short Random;
double abt_per;

const short BOTTOM_LIMIT_RAMP__I16 = 0;
const unsigned char BOTTOM_LIMIT_RAMP__UI8 = 0;
const short UPPER_LIMIT_RAMP__I16 = 1000;
const unsigned char UPPER_LIMIT_RAMP__UI8 = 100;

const double SINUS_AMPLITUDE = 500;
const double SINUS_FREQUENCY = 1;

/* This will be called first if the process is started */
void reference_varis (void)

```

```

{
    REFERENCE_WORD_VAR (ramp_i16, "ramp_i16");
    REFERENCE_WORD_VAR (sinus, "sinus");
    REFERENCE_WORD_VAR (Random, "Random");
    REFERENCE_UBYTE_VAR (ramp_ui8, "ramp_ui8");
    REFERENCE_DOUBLE_VAR (abt_per, "abt_per");
}

/* This function will be called next to reference_varis */
int init_test_object (void)
{
    return 0;    /* == 0 -> No error continue
                  != 0 -> Error do not continue */
}

/* This will call every simulated cycle */
void cyclic_test_object (void)
{
    static double sinus_time;
    if (ramp_i16++ > UPPER_LIMIT_RAMP__I16) ramp_i16 = BOTTOM_LIMIT_RAMP__I16;
    if (ramp_ui8++ > UPPER_LIMIT_RAMP__UI8) ramp_ui8 = BOTTOM_LIMIT_RAMP__UI8;
    sinus_time += 2.0 * 3.14 * abt_per * SINUS_FREQUENCY;
    Random = rand ();
}

/* This will be called if the external processs will be terminated */
void terminate_test_object (void)
{
}

```

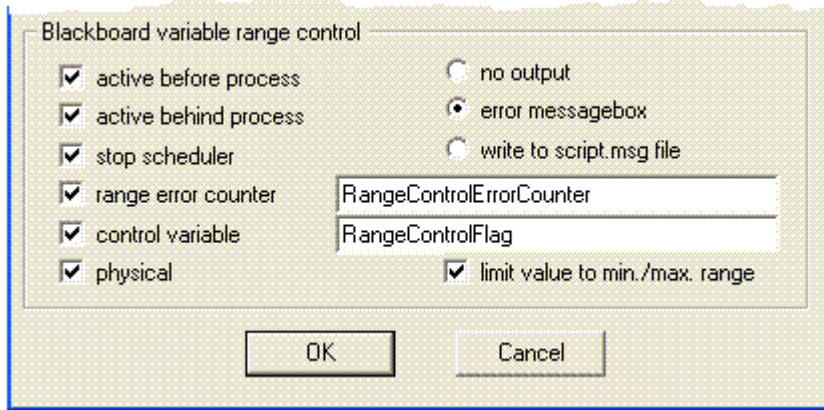
Each external process can get the OpenXilEnv command line handed over as parameter. Example:

```
-q10 path-to/XilEnvGui.exe --ini c:\XilEnv\my_ini.ini
```

That means that the external process tries to log in to OpenXilEnv for 10 seconds. If it is not successful OpenXilEnv will be started by the command line handed over. If OpenXilEnv is not able to login for further 60 seconds, an error message is display and OpenXilEnvs shuts down.

8.3.3. Value range control for variable

For each external process it is possible to control their variables, which are registered in the blackboard, on falling below/exceeding of min./max.-thresholds. The control has to be configured via process-info-dialogue of the OpenXilEnv-control panel. In the first step choose the process whose variables should be monitored. The options for value range control are summarized in the block „Blackboard variable range control“, below in the dialogue (see image). As min.-/max.- thresholds the min.-/max.- specifications from the blackboard are used for each variable.



active before process: Turns on the value range control before each cyclical call.

Active behind process: Turns on the value range control after each cyclical call.

Stop scheduler: Stops OpenXilEnv if a value range exceedance occurs.

Range error counter: Here could be a blackboard variable specified, which will be incremented for each value range exceedance. Datatype is UDWORD.

Control variable: Here could be a blackboard variable specified, whereby the value range control can be switched on or off (0 means off, 1 means on). Datatype is UDWORD.

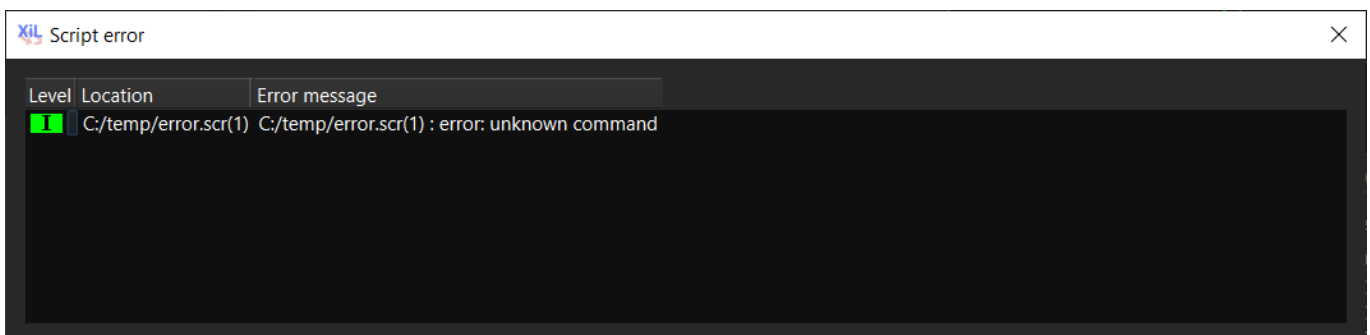
Physical: The value of the variable is converted through the blackboard-registered conversion before the comparison. If no conversion is specified, the raw value is compared.

$\text{Min} > \text{Phys}(\text{Variable}) < \text{Max}$

no output / error message / write to script.msg file: Hereby the type of error message can be set on to a value range exceedance:

no output: No error message is output.

error messagebox: An error message window is opened. If another window is open, an error message will be written into the existing window.



Write to script.msg file : The error message is written into the script.msg-file and into the script-message-window.

Limit value to min./max. range: In case of a value range exceedance the variable is set back to the min.-/max.- values.

9. Script language

9.1. Introduction

The script language serves as:

- automation of test sequences,
- repeatability and reproducibility of standardized test at any time,
- combining and connection of all OpenXilEnv provided options without a required presence of an user.

The script language is a purely interpreter language, which is compiled and executed while runtime. The commands are partially taken from the DOS-batch processing-commands.

Script files can be passed as command line argument or selected and started within the control-panel.

For the creation of the script file any desired text editor can be used. Recommended is the Ultraedit®©. For this editor there is a language file which includes all commands, so syntax-highlighting is also supported for the script language.

When a script file was started, OpenXilEnv does a system check within one cycle for the whole script file and generates the file '**script.err**' in the current directory. All syntactical mistakes found in the processed script file are written into this file.

While checking the syntax, the temporary-files (f.Recorder, Player...) are created on trial for the following execution of the script file.

A debug file '**script.dbg**' is written while the execution of a script file. Within this debug file all executed commands and problems occurred while the execution of OpenXilEnv are recorded (the logical mistakes within a program of script commands can be found in this way).

Only one script-command can be executed per OpenXilEnv-cycle (Exception: commands with curly bracket {} -- in this case everything in the brackets is handed to OpenXilEnv for execution within one cycle, as well as the use of the commands ATOMIC/END_ATOMIC do).

The temporary files generated by script language and all files named **script.***** are written into the -- valid at start time of the script -- current directory (This directory can be determined e.g. by clicking on to the button 'File...' in the OpenXilEnv-control panel).

9.2. Instruction set

9.2.1. Basic instructions[]

START_RAMPE (filename.gen)

With this command the ramp progression for variables is defined, which are included in the complete gen-file. It is also permitted to write the contents of such a gen-file right behind the command with curly brackets {}.

START_PLAYER needs to run the internal processes "SignalGeneratorCompiler" and "SignalGeneratorCalculator"

STOP_RAMPE

With this command the ramp progression for variables can be stopped before the ramps are finished

START_RECORDER / START_REC (filename.cfg)

With this instruction the recorder function of OpenXiLEnv is started. The variables that should be recorded and the trigger variable that starts the recorder is specified in the *.cfg-file of the recorder. It is also permitted to write the contents of such a cfg-file right begin the command with the curly brackets {}. If the recorder configuration should be used several times, a configuration file can be created by CREATE_REC_CFG().

STOP_RECORDER / STOP_REC

If the recording is prematurely terminated, e.g. a failure occurred in the tested program or a certain trigger condition is true or a ramp-file or stimuli-player is over. It can also be used when the recording time is not clear at the start of the script file.

Comment */ or ;

A comment in the script file is marked such as ANSI-C.

Within the curly brackets {} only the semicolon ';' is allowed as comment character!

START_PLAYER (filename.cfg)

Starts the execution of a stimuli-file, which is created e.g. in the vehicle or at the labcar. The *.cfg-file includes the real stimuli-file name, the trigger variable and a value, that must be dropped or exceeded by the trigger variable, so the player can be started indeed. It is also permitted to write the contents of such a cfg-file right behind the instruction with the curly brackets. A START_PLAYER should be followed by a WAIT_PLAYER necessarily, otherwise the script execution runs on after the START_PLAYER without waiting for the end of the player file.

START_PLAYER needs to run the internal processes "TraceQueue" and "TraceRecorder"

STOP_PLAYER

To stop the process "StimuliPlayer" prematurely, e.g. to start another player-/stimulifile or in case of a failure while the execution of the script!

START_TRIGGER / START_EQUATION

If you include the formula in the script with "{}" brackets you can also use /**/ and // comments there.

START_TRIGGER and START_EQUATION are completely equal!

There are the following applications for this instruction:

For the following instructions a START_TRIGGER deletes all previous equations.

1. START_TRIGGER(filename) / START_EQUATION(filename)
2. START_TRIGGER{include the contents from trigger-cfg-file directly} / START_EQUATION{the same}

With both following instructions several blocks of equations can be enabled or disabled independent of each other.

1. START_TRIGGER(blocknr,filename,behind/before, processname)
2. START_TRIGGER(blocknr, behind/before, processname) {content from cfg-file directly}

Starts a process that solve equations or can be used for data manipulation, e.g. $\text{signal1} = 0,5 * \text{signal2}$.

Blocknr all numbers from 1 to 99 are permitted. Can be used for allocating an ID to a block of equations, which can be also used to delete this block again. The number 0 is reserved for a block, that should be saved within the INI-file. The number -1 is reserved for deleting all active blocks. If a START_TRIGGER is called twice with the same block number, the equations will be added and the equations of both will be active. A afterward STOP_TRIGGER with this block number will delete all equations from both START_TRIGGER with the same block number.

Filename is the file that contains the equations. It is also permitted to specify the contents of such a tri-file right behind the instructions with curly brackets {}.

Behind/Before specifies if the equations should be carried out before or after the *Processname*.

Processname to specify which process is meant.

Attention:

After each START_TRIGGER-command **including only one** calling argument, all previous equations are deleted and replaced through the current START_TRIGGER without block number.

The behavior of the equations at the end of a script file can be different. if the START_TRIGGER is called with block number or not. If START_TRIGGER is called with block number the equation will still calculated after script is stopped. If START_TRIGGER is called without block number the equation will be stopped if the script file is stopped. This behavior can be changed with CHANGE_SETTINGS (STOP_EQUATION_IF_SCRIPT_STOPPED, no).

START_TRIGGER without block number needs to run the internal processes "EquationCompiler" and "EquationCalculator"

STOP_TRIGGER, STOP_EQUATION

The effects of these both instructions are totally equal!

After a STOP_TRIGGER-command without calling argument, the equations previously started with START_EUATION or START_TRIGGER without association to a process will be deleted.

To delete a specific block of equation, use to following syntax: STOP_TRIGGER(blocknr, before/behind, processname)

For meaning of the parameters see START_TRIGGER().

WAIT_RAMPE

Waits for the status 'Sleep' for processes "SignalGeneratorCompiler" and "SignalGeneratorCalculator". This can be used e.g. for the synchronization of several STARTxxx-instructions.

The execution of the scriptfile is interrupted while the started ramp-file is completely ended.

WAIT_PLAYER

Waits for the state 'HDPLAY' of the process "StimuliPlayer".

This can be used e.g. for synchronization of several STARTxxx-commands.

The execution of the scriptfile is stopped until the started player-file is completed

SET_BBVARI / SET (bbvari = euqation)

*Note: The BB stands for **B**lack**B**oard*

To assign a certain value to a variable with the name *bbvari* of the blackboards, which can be dependent e.g. from an *equation*. Additionally the buildin-function 'exist (variable name)' can be used, it returns 1 if the variable exists in the blackboard and otherwise it returns 0.

When a variable does not exist it will be automatically created. To switch off this mechanism, see ADD_BBVARI_AUTOMATIC(ON/OFF).

The treatment of text replacement (enum) is described in chapter "enum" at "buildin-functions".

WR_BBVARI_ENABLE (bbvari, pname)

*Note: The WR stands for **W**RITE.*

Is used to allow the process, named *pname* to write into the variable *bbvari* of the blackboard.

WR_BBVARI_DISABLE (bbvari, pname)

This instruction blocks the writing of a process into a blackboard variable.

E.g. very useful for the Stimuli-player to play measured data, which contains recorded programm internal respectively output-values.

ADD_BBVARI (variname, varitype, [Unit]optional)

Includes a new (process independent) variable, named *variname* from type *varitype* and optionally with unit *Unit* into the blackboard. This can be used e.g. to get a process independent variable just for display purposes.

Possible data types for *varitype* are: UBYTE, BYTE, UWORD, WORD, FLOAT, DWORD, UDWORD, DOUBLE. If no *varitype* is set, DOUBLE is used automatically! (U=unsigned, D=double). If the variable already exists with this name, but it is another type, a warning is displayed.

REMOVE_BBVARI (variname)

Deletes a variable from blackboard that was created by the instruction ADD_BBVARI. The variable will not be deleted until the number of REMOVE_BBVARI and ADD_BBVARI are equal and no other process accesses the variable.

ADD_BBVARI_AUTOMATIC (ON/OFF/REMOVE/STOP)

This instruction can be used to control the automatic creation of unknow variables.

If the option „ON" is specified (default setting from OpenXilEnv at start up), a OpenXilEnv-unknow variable is created automatically by ADD_BBVARI and used by SET_BBVARI-instruction. Data type is defined as COUPLE by default.

The feature can disturb automatical test scripts, because it could be unnoticed that an important control unit variable is not available for the tested variant/version but the test was concluded as "OK" falsely.

If the option "OFF" is chosen, the variables are not created automatically but the script is stopped with an error message when using an unknown variable.

If the option „REMOVE" is set, the script-interpreter releases the variable and it is deleted from the blackboard when no other process accesses to this variable.

If the option "STOP" is set, the script will be stopped if a variable doesn't exist.

RUN (scriptfile.scr), CALL (scriptfile.scr)

Both instructions have an absolute identical effect!

This instructions can call other subroutines, which means a script-file can call other script-files which can call other script-files again.

IF (equation)

This instruction is used for conditional execution of instruction sequences. If the expression is true (unequal to zero), all directly following instructions are executed until an ENDIF, ELSE or ELSEIF appears.

The function "exist (variable name)" verifies whether a variable exists in the blackboard (return value = 1) or not (return value = 0).

The function "get_rprocess_state (process name)" query the status of an external process, "get_process_state" returns the status of an external process as follows:

0-> process does not exist

1-> process has status reference (reference_varis-fct)

2-> process has status init (init_xxxx-fct)

3-> process has status running (cyclic_xxxx-fct)

4-> process has status terminate (terminate_xxxx-fct)

Example:


```
IF (get_process_state (EXTPT_32.EXE) == 0)

MESSAGE (Prozess EXTP_32.EXE does not run yet, start it)

START_PROCESS (%XILENV_SCRIPT_DIR%\EXTPT_32.EXE)

ENDIF
```

ELSE

This instruction opens the instruction sequence that should be executed when the IF-condition was not true.

ENDIF

This instruction concludes an IF-/ELSE-/ELSEIF-block.

ELSEIF

This instruction is equal to the instructions ELSE and IF in a row, except that one ENDIF lesser is needed.

DEL (filename)

Delete the file with the name *filename*.

CD (directory)

Change to current directory (like the DOS-command).

Example:

```
'CD (c:\\script')
```

MD (directory)

Create the specified directory (like the DOS-command).

RD (directory)

Remove the specified directory (like the DOS-command).

COPY (filename1, filename2)

Copy file *filename1* into file *filename2* . Neither wildcards nor placeholders are allowed within the file names, which means only one file can be copied at once. A file name can consist of 123 characters including their path.

The comma between both file names is mandatory!

EXIT_TO_DOS(errorlevel)

Complete abortion of a script file and return to DOS / WINDOWS. An error level can be returned as parameter

Important for calling OpenXiLEnv from a batch-file!

EXIT

To exit the processing of the script file, e.g. when a tested function is faulty. The displayed status of the Control-Panal is set to STOP. Trace-Recorder, Stimulti-Player, Formula-calculator and Signal-generator will be stopped.

STOP_OpenXiLEnv (not for HIL option !)

Stop the processing of the script file. This instruction was implemented to interrupt a script-file at any position and continue by user-input. It is a kind of **Breakpoint** in script files. Also see chapter **Script debugging**.

WHILE (equation)

Loop instruction, equal to other programming languages

Variables from the blackboard can be specified.

The function "exist (variable name)" verifies wether a variable exists in the blackboard (return value = 1) or not (return value = 0).

The function "get_rrocess_state (process name) query the status of an external process, "get_process_state" returns the status of an external process as follows:

0-> process does not exist

1-> process has status reference (reference_varis-fct)

2-> process has status init (init_xxxx-fct)

3-> process has status running (cyclic_xxxx-fct)

4-> process has status terminate (terminate_xxxx-fct)

Example:

```
START_PROCESS (%XILENV_SCRIPT_DIR%\EXTPT_32.EXE)

WHILE (get_process_state (EXTPT_32.EXE) != 3)

MESSAGE (Wait)

ENDWHILE
```

ENDWHILE

This instruction has to stand at the end of a WHILE-loop.

BREAK

Is used for leaving a while-loop prematurely.

WAIT_UNTIL (Condition, Timeout, [STOP/CONT/GOTO/GOSUB/CALL_PROC [...], Timeout-message, [message-parameter1, ...]])

The instruction WAIT_UNTIL can interrupt the execution of the script until the condition is true or the timeout passed. The timeout is specified in cycles. Additional to that the behaviour for a timeout can be defined optionally. It can be set by the 3rd parameter either as STOP: aborts the script, or CONT: continue the execution of the script. If no 3rd parameter is existing, the script is will not be stopped (CONT by default). Furthermore the keywords GOTO, GOSUB, CALL_PROC are allowed.

An additional last parameter can be displayed as message in case of a timeout. All following parameters comply with the %z/%x elements in the message text(equal to the MESSAGE-instruction).

Sampels:

```
WAIT_UNTIL (Signal \> 1000, 100)
```

This will wait until the statment "Signal > 1000" will be **not** zero or 100 cycles timeout have elapsed. Afterwards the script will be continue.

```
WAIT_UNTIL (Signal \> 100, 100, \"timeout reached\")
```

The same behaviour as above. But with the message "timeout reached".

```
WAIT_UNTIL (Signal \> 100, 100, STOP)
```

If the statement "Signal > 1000" will stay zero for 100 cycles the script will stopped.

```
WAIT_UNTIL (Signal \> 100, 100, GOTO, LABEL)
```

This will wait until the statment "Signal > 1000" will be **not** zero or 100 cycles timeout have elapsed. If the timeout reached the script will be jump to the given label like the GOTO command. If the timeout is not reached the script will be continue with the next command.

```
WAIT_UNTIL (Signal \> 100, 100, GOTO, LABEL, \"timeout reached\")
```

The same behaviour as above. But with the message "timeout reached".

```
WAIT_UNTIL (Signal \> 100, 100, GOSUB, LABEL)
```

This will wait until the statment "Signal > 1000" will be **not** zero or 100 cycles timeout have elapsed. If the timeout reached the script will be branch to the given label like the GOSUB command. With the RETURN Command it will be jumped back to the next command afterwards the WAIT_UNTIL command. If the timeout is not reached the script will be continue with the next command.

```
WAIT_UNTIL (Signal \> 100, 100, GOSUB, LABEL, \"timeout reached\")
```

The same behaviour as above. But with the message "timeout reached".

```
WAIT_UNTIL (Signal \> 100, 100, CALL_PROC, MyProc)
```

This will wait until the statment "Signal > 1000" will be **not** zero or 100 cycles timeout have elapsed. If the timeout reached the script will be call to the given procedure "MyProc" like the CALL_PROC command. After leaving the procedure the the script will be continue with next command afterwards the WAIT_UNTIL command. If the timeout is not reached the script will be continue with the next command.

```
WAIT_UNTIL (Signal \> 100, 100, CALL_PROC, MyProc, \"timeout reached\")
```

The same behaviour as above. But with the messge "timeout reached".

```
WAIT_UNTIL (Signal \> 100, 100, CALL_PROC, MyProc, 10, Signal + 5)
```

This will wait until the statment "Signal > 1000" will be **not** zero or 100 cycles timeout have elapsed. If the timeout reached the script will be call to the given procedure "MyProc" like the CALL_PROC command with two parameters "10" and "Signal + 5". After leaving the procedure the the script will be continue with next command afterwards the WAIT_UNTIL command. If the timeout is not reached the script will be continue with the next command.

```
WAIT_UNTIL (Signal \> 100, 100, CALL_PROC, MyProc, 10, Signal + 5, \"timeout reached\")
```

The same behaviour as above. But with the message "timeout reached".

MESSAGE / PRINT(text %z text %x text, variname1, variname2)

This instruction prints a message to the message window and into the script.msg file. The first parameter are the text which should be printed. All following parameters are optional and depends on the existence of one ore more format specifier inside the text of the first parameter. The script.msg file can be used as an automated documentation (output of results) of automated tests.

Following format specifier are possible:

%z: Output floating point value into exponential representation (same as %g)

%g: Output floating point value into exponential representation (same as %z)

%d Output as a signed integer value

%u Output as an unsigned integer value

%x Output as hexadecimal integer value

%s Output as texreplace if an enum is defined as conversion otherwise it will be printed same as %g

To print the % character use %%.

The '**script.msg**'-file exists not until the syntax check (creation of the file '**script.err**') was successful. The '**script.msg**'-file is deleted at each restart of the script file!!!

Also see the command VERBOSE(ON/OFF).

Within the text there are no commas allowed, because they are characteristics for the beginning of parameters!!!

DELAY (number_of_cycles)

To interruppt the processing of the script-file for *number_of_cycles*.

One cycle has the value that was set through the menu 'Settings'.

While the delay time all other process keep on running!!!

New from V4.203: instead of a fixed number_of_cycles there can be specified an equation or variable as well!

START_PROCESS (process name, SVL-file[opt])

Start the process called „*processname*“. When starting an external process, a SVL-file can be handed over optionally. This SVL-file is loaded right after the start of the external process but before the call of the init-function.

e.g.: 'START_PROCESS (ECU)' to execute the gear box software cyclic.

START_PROCESS_EX (process name, ...)

START_PROCESS_EX (process name, priority[opt], period[Opt], delay[opt], timeout[opt], SVL-file[opt], Blackboard prefix[opt], RangeControlBefore[opt], RangeControlBehind[opt], RangeCotrolStopScheduler[opt],

RangeCotrolOutput[opt], RangeCotrolErrorCounter[opt], RangeControlVariable[opt], RangeCotrolPysical[opt], RangeControlLimitValues[opt])

The command START_PROCESS_EX starts an internal or external process. Additional to the process name, any other process-relevant setting can be passed as parameter.

Process name is the name of the started process. If the name does not correspond to an internal process, an EXE-file of the same name is tried to execute.

Priorität[opt]: determination of the execution order of processes. The process having the lowest priority is carried out at first per cycle.

Periode[Opt]: factor specifying the number of OpenXilEnv-cycles after that a process should be called cyclic.

Delay[opt]: delayed call

Timeou[opt]: Timeout after that an external process must log into OpenXilEnv at the latest.

SVL-File[opt]: SVL-file that should be loaded immediately after the login, empty string when no file.

Blackboardprefix[opt]: Prefix for all referenced variables , empty string when no prefix.

RangeControlBefore[opt]: When ≥ 1 , the referenced variable is checked on dropping or exceeding the min./max.-thresholds before calling an external process.

RangeControlBehind[opt]: When ≥ 1 , the referenced variable is checked on dropping or exceeding the min./max.-thresholds after calling the external process.

RangeCotrolStopScheduler[opt]: When ≥ 1 , the OpenXilEnv-scheduler is stopped when a dropping or exceeding of the referenced variable occurs.

RangeCotrolOutput[opt]: Specifies the where the output of the range control is written:

== 0: no output

== 1: into the error-output message-box

== 2: into the file 'script.msg'

RangeCotrolErrorCounter[opt]: Defines a blackboard-variable where in each appearance of a value range overflow is counted. The data type is UDWORD.

RangeControlVariable[opt]: Defines a blackboard-variable to switch off the value range control:

== 0: no value range control

≥ 1 : value range control activated

RangeCotrolPysical[opt]: When ≥ 1 , the value range is physically compared with min./max.-thresholds.

RangeControlLimitValues[opt]: When ≥ 1 , the values are limited to the min./max.- thresholds.

START_PROCESS_EX2 (process name, ...)

START_PROCESS_EX2 (process name, priority[opt], period[Opt], delay[opt], timeout[opt], SVL-file[opt], A2L-file[opt.], A2LFlags[opt], Blackboard prefix[opt], RangeControlBefore[opt], RangeControlBehind[opt], RangeCotrolStopScheduler[opt], RangeCotrolOutput[opt], RangeCotrolErrorCounter[opt], RangeControlVariable[opt], RangeCotrolPysical[opt], RangeControlLimitValues[opt])

The command START_PROCESS_EX2 starts an internal or external process. Additional to the process name, any other process-relevant setting can be passed as parameter.

Process name is the name of the started process. If the name does not correspond to an internal process, an EXE-file of the same name is tried to execute.

Priorität[opt]: determination of the execution order of processes. The process having the lowest priority is carried out at first per cycle.

Periode[Opt]: factor specifying the number of OpenXilEnv-cycles after that a process should be called cyclic.

Delay[opt]: delayed call

Timeou[opt]: Timeout after that an external process must log into OpenXilEnv at the lastest.

SVL-File[opt]: SVL-file that should be loaded immediately after the login, empty string when no file.

A2L-File[opt]: A2L-file that should be loaded immediately after the login, empty string when no file.

A2LFlags[opt]: If a A2L-file is defined some optionl flags can defined after the A2L-file.

1. A2L_UPDATE

This will initiate a address update after loading the A2L file, the address of all unknown labels will be to zero.

2. A2L_DLL_OFFSET

If A2L_UPDATE is defiend this have to be set if your external process live inside a DLL (not for FMUs).

3. A2L_MULTI_DLL_OFFSET

If A2L_UPDATE is defiend this have to be set if your external processes live inside more than one DLL (not for FMUs).

4. A2L_REMEMBER_REF_LABELS

If A2L_REMEMBER_REF_LABELS is set all referenced measurement will be referenced again if the external process will be restarted.

Blackboardprefix[opt]: Prefix for all referenced variables , empty string when no prefix.

RangeControlBefore[opt]: When ≥ 1 , the referenced variable is checked on dropping or exeeding the min./-max.-thresholds before calling an external process.

RangeControlBehind[opt]: When ≥ 1 , the referenced variable is checked on dropping or exeeding the min./-max.-thresholds after calling the external process.

RangeCotrolStopScheduler[opt]: When ≥ 1 , the OpenXilEnv-scheduler is stopped when a dropping or exceeding of the referenced variable occurs.

RangeCotrolOutput[opt]: Specifies the where the output of the range control is written:

== 0: no output

== 1: into the error-output message-box

== 2: into the file 'script.msg'

RangeCotrolErrorCounter[opt]: Defines a blackboard-variable wherein each appearance of a value range overflow is counted. The data type is UDWORD.

RangeControlVariable[opt]: (only SiL) Defines a blackboard-variable to switch off the value range control:

== 0: no value range control

≥ 1 : value range control activated

RangeCotrolPysical[opt]: When ≥ 1 , the value range is physically compared with min.-/max.-thresholds.

RangeControlLimitValues[opt]: When ≥ 1 , the values are limited to the min.-/max.- thresholds.

STOP_PROCESS (parameter)

STOP_PROCESS will terminate the process with the name *parameter*. For external processes it is not necessary to define the hole path. The executable name are enough.

RESET_PROCESS (parameter)

The process called *parameter* is going to be stopped and restarted automatically at the next cycle. If the process was debugged by BORLAND/MICROSOFT while the process was stoppped, than the connection with the debugger is stopped also by this instruction!

LOAD_SVL (filename, processname)

This instruction loads the SVL-file named *filename* for the process *processname*. *Filename* can be any name according to the DOS-convention. *Processname* must comply with a OpenXilEnv-registered process.

SAVE_SVL (filename, processname, opt. labelfilter)

This instruction is used to save values of lables that were selected with the *labelfilter* (e.g. **,KL***) into the SVL-file with name *filename* for the certain process *processname*. *Filename* can be any name according to the DOS-convention. *Processname* must comply with a OpenXilEnv-registered process.

SAVE_SAL (filename, processname, opt. labelfilter, opt. INC_POINTER)

This instruction is used to save lables and the address that were selected with the *labelfilter* (e.g. **,KL***) into the SVL-file with name *filename* for the certain process *processname*. *Filename* can be any name according to the

DOS-convention. *Processname* must comply with a OpenXilEnv-registered process. If the last optional parameter is set to "INC_POINTER" pointer will included otherwise no pointer will be included.

SAVE_SATVL (filename, processname, opt. labelfilter)

This instruction is used to save address, data type and values of lables that were selected with the *labelfilter* (e.g. **,KL***) into the SATVL-file with name *filename* for the certain process *processname*. *Filename* can be any name according to the DOS-convention. *Processname* must comply with a OpenXilEnv-registered process.

EXPORT_ROBFILE (filename)

This instruction serves for the export of a ROB-file from the information included in the OpenXilEnv-INI-file. The ROB-file is called *filename* . This instruction within the function is equal to the item of the file menu „Export Robfile“. It is helpful for automated test that are implemented through script-files and their evaluation is done by the programm Gredi (Copyright by Kleinknecht).

Note: If the generation of the ROB-file is not possible, the running script will not be stopped. Nethertheless there will be the text "failed" behind the instruction EXPORT_ROBFILE(name) of the debug-file.

EXPORT_ASAPFILE (filename)

This instruction is used to export an A2L-file from the information included in the OpenXilEnv-INI-file. The A2L-file is named *filename*. This instruction is equal to EXPORT_ROBFILE().

LOAD_SYMBIN (symfilename, binfilename, processname, format, hexadresse)

This instruction is used to load the SYM-file called *symfilename* and the BIN-file called *binfilename* for the process *processname* of the filling scheme *format*. The file names must be specified with/without path equal to DOS/WINDOWS. *Processname* must comply with a OpenXilEnv-registered process. *Format* is either MOTOROLA or INTEL (no statement means INTEL). The parameter *hexadresse* is optional.

SAVE_TO_BIN (symfilename, binfilename, processname, format, labelfilter)

This instruction is used to save application parameters that were selected with the *labelfilter* (e.g. **,KL***) into the BIN-file called *binfilename* for the process *processname* in the filing scheme *format*. The SYM-file called *symfilename* has to be equal to the BIN-file! The file names must be specified with/without a path equal to DOS/WINDOWS. *Processname* must comply with a OpenXilEnv-registered process. *Format* is either MOTOROLA or INTEL (no statement means INTEL).

WRITE_SECTION_TO_EXE (Prozess, Section)

This instruction is used to save a section of an application back to its executable file. This will not happend immediately. The data will be written after the external process is closed normal (not killed by taskmanager or debugger). Write back only self defined scctions and not standart sections like ".text", ".rdata", ... this can destroy the executable.

LOAD_DESKTOP(filename)

By using this instruction a saved OpenXilEnv-configuration (window, positioning at the desktop) can be loaded. DESKTOP-files can not be save via SCRIPT.

CLEAR_DESKTOP

Removes the complete contents of the desktop with all windows.

ATOMIC / END_ATOMIC

All script-commands between both of these key words are carried out at on OpenXilEnv-cycle. Especially useful for nested IF/ELSE-blocks or MESSAGE/PRINT-blocks.

!!! **Attention:** Within an ATOMIC-block there are no loops- or jump-instructions permitted, these can crash OpenXilEnv or result in an infinite loop!!!

Note: Take care of the minimum task cycle time for HiL!

LOAD_SEL_CAN_NODE(Filename, Channel)

Loads a CAN-variant and assigns it to a CAN-channel. If there are already assigned a variant(s) the loaded variant is added to the list. The channel counts from 1. That means the first channel is 1. To update the changes on the CAN-Bus, stop the CAN-server and reload it. E.g.:

```
DEL_ALL_CAN_VARIANTS
LOAD_SEL_CAN_NODE (Basis.can,1)
LOAD_SEL_CAN_NODE (TCU.can,1)
LOAD_SEL_CAN_NODE (speedrangeselector.can,1)
STOP_PROCESS (NewCANServer)
DELAY (100)
START_PROCESS (NewCSNServer)
```

The sample code first remove all CAN variants. Than load 3 variants Basis.can, TCU.can and speedrangeselector.can and connect all 3 to the channel 1. At least the CAN-Server will be restarted with the new configuration.

LOAD_CAN_VARIANT(Filename, Channel)

Loads a CAN-variant and assigns it to a CAN-channel. The channel counts from 1. That means the first channel is 1. To update the changes on the CAN-Bus, stop the CAN-server and reload it. E.g.:

```
DEL_ALL_CAN_VARIANTS
LOAD_CAN_VARIANT (vehicle.can,1)
LOAD_CAN_VARIANT (labcar.can,2)
STOP_PROCESS (NewCANServer)
DELAY (100)
START_PROCESS (NewCANServer)
```

The sample code first remove all CAN variants. Than load one variants vehicle.can, and connet this to the channel 1. Than load one variant labcar.can and connet this to the channel 2. At least the CAN-Server will be restarted with the new configuration.

APPEND_CAN_VARIANT (Filename, Channel)

Adds all CAN-objects defined in "filename" to the CAN variant which is associated with the defined "channel". All messages that were configured before at this CAN-channel remain. The appended variant will **not** exist independent in the CAN-DB. All its CAN objects will be found in the CAN variant which is associated with the defined "channel". To update the changes on the CAN-Bus, stop the CAN-server and reload it. The channel counts from 1. That means the first channel is 1. E.g.:

```
DEL_ALL_CAN_VARIANTS
LOAD_CAN_VARIANT (Basis.can,1)
APPEND_CAN_VARIANT (TCU.can,1)
APPEND_CAN_VARIANT (speedrangeselector.can,1)
STOP_PROCESS (NewCANServer)
DELAY (100)
START_PROCESS (NewCSNServer)
```

The sample code first remove all CAN variants. Than load variants Basis.can and connet this to the channel 1. Afterwards append all CAN objects from variante TCU.can to the variant connected to channel 1. Then append all CAN objects from variante speedrangeselector.can to the variant connected to channel 1. The result is one variante with the name from Basis.can includes all CAN object form Basis.can, TCU.can and speedrangeselector.can. At least the CAN-Server will be restarted with the new configuration.

DEL_ALL_CAN_VARIANTS

Deletes all CAN-variants (Attention: they are removed irrevocably from the INI-file!!!)

DEL_CAN_VARIANT (Channel)

Deletes one CAN-variant (Attention: they are removed irrevocably from the INI-file!!!) The channel counts from 1. That means the first channel is 1.

DEL_CAN_VARIANT_BY_NAME (Name)

This instruction deletes a CAN-variant from the CAN-database of OpenXilEnv independent from the allocation to a CAN-channel.

TRANSMIT_CAN(channel, Id, "Ext"optional, Size, DataByte1...8)

One-time sending a CAN-message through the CAN-bus independent of the CAN-configuration. E.g.:

```
TRANSMIT_CAN (0, 0x100, 8, 1, 2, 3, 4, 5, 6, 7, 8)
/* 11bit ID, 8 Data-Bytes */
```

```
TRANSMIT_CAN (0, 0x12345678, Ext, 4, 1, 2, 3, 4)
/* 29bit ID, 4 Data-Bytes */
```

SET_CAN_ERR (Channel, Id, Startbit, Bitsize, Byteorder, Cycles, ErrValue)

SET_CAN_ERR overwrites the number „Bitsize“ of bits from bit-position „Startbit“ of the CAN-send message with identifier „Id“ on the CAN-channel „Channel“ with the value „ErrValue“ for the duration „Cycles“. The parameter „Byteorder“ can be set to INTEL or MOTOROLA.

Special cases:

When the parameter "Startbit" is set to '-1' the data length of the CAN-object is set to value "Bitsize" for the duration of "Cycles". It must be regarded that "Bitsize" is multiple of 8 (8, 16, ..., 64). The parameters "Byteorder" and "ErrValue" are ignored thereby. If the parameter "Startbit" is set to '-2', the sending of the CAN-object, defined by parameters "Channel" and "Id", is interrupted for the duration of "Cycles" send-cycles.

SET_CAN_ERR (Channel, Id, Signalname, Cycles, ErrValue)

SET_CAN_ERR overwrites the signal of the send-CAN-message called "Signalname" on the CAN-channel "Channel". The CAN-message has the identifier "Id" and the overwritten signal the new value "ErrValue" for the duration of "Cycles".

When the parameters "Channel" and "Id" have the value '-1', the signal is searched at all CAN-channels with all objects. If the signal name appears several times within different CAN-objects, the place/CAN-object/channel found at first will be changed.

CLEAR_CAN_ERR

CLEAR_CAN_ERR stops the overwriting of Bit within a CAN-message before the end of the set duration "Cycles".

GOTO(jump label)

Works equal to BASIC. That means that there is a jump to a specified place in the script programm. Jump labels are specified as „:**JUMP LABEL**“.

A jump label may only appear once in the main file respectively all related sub script files. A GOTO in the main file must not target at a sub script file, it would cause problems! It must not be used to jump from WHILE-loops into another WHILE-loop.

Jump labels may only include alphabetic character, numbers or an underline '_' !

GOSUB(jump label) / RETURN

Works equal to BASIC. That means there is a jump to a specified place in the script programm. Jump labels are specified as „:**JUMP LABEL**“.

At the end of the GOSUB-blocks is always a RETURN to turn back to the position in the script file, where GOSUB was started.

A jump label may only appear once in the main file respectively all related sub script files. A GOSUB in the main file must not target at a sub script file because this would cause bugs! It must also not be used to jump from WHILE-loops into another WHILE-loop.

Jump labels may only include alphabetic character, numbers or an underline '_'!

BEEP

Is used to give a short information signal.

Works only when the "System-Speaker" of the PC is activated!

Regedit -- HKEY_CURRENT_USER/Control_Panel/Sound/Beep (value = yes or no to switch on/off). Possibly the system volume has to be adjusted.

START_EXE(Exe-filename,par1,par2,par3,parn)

Starts an executable program without waiting for the end of this program. Can be used e.g. to perform comparison of record files out of the script file. The error level (= return value of the EXE-program) is available at the blackboard variable **ExitCode**. Using commas after the program name, command line parameters can be passed.

Hints:

- After START_EXE generally the script.msg-file is closed, so that the called EXE can work this file.
- To carry out **DOS-Batches** do the following: **START_EXE(cmd.exe /C c:\dateiname.bat)**
- START_EXE without WAIT_EXE leads to a parallel process of OpenXilEnv and EXE-file.

Example: Calling a diff.-comparison:

```
START_EXE(cleartool,diff,-graphical,Script1.msg,Script2.msg)
```

WAIT_EXE

Waits to proceed the processing of the script file and (new since V5.00) other OpenXilEnv-processes, until the program that was started by START_EXE is finished. The error level (= return value of the EXE-program) is available at the blackboard variable **ExitCode**.

Hint: For OpenXilEnv for HiL all other processes continue process!

LOAD_REF_LIST(reflist-filename, process-filename)

This instruction loads and references a list of variables, saved in the ASCII-file named *reflist-filename*, for the process called *process-filename*. The reflist-filename-file can be generated with the instruction SAVE_REF_LIST. When loading the file, all manual referenced variables and/or a referenced list loaded before, are dereferenced.

Example:

```
LOAD_REF_LIST(reflist.txt, C:\\GSE24\\NEW\\SCRWIN.EXE).
```

Hint: The process name can be specified without path.

SAVE_REF_LIST(reflist-filename, process-filename)

This instruction leads to a storage of the manual references from the process *process-filename* into the file *reflist-filename*.

Example:

```
SAVE_REF_LIST(reflist.txt, C:\\GSE24\\NEW\\SCRWIN.EXE).
```

Hint: The process name can be specified without path.

ADD_REF_LIST (reflist-filename, process-filename)

This instruction load and references a list of variables, saved in the ASCII-file named *reflist-filename*, for the process called *process-filename*. The *reflist-filename* file can be generated with the instruction `SAVE_REF_LIST` described in {FEHLENDE REFERENZ!}.

When loading the file, all manual referenced variables and/or a referenced list loaded before, are **not** dereferenced but new references added.

Example.:

```
ADD_REF_LIST(reflist.txt, C:\\GSE24\\NEW\\SCRWIN.EXE).
```

Hint: The process name can be specified without a path.

CREATE_DIALOG/CREATE_DLG(dialogname)

A storage area is created OpenXilEnv-internal with the named *dialogname*.

For *dialogname* also special characters are permitted. This name appears in the headline of the user dialogue displayed by `SHOW_DIALOG` (also see Example B).

Hint: There must be a `SHWO_DIALOG` before the next `CREATE_DIALOG`.

ADD_DIALOG_ITEM/ADD_DLG_ITEM(selectionname, selectionvariable)

This instruction adds new items of selection, called *selectionname* to the dialogue created by `CREATE_DIALOG`. The value written behind *selectionname* is saved in the variable *selectionvariable* and available for the script. If

the second parameter is passed as string **HEADLINE**, a headline containing the first parameter is output at this position in the dialogue. If the second parameter is passed as string **NEWPAGE**, the dialogue is separated at this position and all following entries are written on the next page. Using Prev- and Next-button it can be switched between the pages (see the example).

```
*ADD_BBVARI (Test1, DOUBLE)*

*ADD_BBVARI (Test2, DOUBLE)*

*ADD_BBVARI (Test3, DOUBLE)*

*CREATE_DLG (Example Dialog Box)*

*ADD_DLG_ITEM (Headline 1 page:, \*HEADLINE\*)*

*ADD_DLG_ITEM (Input for Test1, Test1)*

*ADD_DLG_ITEM (Input for Test2, Test2)*

*ADD_DLG_ITEM (\*NEWPAGE\*, \*NEWPAGE\*)*

*ADD_DLG_ITEM (Headline 2 page:, \*HEADLINE\*)*

*ADD_DLG_ITEM (Input for Test3, Test3)*

*SHOW_DLG*
```

SHOW_DIALOG/SHOW_DLG

Shows the dialogue that was created new by CREATE_DIALOG and some ADD_DIALOG_ITEM and waits for inputs of the user. Afterwards the processing of the other OpenXiLEnv-processes will be continued.

Hints: Using OpenXiLEnv for HiL, the other processes continue! After 15 minutes without an input, the script and dialogue are exit automatically!

CREATE_RECORDER_CFG / CREATE_REC_CFG(filename){ cfgfilecontents }

Creates a config-file (cfg-file) called *filename* for the recorder. This config-file can be used several times within the scriptfile.

The config-file contains everything within the curly brackets. A syntax check of the contents takes place when executing the script.

VERBOSE(ON) and VERBOSE(OFF)

Using „ON“ additional information are included in the script-message-file (script.msg). E.g. the cycle counter, which is displayed in the Control Panel.

Standard is VERBOSE(OFF) !

VERBOSE (MESSAGE_PREFIX_CYCLE_COUNTER), this display the cycle counter in front of each debug line.

OPEN_RS232(port, configuration)

The interface can be selected by the port (permitted are e.g.: **COM1** or **COM2**). The configuration must have following settings (these values are examples only): **baud**=1200 **parity**=N **data**=8 **stop**=1.

*Hint: The additions **dtr**=off **rts**=off can be used to switch off a non-required HW-handshake. The settings can be checked through the MODE-command in a DOS-box!*

CLOSE_RS232(port)

With this instruction the opened interface can be closed again, e.g.: CLOSE_RS232(COM1)

SEND_RS232(port, text %z, variname)

This instruction allows to output text strings at an opened serial interface (number is specified with port,e.g.: COM1 or COM2). The text can contain several wildcards with %z. For variname a corresponding blackboard variable can be specified, whose contents are included in the text.

Example:

```
SEND_RS232(COM1, blablabla %z, variablename)
```

START_CCP(var1, var2, var3...)

This instruction allows to address a CCP-protocol that is implemented in the control unit. The variables specified by var1,var2... are provided continuously by the control unit and can be used in OpenXiLEnv for HiL-scripts.

START_CCP2(connection, var1, var2, var3...)

START_CCP2 differs from START_CCP that the first parameter can be a number of connection (0...3).

This instruction allows to address a CCP-protocol that is implemented in the control unit. The variables specified by var1,var2... are provided continuously by the control unit and can be used in OpenXiLEnv for HiL-scripts.

STOP_CCP

This instruction breaks the CCP-transfer.

START_CCP_CAL(parameter1, parameter2, parameter3...)

This instruction allows to break a CCP-protocol that is implemented in the control unit. The calibration parameter specified by parameter1,parameter2... are transferred to the control unit when they were changed in

the blackboard.

START_CCP_CAL2(connection, parameter1, parameter2, parameter3...)

START_CCP_CAL2 differs from START_CCP_CAL that the first parameter can be a number of connection(0...3):

This instruction allows to break a CCP-protocol that is implemented in the control unit. The calibration parameter specified by parameter1, parameter2... are transferred to the control unit when they were changed in the blackboard.

STOP_CCP_CAL

This instruction breaks the CCP-calibration.

LOAD_CCP_CFG(filename)

The CCP-Config-file called *filename* will be loaded. Further information are in the description of the CCP-process.

LOAD_CCP_CFG2(connection, filename)

LOAD_CCP_CFG2 differs from LOAD_CCP_CFG that the first parameter can be a number of connection (0...3). The CCP-Config-file called *filename* will be loaded. Further information are in the description of the CCP-process.

START_XCP(connection, var1, var2, var3...)

This instruction allows to address a XCP-protocol implemented in the control unit over CAN. The variables specified by var1, var2, var3... are provided continuously by the control unit and can be used in HiL-scripts.

The parameter [**connection**]{Source .Char} specifies the connection (0...3).

STOP_XCP (connection)

This instruction breaks the XCP-transfer. The parameter [**connection**]{Source .Char} specifies the connection (0...3).

START_XCP_CAL(connection, parameter1, parameter2, parameter3...)

This instruction allows to address a XCP-protocol implemented in the control unit over CAN. The calibration parameter chosen by parameter1, parameter2...are transferred to the control unit when they were changed in the blackboard. The parameter [**connection**]{Source .Char} specifies the connection (0...3).

STOP_XCP_CAL(connection)

This instruction breaks the XCP-calibration. The parameter [**connection**]{Source .Char} specifies the connection (0...3).

LOAD_XCP_CFG(connection, filename)

This instruction loads the XCP-config-file called *filename*. Further information can be found in the description of the XCP-process. The parameter [**connection**]{.Source .Char} specifies the connection (0...3).

REPORT(keyword1 keyword2 keywordn, text %z text, variname)

This instruction allows to output messages into a HTML-report file. The syntax is identic to the MESSAGE-command from the 2nd parameter. Following keywords are implemented currently:

Line : Inserts a dividing line

Green: Text is green

Red: Text is red

Black: Text is black

Yellow: Text is yellow

Blue: Text is blue

Bold: **Text is bold.**

Header1...Header6: The following text is displayed as headline. Smaller according to the number.

REPORT_LINK_TO(link,text)

Creates a link in the report-file whereby **link** is the referenced document (e.g.: [\\homepage.html] (file:///c:/user/homepage.html)) and **text** is the displayed text.

REPORT_RAW *Html-Text*

This instruction writes all following characters into the report.html-file directly until the end of the line. A line break can be included by the instruction REPORT_RAW_NL.

(!!Attention: the instruction requires no brackets!!)

REPORT_RAW_NL

Inserts a line break into the file report.html.

REPORT_RAW_PAR (text %z text, variname)

This instruction can be used to output messages in a HTML-report file. The syntax is identic to the MESSAGE-instruction. In contrast to the REPORT-file, there are no HTML-tags inserted.

SET_ENV_VAR (Name, Value)

This instruction sets a internal environment variable. It is only visible within the testsystem. Each %Name% is replaced through the string value after calling SET_ENV_VAR.

SET_ENV_SYS_VAR (Name, Value)

This instruction sets a OpenXiLEnv environment variable. It is visible within testsystem and each calling processes. Each %Name% is replaced through the string value after calling SET_ENV_SYS_VAR.

REMOVE_ENV_VAR (Name)

Removes the environment variable which was set by SET_ENV_VAR previously. Predefined and system environment variables can not be removed.

REMOVE_ENV_SYS_VAR (Name)

Removes the environment variable which was set by SET_ENV_SYS_VAR previously or any system environment variable.

IMPORT_HOTKEY (Filename)

This instruction loads a hotkey-definition from a file.

EXPORT_HOTKEY (Filename)

This instruction exports a hotkey-definition into a file.

NEW_REPORT_FILE (Filename)

This function closes the opened report-file and creates a new one. All following outputs are written to this report file.

OPEN_REPORT_FILE (Filename)

This function closes the open report-file and opens another already existing one. All following outputs are written into this report file.

NEW_PARAM_LIST (Listname, [Element1, Element2, ...Element_n]opt)

There is the possibility to create so called parameter lists. These parameter lists can be used instead of separate parameters in script-commands. The syntax is a prefixed \$-character followed by the name of the parameter list in square brackets, e.g.: %[parameterlistname].

The command NEW_PARAM_LIST creates a new parameter list called "Listname". The list can be filled with entries "Elementx".

Example:

```
NEW_PARAM_LIST (VariableList, Signal1, Signal2, Signal3, Signal4)
```

```
START_XCP (\$\[VariableList\])
```

Behaves like:

```
START_XCP (Signal1, Signal2, Signal3, Signal4)
```

DEL_PARAM_LIST (Listname)

The instruction DEL_PARAM_LIST deletes a parameter list that was created by the command NEW_PARAM_LIST. If the parameter list is not available, the script will not be interrupted but keeps on running without an error message.

ADD_PARAMS_TO_LIST (Listname, Element1, Element2, ... Element_n)

The instruction ADD_PARAMS_TO_LIST inserts further parameter elements "Elementx" into the parameter list "Listname".

Example:

```
NEW_PARAM_LIST (VariableList, Signal1, Signal2, Signal3, Signal4)

ADD_PARAMS_TO_LIST (VariableList, Signal5, Signal6)

START_XCP (\$\[VariableList\])
```

Behaves like:

```
START_XCP (Signal1, Signal2, Signal3, Signal4, Signal5, Signal6)
```

DEL_PARAMS_FROM_LIST (Listname, Element1, Element2, ... Element_n)

The instruction DEL_PARAMS_FROM_LIST deletes parameter elements „Elementx" from a parameter list „Listenname".

Example:

```
NEW_PARAM_LIST (VariableList, Signal1, Signal2, Signal3, Signal4, Signal5)

DEL_PARAMS_FROM_LIST (VariableList, Signal2, Signal3)

START_XCP (\$\[VariableList\])
```

Behaves like:

```
START_XCP (Signal1, Signal4, Signal5)
```

START_MEASUREMENT (Connection-protocol , ConnectionNumber, Variable1, Variable2, ...Variablen)

START_MEASUREMENT is a general possibility to start the measurement of variables. The protocol used is defined through the second parameter "Connection-protocol".

Following values are possible for "Connection-protocol":

XCP: The XCP-Protocol is used (Description see START_XCP)

CCP: The CCP-Protocol is used (Description see START_CCP)

Or a process name of an external process e.g.: TCU.EXE (only SiL)

Example:

```
IF (Realtime == 1)

SET_ENV_VAR (Protocol, XCP)

ELSE

SET_ENV_VAR (Protocol, TCU.EXE)

ENDIF

...

START_MEASUREMENT (%Protocol%, 0, Signal1, Signal2, Signal3)
```

STOP_MEASUREMENT (Connection-protocol, ConnectionNumber)

The instruction STOP_MEASUREMENT exits a measurement that was started by START_MEASUREMENT. The protocol used is defined by the second parameter "Connection-protocol".

Following values are possible for "Connection-protocol":

XCP: The XCP-Protocol is used (Description see START_XCP)

CCP: The CCP-Protocol is used (Description see START_CCP)

Or a process name of an external process e.g.: TCU.EXE (only SiL)

START_CALIBRATION (Connection-protocol, ConnectionNumber, Parameter1, Parameter2, ...Parameter_v)

The START_CALIBRATION is a general possibility to start a parameter calibration. The protocol used is defined by the second parameter "Connection-protocol".

Following values are possible for "Connection-protocol":

XCP: The XCP-Protocol is used (Description see START_XCP_CAL)

CCP: The CCP-Protocol is used (Description see START_CCP_CAL)

Or a process name of an external process e.g.: TCU.EXE (only SiL)

The parameter "ConnectionNumber" can have a value between 0 and 3.

Example:

```
IF (Realtime == 1)

SET_ENV_VAR (Protocol, XCP)

ELSE

SET_ENV_VAR (Protocol, TCU.EXE)

ENDIF

...

START_CALIBRATION (%Protocol%, 0, Parameter1, Parameter2)
```

STOP_CALIBRATION (Verbindungs-Protokoll, Verbindungsnummer)

The instruction STOP_CALIBRATION exits a calibration-connection that was started by START_CALIBRATION. The protocol used is defined by the second parameter "Connection-protocol".

Following values are possible for "Connection-protocol":

XCP: The XCP-Protocol is used (Description see STOP_XCP_CAL)

CCP: The CCP-Protocol is used (Description see STOP_CCP_CAL)

Or a process name of an external process e.g.: TCU.EXE (only SiL)

CHANGE_SETTINGS (Settingname, Settingvalue)

Using the instruction CHANGE_SETTINGS certain global settings of XiEnv can be changed out of the script. Following "Settingname" are possible:

Setting	Value(s)	Description
NOT_FASTER_THAN_REALTIME	Yes or No	OpenXiEnv does not run faster than real time. This setting is ignored at OpenXiEnv for HiL.
DONT_MAKE_SCRIPT_DEBUGFILE	Yes or No	Do not create a Debug-file when executing the script.

Setting	Value(s)	Description
SCRIPT_START_EXE_AS_ICON	Yes or No	Programs called by the script-command START_EXE are started as Icon.
SCRIPT_STOP_IF_CCP_ERROR	Yes or No	When executing the script-instructions START_CCP, STOP_CCP, START_CCP_CAL or STOP_CCP_CAL an error occurs, the script is exit with an error message.
SCRIPT_STOP_IF_XCP_ERROR	Yes or No	When executing the script-instructions START_XCP, STOP_XCP, START_XCP_CAL or STOP_XCP_CAL an error occurs, the script is exit with an error message.
STOP_EQUATION_IF_SCRIPT_STOPPED	Yes or No	Specifies whether the global equation-block is stopped when ending the script. This is set to "yes" by default.
STOP_GENERATOR_IF_SCRIPT_STOPPED	Yes or No	Specifies wether the ramp-generator is stopped when ending the script. This is set to "yes" by default.
STOP_RECORDER_IF_SCRIPT_STOPPED	Yes or No	Specifies wether the recorder is stopped when ending the script. This is set to "yes" by default.
STOP_PLAYER_IF_SCRIPT_STOPPED	Yes or No	Specifies wether the stimuli-player is stopped when ending the script. This is set to "yes" by default.
DONT_INIT_EXISTING_VARIABLES_DURING_RESTART_OF_CAN	Yes or No	Specifies wether the initialization-values of signals, which are available at the blackboard, are written when starting the CAN-server. This is set to "no" by default.
ADD_BBVARI_DEFAULT_UNIT	String	Specifies the default unit of a variable created by a ADD_BBVARI script command (if parameter unit is not set).

Setting	Value(s)	Description
DONT_WAIT_FOR_RESPONSE_OF_START_RECORDER_RPC	Yes or No	Specifies wether the RPC-Call XilEnv_StartRecorder wait until the recorder is running or the function is return immediately. If XilEnv_StartRecorder called inside a cyclic function this have to set to "Yes" otherwise the call of XilEnv_StartRecorder will end in a death lock.
SAVE_REF_LIST_IN_NEW_FORMAT_AS_DEFAULT	Yes or No	Specifies wether the the reference list (SCRefL) is exported in the new format (variable_name = display_name) old indexed format (Refx = variable_name [Dspx = display_name])
CONVERT_ERRORS_TO_MESSAGE	Yes or No	If it is set to Yes all popup errors and warnings will be confirmed automaticly with OK. The message will be witten to the script.msg file. Same behavior as the parameter -err2msg
STOP_SCRIPT_IF_LABEL_DOS_NOT_EXIST_INSIDE_SVL	Yes or No	If it is set to Yes script will be stopped with an error message. Default value is No.

CHANGE_SETTINGS_PUSH (Settingname, Settingvalue)

Using the instruction CHANGE_SETTINGS_PUSH certain global settings of OpenXiLEnv can be changed out of the script. Before the change take place the old value will we stored inside an internal stack. You can fetch the saved value from the stack and restore the the setting with the command CHANGE_SETTINGS_POP. The possible "Settingname" are the same as descript in the section CANGE_SETTINGS.

CHANGE_SETTINGS_POP (Settingname)

The instruction CHANGE_SETTINGS_POP will fetch a setting value that are stored before with the command CHANGE_SETTINGS_PUSH from the internal stack and restore it.

IMPORT_VAR_PROPERTIES (Filename)

Imports the settings of all blackboard-variables included in the file "Filename".

Thereby the following properties are overwritten: unit, min-/max-thresholds, conversion, display, increment and default color.

SELECT_SHEET (SheetName)

Switches display to sheet "SheetName".

ADD_SHEET (SheetName)

Inserts a new sheet called „SheetName" into the display.

DELETE_SHEET (SheetName)

Deletes an existing sheet called "SheetName".

RENAME_SHEET (OldSheetName, NewSheetName)

Changes the name of an existing sheet called "OldSheetName" into "NewSheetName".

OPEN_WINDOW (WindowName)

Opens a display-window called "WindowName". The window must exist so it had to be opened once before. Wildcards (*,?) are allowed.

CLOSE_WINDOW (WindowName)

Closes an opened display-window „WindowName". Wildcards (*,?) are allowed.

DELETE_WINDOW (WindowName)

Deletes a window named „WindowName" from the INI-file. If the window is opened, it will be closed before deleting it. Wildcards (*,?) are allowed.

IMPORT_WINDOW (WindowName, FileName)

Imports display-windows from the file „FileName" and opens them in the current sheet. Wildcards (*,?) are allowed.

EXPORT_WINDOW (SheetName, WindowName, FileName)

Exports display-windows from the file „FileName". Wildcards (*,?) are allowed.

SET_MIN (Variablename, Min.-Value)

This instruction sets the minimum value of a blackboard variable.

SET_MAX (Variablename, Max.-Value):

This instruction sets the maximim value of a blackboard variable.

SET_MIN_MAX (Variablename, Min.-Value, Max.-Value)

This instruction sets minimum and maximum value of a blackboard variable simultaneously.

ENABLE_RANGE_CONTROL (Processname, Variable1, Variable2, ... VariableX)

This instruction enables the range monitoring of a process for certain variables. By default the range monitoring is enabled.

Both process name and variables can include wildcards (*?).

DISABLE_RANGE_CONTROL (Processname, Variable1, Variable2, ... VariableX)

This instruction disables the range monitoring of a process for certain variables. By default the range monitoring is enabled.

Both process name and variables can include wildcards (*?).

SET_CAN_CHANNEL_COUNT (Channel number)

This instruction sets the number of active CAN-channels (value range 0...7).

DEF_PROC (Procedure name, Parameter_1, Parameter_2, ..., Parameter_n)

Beginning of a procedure named " Procedure name". 0 to a maximum of 50 parameters can be passed to a procedure. Within the procedure the parameters can be used like blackboard-variables, but they are not visible at the blackboard. The data type of the parameters is always Double (64bit floating point). A DEF_PROC must always end with an END_DEF_PROC. A parameter can be defined as a reference if a '*' is defined as a prefix. If you want access such a reference you should also add the '*' prefix.

Example:

```
DEF_PROC (MyProc, a, b, c)
  MESSAGE ("MyProc got a=%z b=%z c=%z" , a, b, c)
END_DEF_PROC
CALL_PROC (MyProc, 1, 1+1, 3)
```

Example with reference:

```
DEF_PROC (MyProc, a, b, *c)
  MESSAGE ("MyProc got a=%z b=%z", a, b)
  SET (*c = a + b)
END_DEF_PROC
DEF_LOCALS(c)
  CALL_PROC (MyProc, 1, 2, &c)
  MESSAGE("%z + %z = %z", 1, 2, c)
END_DEF_LOCALS
```

END_DEF_PROC

This instruction ends the definition of a procedure (also see DEF_PROC)

CALL_PROC (Procedure name, Parameter_1, Parameter_2, ..., Parameter_n)

Calls a procedure named " Procedure name". 0 to a maximum of 50 parameters can be passed. The number of parameters must be equal to the number of parameters from the DEF_PROC-instruction. Only values or references can be passed as parameters. If you want pass a reference you have to add the '&' prefix.

Example:

```
DEF_PROC (MyProc, a, b, c)
  MESSAGE (MyProc got a=%z b=%z c=%z , a, b, c)
END_DEF_PROC
CALL_PROC (MyProc, 1, 1+1, 3)
```

DEF_LOCALS (Local_1, Local_2, ..., Local_n)

Creates local variables, which are valid from the instruction DEF_LOCALS to the corresponding END_DEF_LOCALS. Predefined local variables or blackboard-variables are interfered. Local variables can be used as blackboard-variables within DEF_LOCALS/END_DEF_LOCALS-block, but they are not visible at the blackboard. The data type of local variables is always Double (64bit floating point). All variables are initialized with '0'.

Example:

```
DEF_LOCALS (a, b, c)
SET (a = 1)
SET (b = a + 1)
SET (c = a + b)
MESSAGE (Local variable a=%z b=%z c=%z, a, b, c)
END_DEF_LOCALS
```

END_DEF_LOCALS

This instruction removes all local variables that were defined by the DEF_LOCALS-instruction.

USING (Script file name)

This instruction visualize all procedures defined in the script file "Script file name" and they can be called by CALL_PROC.

Example:

ExtProc.scr:

```
DEF_PROC (MyProc, a, b, c)
MESSAGE (MyProc got a=%z b=%z c=%z , a, b, c)
END_DE_PROC
```

Main.scr:

```
USING (%XILENV_SCRIPT_DIR%\ExtProc.scr)
CALL_PROC (MyProc, 1, 1+1, 3)
```

SET_CAN_SIG_CONV (Channel, ID, Signalname, Conversion)

This instruction implements a conversion in the active CAN-simulation of the remaining bus. This change is active as long as the CAN-server was new initialized or canceled through RESET_CAN_SIG_CONV. **Channel** is the CAN-channel number (0...3), **ID** is the CAN-identifier of the object, **Signalname** is the signal in the CAN-object which conversion should be changed and **Conversion** is the new calculation rule for this signal.

If **Channel** has the value '-1' object/signal are searched on all CAN-channels.

If **ID** has the value '-1' the signal is searched on all CAN-channels.

RESET_CAN_SIG_CONV (Channel, ID, Signalname)

This instruction resets the changes in the active CAN-simulation of the remaining bus that was carried out by one or more SET_CAN_SIG_CONV-instructions. **Channel** is the CAN-channel number (0...3), **ID** is the CAN-identifier of the object, **Signalname** is the signal in the CAN-object whose conversion should be reset. If the signal name is left out, all changes on the CAN-bus and the chosen object are reset. If **ID** has the value '-1' or is left out, all changes in all CAN-objects are reset. If **Channel** has the value '-1' or is left out, all changes are reset.

START_CAN_RECORDER (filename.txt, trigger event, [TIME]opt., channel, start id, end id, [channel, start id, end id]opt.(10 times))

With this instruction the CAN message recorder function of OpenXiLEnv is started. It will write the receive and transmit messages to the file "filename.txt". A "trigger event" can be defined. If no trigger event are needed a empty string "" must be used. With the optional parameter TIME can be switch on the first column used for the timestamp. Only one recorder can be started at one time. A before started CAN recording will be stopped.

STOP_CAN_RECORDER

With this instruction the CAN message recorder function of OpenXiLEnv will be stopped.

9.2.2. Only HiL instructions

9.2.3. Environment variable within script-instructions

The parameter from most of the script-instructions can contain so called environment variables. The syntax for the access to an environment variable is defined as follows: %VARIABLE%. Each appearance of an environment variable is replaced by corresponding text, before executing the instruction (similar to the C-preprocessor).

There are 3 types of environment variables, the search order is: 1. Constant defined environment variable, 2. freely definable, 3. System environment variable.

Constant defined environment variable

there are a part of predefinition environment variables:

Name	Meaning
%XILENV_INI_DIR%	Directory of the current loaded INI file
%XILENV_INI_PATH%	Path to the loaded INI file
%XILENV_SCRIPT_NAME%	Name of the current executed script file
%XILENV_SCRIPT_DIR%	Directory of the current active script file
%XILENV_SCRIPT_LINE%	Current line number inside running script
%XILENV_WORK_DIR%	Workspace from basic-settings
%XILENV_EXE_DIR%	Directory of OpenXiEnv-EXE
%XILENV_VARIABLE:Blackboardvariablenname	Value of a blackboard-variable
%XILENV_PROCESS_DIR:ProzessName%	Directory of an externalprocess
%XILENV_CURRENT_DIR%	Current directory
% XiEnv_CCP_SWSTAND%	Name of the software-status that was read by CCP. Only valid for enabled CCP-connection, otherwise it returns "error". Identic to %XILENV_CCP0_SWSTAND%
%XILENV_CCP0_SWSTAND%, %XILENV_CCP1_SWSTAND%, %XILENV_CCP2_SWSTAND%, %XILENV_CCP3_SWSTAND%	Name of the software-status that was read by XCP-connection 0 to 3. Only valid for enabled XCP-connection, otherwise it returns "error".
%XILENV_XCP0_SWSTAND%, %XILENV_XCP1_SWSTAND%, %XILENV_XCP2_SWSTAND%, %XILENV_XCP3_SWSTAND%	Name of the software-status that was read by XCP-connection 0 to 3. Only valid for enabled XCP-connection, otherwise it returns "error".
%XILENV_RANDOM%	A random number
%XILENV_TIME_STRING%	Time as string hh:mm:ss
%XILENV_TIME2_STRING%	Time as string hh.mm.ss no ':' but '.' as separator
%XILENV_DATE_STRING%	Time as string

Name	Meaning
%XILENV_WINDOWS_TICK_COUNTER%	Returns the time in milliseconds since the start of Windows-system (overflow after 49,7 days)
%XILENV_BASIC_SETTINGS:BasicSettingName%	Returns the value of the settings value. Possible names are listet inside the section CHANGE_SETTINGS
%XiEnv_Instance%	Returns the instance name set with the transfer parameter-Instance

Freely defineable environment variable

Using the script-instruction SET_ENV_VAR (Name, Value) new environment variables can be added or overwritten at any time. The constant predefined environment variables cannot be overwritten. The search order are: 1. constant predefined, 2. user defined, 3. defined inside INI file, 4. system environment variables.

Example:

```
SET_ENV_VAR (MY_ENV_VAR, Text)
MESSAGE (MY_ENV_VAR = %MY_ENV_VAR%)
```

Output: "MY_ENV_VAR = Text"

System-environment variable

All environment variables defined by the operating system.

Example:

```
MESSAGE (Windows-directory = %SystemRoot%)
```

Output: "Windows-directory = C:\WINDOWS"

The script command SET_ENV_SYS_VAR and REMOVE_ENV_SYS_VAR can be used to change system environment variable for OpenXiEnv and each afterwards called process.

9.3. Notes on the instructions

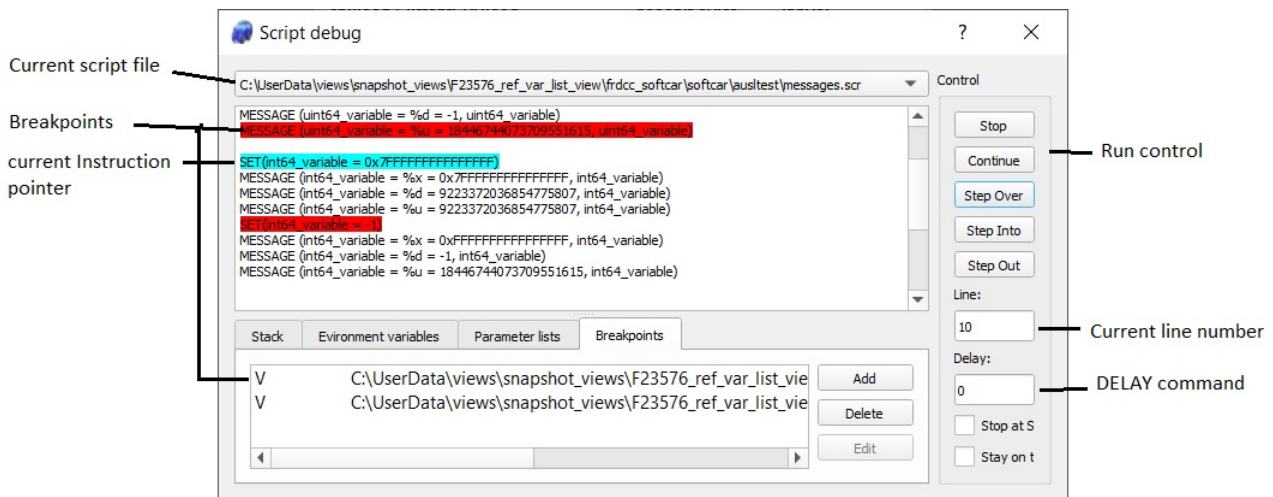
- **RS232**-instructions: Using the **MODE**-instruction in a DOS-box the settings of the serial interface can be requested.
- One line of the script file must not contain **more than 8191 characters!!!**
- In one script file including all called subscripts may only contain a **maximum of "unsigned long" lines!!!**
- **Parameter** of an instruction must also stand **in one line!!!**

- A line can be break into several lines by using the '\'-character equal to the C-preprocessor (Attention: the max. number of characters is also limited to 8191).
- The length of a **parameter / equation** is limited to the maximum length of the line.
- For letters within a **parameter string** the lower case is recommended! (better to difference to the instruction words which are written with upper case).
- For each *filename* the extension (doc, scr, ...) [**must**]{**.underline**} be specified.
- For each *filename* the path may be specified.
- The **instruction words** are only accepted in **upper case!!!**
- In front of **instructions** may stand any number of **whitespaces**.
- Between the **intruction word** and an **instruction parameter** (e.g. filename) respectively in from of an instruction word may stand any number of **whitespaces**.
- For **Variablen names** the upper/lower case is also evaluated**!!!**
- The instruction '**RUN/CALL scriptfile**' may have a nesting depth of up to 20 subprograms calls, that is: Main ->Uprg1->Uprg2...->Uprg5.
- When filenames have no **path** specified with, the first command should be a 'CD()', to specify the basic path for all following file accesses.
- **IF-ELSE-ENDIF**-instructions can be nested up to 20 times.
- **WHILE-ENDWHILE**-instructions can be nested up to 20 times.
- Using the instructions *Start_Recorder*, *Start_Rampe*, *Start_Trigger*, *Start_Player* with curly brackets, in each case a temporary file is created. Therefore the last instruction within a scriptfile should be '**DEL (~*.*)**' if these created files should not be used again or they disturb.
- Within curly brackets {} script instructions are not permitted!
- The **Syntax** between curly brackets {} is not verified by the script-interpreter!
- Within **curly brackets { }** only the semicolon is permitted to mark **comments!!!**
- Nested **comments** are permitted (32766-times).
- C++-**comments '//'** are not permitted!!
- A maximum of 50 **GOTOs** or **GOSUBs** may occur in a script file and all called subscript files.
- The maximum nesting depth for **GOTO / GOSUB** is 50 at the moment.
- To comment out a series of instructions, use the following: **/* BEFEHL 1 BFFEHL n */** not permitted: **BEFEHL 1 /* BEFEHL n */**
- To exclude a parameter from an instruction the following is permitted: **BEFEHL /*(parameter)*/ (paramter_new)**

9.4. Script debug window

Hint: The script command VERBOSE(ON) can be very helpful for debugging. Aso a script debug file, with all executed commands, will be written (may be you have to disable inside the OpenXiLEnv basic settings dialog tab "script" the checkbox "do not make a script debug file"). The "script.dbg" file can be open with the "bug" button inside the toolbar of the OpenXiLEnv main window.

The script debug window was implemented to control and follow the execution of a script file. To get to the script debug-mode, turn on the button 'Script-Debugging' onto the OpenXiLEnv-control-panel. Atferwards the following window will be opened immediately:



The line of the script-file that should execute next are displayed blue. Breakpoints are marked red. An automated indentation is done to improve the readability. In the lower section of the window, four tabs are displayed.

The following for tabs are available:

- **Stack:** Inside this tab the current stack is displayed. You can see there the function, script calling stack. You can also see the values of local defined variables here.
- **Environment variables:** Inside this tab all user defined environment variables are shown (defined with SET_ENV_VAR).
- **Parameter lists:** Displays all current defined parameter list (defined with the NEW_PARAM_LIST script command).
- **Breakpoint:** Displays all current active breakpoints.

Procedure of debugging:

Always open the debug-window at first!

1. The 'STOP'-button on the control-panel should be pressed after starting OpenXiLEnv. Afterwards the script file can be started to debug. Using the buttons (NEXT_ONE, NEXT and CONTINUE) of the control-panel, the execution of the script file can be controlled.
2. Setting the command '**STOP_OpenXiLEnv**' at the point of failure within the script-file. Then start the script-file. When the STOP_OpenXiLEnv-command is reached, the execution of the script-file is stopped.

immediately and waits for an input of the user.

Hints:

- The contents of the debug window are not restored after closing the debug window - but is available at the file 'SCRIPT.DBG'.
- In a script-file a breakpoint can be set by using the script-command '**STOP_OpenXiLEnv**'.
- Using IF/ENDIF-commands directly in a row, only one script-cycle is needed!

9.5. Other

9.5.1. The 'Script_Status_Flag'

Displayed under 'Status' within the control-panel (if script was chosen!)

This variable contains the state of the process 'Script' as numerical value and can have the following states/values:

Value	Meaning	Description
0	sleep	displays a syntactic clean script-file that is ready to be executed by the script-process
1	start	will be set by OpenXiLEnv or user to start the execution of the script file.
2	stop	will be set by OpenXiLEnv or user to stop the execution of the script-file. After stopping the script_status_flag is set to 'READY'. Also set by EXIT-commands.
3	running	Displays that a script-file is currently executed.
4	ready	is set to display that the script-file is completely executed.
5	finished	displays that all resources used by the script-functions are free.
6	error	Error occurs during the syntax check or serious error while executing the script-file. Cause of syntax-error is written into the file SCRIPT.ERR . Cause of execution-error is written into the file SCRIPT.DBG . Serious errors can result in the termination of the script-process, that means XiLEnv removes the process 'Script'. As a result no other script-files can be executed anymore.

Note: Only the states START and STOP can be set by the user; all other states are blocked for the access of the user!

9.5.2. Error messages

Error messages are written into the file '**script.err**'. Additionally to the error, the line number of the script-file where the error occurred, will be output.

When an error message does not make any sense, it is often the result of a syntax-error between the curly brackets {}.

Semantic errors, that do not make an entry in the file '**script.err**' can be removed by reading the file '**script.dbg**'.

The '**script.dbg**'-file will be created after finishing the syntax-check, that means creation of the '**script.err**'-file, successful.

The '**script.dbg**'-file will be deleted at every restart of a script file!!!

When no script-file is executable and also the status of the process 'Script' does not change on the OpenXilEnv-control-panel, the process 'Script' was removed by OpenXilEnv for certain. OpenXilEnv must be closed and restarted that scripts can be executed again!

When using automated OpenXilEnv-runings, that means the executed script will be specified as calling-parameter, the diversion of OpenXilEnv-error messages into the **script.msg**-file can be forced by the command line parameter 'ERR2MSG'.

9.5.3. Notes

- The instructions explained are understood as basis that can be expanded by additional instructions, according to the needs of the user.
- The user should give a feedback, if the information in the files SCRIPT.ERR and SCRIPT.DBG are helpful to them!!! (Which additional information are required?!)
- Script-files should have the extension 'SCR' to mark them clearly.

9.5.4. Tips & Tricks

- After the creation of a script-file it can be used and started in OpenXil. If nothing happens, look at the control-panel, if the status is 'ERROR'. If so, open and read the file **script.err** in the current directory.
- When the scriptfile runs, but not as expected, take a look at the file **script.dbg**. Here's the information what the script-interpreter has recognized and executed. This file can also be used as template for new script-files.
- The temporary-files (~*.*) which were created by the instructions 'START_...' with curly brackets, can be copied to 'Name.*'. Then they can be used as 'normal' configuration files.

9.5.5. Examples

10. Remote Control

XilEnv has a **Remote Procedure Call** interface for the automation with external programs (Python, C++/C, Perl, Java, ...). The following files are needed additionally:

XilEnvRpc.dll: This dll can be used inside a 64 bit executable to use the OpenXilEnv remote API. It has to be placed in the searched path.

include\XilEnvRpc.h: for C und C++ only

XilEnvRpc.py: This file include a wrapper class for python.

10.1. Connection buildup and others

10.1.1. XilEnv_ConnectTo

C:

```
int XilEnv_ConnectTo(const char* NetAddr);
```

VB:

```
Public Declare Function XilEnv_ConnectTo (ByVal NetAddr As String) As Long
```

The function 'XilEnv_ConnectTo' builds up to connection to a running OpenXilEnv. A name of a remote-computer can be specified by the parameter 'NetAddr'. When OpenXilEnv is running on the same PC, use a empty string ("") as parameter. This function can only connect to a OpenXilEnv instance with no name. If you have started OpenXilEnv with an instance name you have to use XilEnv_ConnectToInstance function instead.

Return value:

0 -> OK

-1 -> Error

-2 -> OpenXilEnv(RT) version is not equal to the version of XilEnvpcdll.dll respectively XilEnv2py.dll.

10.1.2. XilEnv_ConnectToInstance

C:

```
int XilEnv_ConnectToInstance(const char* NetAddr, const char* InstanceName);
```

VB:

```
Public Declare Function XilEnv_ConnectToInstance (ByVal NetAddr As String, ByVal InstanceName As String) As Long
```

The function 'XilEnv_ConnectToInstance' builds up to connection to a running OpenXilEnv instance. A name of a remote-computer can be specified by the parameter 'NetAddr'. When OpenXilEnv is running on the same PC, use a empty string ("") as parameter. If OpenXilEnv started with no instance you can define InstanceName as an empty string (""). Than this function is the same as XilEnv_ConnectTo.

Return value:

0 -> OK

-1 -> Error

-2 -> OpenXilEnv(RT) version is not equal to the version of XilEnvpcdll.dll respectively XilEnv2py.dll.

10.1.3. XilEnv_DisconnectFrom

C:

```
int XilEnv_DisconnectFrom(void);
```

VB:

```
Public Declare Function XilEnv_DisconnectFrom () As Long
```

XilEnv_DisconnectFrom stops the connection to OpenXilEnv. No other instructions apart from 'XilEnv_ConnectTo' are allowed afterwards. If OpenXilEnv should also be closed use XilEnv_DisconnectAndClose instead.

Return value is always 0.

10.1.4. XilEnv_DisconnectAndClose

C:

```
int XilEnv_DisconnectAndClose(int SetErrorLevelFlag, int ErrorLevel);
```

VB:

```
Declare Function XilEnv_DisconnectAndClose  
    (ByVal SetErrorLevelFlag As Long, ByVal ErrorLevel As Long) As Long
```

'XilEnv_DisconnectAndClose' close the connection to OpenXilEnv and close it. If the value of SetErrorLevelFlag is not zero OpenXilEnv gives back "ErrorLevel" as exit code. The return value of this function is always zero.

10.1.5. XilEnv_IsConnectedTo

C:

```
int XilEnv_IsConnectedTo(void);
```

VB:

```
Declare Function XilEnv_IsConnectedTo () As Long
```

'XilEnv_IsConnectedTo' checks if a connection to OpenXilEnv exists. Return value is '1' when a connection exists, otherwise the return value is '0'

10.1.6. XilEnv_GetVersion

C:

```
int XilEnv_GetVersion(void);
```

VB:

```
Declare Function XilEnv_GetVersion () As Long
```

'XilEnv_GetVersion' returns the version of OpenXilEnv. You have to call the function 'XilEnv_ConnectTo' before. The return version number is the OpenXilEnv version multiplied by 100.

10.1.7. XilEnv_GetAPIVersion

C:

```
int XilEnv_GetAPIVersion(void);
```

VB:

```
Declare Function XilEnv_GetAPIVersion () As Long
```

'XilEnv_GetAPIVersion' returns the version of library OpenXilEnv. You can call the function before 'XilEnv_ConnectTo'. The return version number is the OpenXilEnv version multiplied by 100.

10.1.8. XilEnv_GetAPIModulePath

C:

```
char* XilEnv_GetAPIModulePath (void);
```

VB:

```
Declare Function XilEnv_GetAPIModulePath () As String
```

'XilEnv_GetAPIModulePath' returns the location path of the loaded library. You can call the function before 'XilEnv_ConnectTo'.

10.1.9. XilEnv_CreateFileWithContent

C:

```
int XilEnv_CreateFileWithContent (const char *Filename, const char *Content);
```

VB:

```
Public Declare Function XilEnv_CreateFileWithContent  
    (ByVal Filename As String, ByVal Content As String) As Long
```

'XilEnv_CreateFileWithContent' saves a string '[Content]{.Source.Char}' to a file '[Filename]{.Source.Char}' from the view of OpenXilEnv. The current path-position from the calling program is not significant.

Return '0' when everything is OK, '-1' when the file could not be created or '-2' when not the complete contents could be written (full disk).

10.1.10. XilEnv_SetEnvironVar

C:

```
int XilEnv_SetEnvironVar (const char *EnvironVar, const char *EnvironValue);
```

VB:

```
Public Declare Function XilEnv_SetEnvironVar (ByVal EnvironVar As String,  
    ByVal EnvironValue As String) As Long
```

Set the value of a OpenXilEnv environment-variable. If the environment-variable doesn't exists it will be added. If it exists the value will be changed. You cannot overwrite fixed or system environment-variable.

10.1.11. XilEnv_GetEnvironVar

C:

```
char* XilEnv_GetEnvironVar (const char *EnvironVar);
```

VB:

```
Public Declare Function XilEnv_GetEnvironVar (ByVal EnvironVar As String) As String
```

Returns the value of a OpenXilEnv environment-variable.

10.1.12. XilEnv_ChangeSettings

C:

```
int XilEnv_ChangeSettings (const char *SettingName, ValueString);
```

Changes the global settings of OpenXilEnv. The following settings are changeable currently:

Name	Description	Values
NOT_FASTER_THAN_REALTIME	OpenXilEnv does not run faster than the simulated time, but it can be slower (ignored by HiL	Yes/No
DONT_MAKE_SCRIPT_DEBUGFILE	Do not create a Script-debug-file. Useful for endurance tests, otherwise the debug-files would become very big.	Yes/No

10.1.13. XilEnv_TextOut

C:

```
int SCTextOut (const char *Text);
```

Outputs a text in the OpenXilEnv-message-window and writes it into the 'Script.msg'-file.

10.2. XilEnv_ErrorTextOut

C:

```
int SCErrorTextOut (int ErrLevel, const char *Text);
```

Outputs an error message in OpenXilEnv. Following error-levels are defined:

MESSAGE_ONLY 0

MESSAGE_STOP 1

ERROR_NO_CRITICAL 2

ERROR_NO_CRITICAL_STOP 3

ERROR_CRITICAL 4

ERROR_SYSTEM_CRITICAL 5

ERROR_OKCANCEL 6

RT_ERROR 7

ERROR_OK_OKALL_CANCEL 8

INFO_NO_STOP 9

WARNING_NO_STOP 10

ERROR_NO_STOP 11

RT_INFO_MESSAGE 12

10.3. Scheduler

10.3.1. XilEnv_StopScheduler

C:

```
void XilEnv_StopScheduler(void);
```

VB:

```
Public Declare Sub XilEnv_StopScheduler ()
```

XilEnv_StopScheduler stops the scheduler, no other processes are carried out.

10.3.2. XilEnv_ContinueScheduler

C:

```
void XilEnv_ContinueScheduler(void);
```

VB:


```
Public Declare Sub XilEnv_ContinueScheduler ()
```

XilEnv_ContinueScheduler restarts the schedules

10.3.3. XilEnv_IsSchedulerRunning

C:

```
int XilEnv_IsSchedulerRunning(void);
```

VB:

```
Public Declare Sub XilEnv_SchedulerRunning () As Long
```

XilEnv_IsSchedulerRunning verifies if the scheduler was stopped (return-value = 0) or if it is running (return-value = 1).

10.3.4. XilEnv_DoNextCycles

C:

```
void XilEnv_DoNextCycles (int Cycles);
```

VB:

```
Public Declare Sub XilEnv_DoNextCycles (ByVal Cycles As Long)
```

XilEnv_DoNextCycles starts the scheduler for the duration of parameter-defined "Cycles" (number of cycles). The function returns immediately and does not wait for the cycles to be executed.

10.3.5. XilEnv_DoNextCyclesAndWait

C:

```
void XilEnv_DoNextCyclesAndWait (int Cycles);
```

XilEnv_DoNextCyclesAndWait starts the scheduler for the duration of parameter-defined "Cycles" (number of cycles). The function does not return immediately and waits for the cycles to be executed.

10.3.6. XilEnv_StartProcess

C:

```
int XilEnv_StartProcess(unsigned char* name);
```

VB:

```
Public Declare Function XilEnv_StartProcess (ByVal Name As String) As Long
```

XilEnv_StartProcess starts a OpenXilEnv-process (internal or external)

Return value:

>0 -> internal process was started (Process-ID)

=0 -> external process was started.

<0 -> error

-810 -> process is already running.

10.3.7. XilEnv_StartProcessAndLoadSvl

C:

```
int XilEnv_StartProcessAndLoadSvl (unsigned char* name, const char* Sv1Name);
```

VB:

```
Public Declare Function XilEnv_StartProcessAndLoadSvl (ByVal Name As String,  
    ByVal Sv1Name As String) As Long
```

XilEnv_StartProcessAndLoadSvl starts an external OpenXilEnv-process and load directly a SVL file if the process is successful.

Return value:

>0 -> internal process was started (Process-ID)

=0 -> external process was started.

<0 -> error

-810 -> process is already running.

10.4. XilEnv_StartProcessEx

C:

```
int XilEnv_StartProcessEx(const char* ProcessName,
                        int Prio,
                        int Cycle,
                        short Delay,
                        int Timeout,
                        const char *SVLFile,
                        const char *BBPrefix,
                        int UseRangeControl,
                        int RangeControlBeforeActiveFlags,
                        int RangeControlBehindActiveFlags,
                        int RangeControlStopSchedFlag,
                        int RangeControlOutput,
                        int RangeErrorCounterFlag,
                        const char *RangeErrorCounter,
                        int RangeControlVarFlag,
                        const char *RangeControl,
                        int RangeControlPhysFlag,
                        int RangeControlLimitValues);
```

XilEnv_StartProcessEx starts an internal or external process. Additional to the process name, any other process-relevant setting can be passed as parameter.

ProcessName is the name of the started process. If the name does not correspond to an internal process, an EXE-file of the same name is tried to execute.

Prio: determination of the execution order of processes. The process having the lowest priority is carried out at first per cycle.

Cycle: factor specifying the number of OpenXilEnv-cycles after that a process should be called cyclic.

Delay: delayed call

Timeout: (only OpenXilEnv) Timeout after that an external process must log into OpenXilEnv at the latest.

SVLFile: (only OpenXilEnv) SVL-file that should be loaded immediately after the login, empty string when no file. If no

BBPrefix: (only OpenXilEnv) Prefix for all referenced variables , empty string when no prefix.

RangeControlBefore: (only OpenXilEnv) When ≥ 1 , the referenced variable is checked on dropping or exceeding the min./max.-thresholds before calling an external process.

RangeControlBehind: (only OpenXilEnv) When ≥ 1 , the referenced variable is checked on dropping or exceeding the min./max.-thresholds after calling the external process.

RangeCotrolStopScheduler: (only OpenXilEnv) When ≥ 1 , the OpenXilEnv-scheduler is stopped when a dropping or exceeding of the referenced variable occurs.

RangeCotrolOutput: (only OpenXilEnv) Specifies the where the output of the range control is written:

== 0: no output

== 1: into the error-output message-box

== 2: into the file 'script.msg'

RangeCotrolErrorCounter: (only OpenXilEnv) Defines a blackboard-variable where in each appearance of a value range overflow is counted. The data type is UDWORD.

RangeControlVariable: (only OpenXilEnv) Defines a blackboard-variable to switch off the value range control:

== 0: no value range control

≥ 1 : value range control activated

RangeCotrolPysical: (only OpenXilEnv) When ≥ 1 , the value range is physically compared with min.-/max.-thresholds.

RangeControlLimitValues: (only OpenXilEnv) When ≥ 1 , the values are limited to the min.-/max.-thresholds.

Return value:

>0 -> internal process was started (Process-ID)

$=0$ -> external process was started.

<0 -> error

-810 -> process is already running.

10.4.1. XilEnv_StartProcessEx2

C:

```
int XilEnv_StartProcessEx2(const char* ProcessName,
                          int Prio,
                          int Cycle,
                          short Delay,
                          int Timeout,
                          const char *SVLFile,
                          const char *BBPrefix,
                          int UseRangeControl,
                          int RangeControlBeforeActiveFlags,
                          int RangeControlBehindActiveFlags,
                          int RangeControlStopSchedFlag,
                          int RangeControlOutput,
```

```

int RangeErrorCounterFlag,
const char *RangeErrorCounter,
int RangeControlVarFlag,
const char *RangeControl,
int RangeControlPhysFlag,
int RangeControlLimitValues);

```

XilEnv_StartProcessEx starts an internal or external process. Additional to the process name, any other process-relevant setting can be passed as parameter.

ProcessName is the name of the started process. If the name does not correspond to an internal process, an EXE-file of the same name is tried to execute.

Prio: determination of the execution order of processes. The process having the lowest priority is carried out at first per cycle.

Cycle: factor specifying the number of OpenXilEnv-cycles after that a process should be called cyclic.

Delay: delayed call

Timeout: (only OpenXilEnv) Timeout after that an external process must log into OpenXilEnv at the latest.

SVLFile: (only OpenXilEnv) SVL-file that should be loaded immediately after the login, empty string when no file. If no

A2LFile: (only OpenXilEnv) A2L-file that should be loaded immediately after the login, empty string when no file.

A2LFlags: (only OpenXilEnv) If a A2L-file is defined some optionl flags can defined for loading the A2L-file.

```

#define A2L_LINK_NO_FLAGS                0x0
#define A2L_LINK_UPDATE_FLAG             0x1
#define A2L_LINK_UPDATE_IGNORE_FLAG      0x2
#define A2L_LINK_UPDATE_ZERO_FLAG        0x4
#define A2L_LINK_ADDRESS_TRANSLATION_DLL_FLAG 0x8
#define A2L_LINK_ADDRESS_TRANSLATION_MULTI_DLL_FLAG 0x10
#define A2L_LINK_REMEMBER_REFERENCED_LABELS_FLAG 0x20

```

BBPrefix: (only OpenXilEnv) Prefix for all referenced variables , empty string when no prefix.

RangeControlBefore: (only OpenXilEnv) When ≥ 1 , the referenced variable is checked on dropping or exceeding the min.-/max.-thresholds before calling an external process.

RangeControlBehind: (only OpenXilEnv) When ≥ 1 , the referenced variable is checked on dropping or exceeding the min.-/max.-thresholds after calling the external process.

RangeCotrolStopScheduler: (only OpenXilEnv) When ≥ 1 , the OpenXilEnv-scheduler is stopped when a dropping or exceeding of the referenced variable occurs.

RangeCotrolOutput: (only OpenXilEnv) Specifies the where the output of the range control is written:

== 0: no output

== 1: into the error-output message-box

== 2: into the file 'script.msg'

RangeCotrolErrorCounter: (only OpenXilEnv) Defines a blackboard-variable where in each appearance of a value range overflow is counted. The data type is UDWORD.

RangeControlVariable: (only OpenXilEnv) Defines a blackboard-variable to switch off the value range control:

== 0: no value range control

>= 1: value range control activated

RangeCotrolPysical: (only OpenXilEnv) When >= 1, the value range is physically compared with min.-/max.-thresholds.

RangeControlLimitValues: (only OpenXilEnv) When >= 1, the values are limited to the min.-/max.-thresholds.

Return value:

>0 -> internal process was started (Process-ID)

=0 -> external process was started.

<0 -> error

-810 -> process is already running.

10.4.2. XilEnv_StopProcess

C:

```
int XilEnv_StopProcess(unsigned char* name)
```

VB:

```
Public Declare Function XilEnv_StopProcess (ByVal Name As String) As Long
```

XilEnv_StopProcess stops an internal or external process.

Return value:

0 -> Process was ended

!=0 -> error

10.4.3. XilEnv_GetNextProcess

C:

```
char* XilEnv_GetNextProcess(int flag, char* filter);
```

VB:

```
Public Declare Function XilEnv_GetNextProcess (ByVal flag As Long,  
    ByVal filter As String) As String
```

XilEnv_GetNextProcess return the name of the running OpenXilEnv-process (internal or external).

If 'flag' = 1 it is started with the first process, afterwards 'flag' should be set to '0' to read all other processes. A filter can be used by the parameter "filter", e.g. "*" for all.

Return value:

=NULL -> no process found

=String -> process name

10.4.4. XilEnv_GetProcessState

C:

```
int XilEnv_GetProcessState(const char* name);
```

VB:

```
Public Declare Function XilEnv_GetProcessState (ByVal name As String) As Long
```

XilEnv_GetProGessState returns the status of a running OpenXilEnv-process (internal or external).

Return value:

=-808 -> no process found called "name"

=0 -> process initialized and running

=1 -> process sleeping

=2 -> process has „none sleepable" state

=3 -> process has „refence variables" state

=4 -> process has „init" state

=5 -> process has "Terminate" state

10.4.5. XilEnv_AddBeforeProcessEquationFromFile

C:

```
int XilEnv_AddBeforeProcessEquationFromFile(int Nr, const char *ProcName, const char *EquFile);
```

VB:

```
Public Declare Function XilEnv_AddBeforeProcessEquationFromFile (ByVal Nr As Long, ByVal ProcName As String, ByVal EquFile As String) As Long
```

XilEnv_AddBeforeProcessEquationFromFile added an equation block "EquFile" before a process "ProcName". The block number "Nr" can be defined free in the range 1 and above. If "Nr" is 1 a reference the equation file is writte to the INI file. This equation file will be loaded afterward each time the the proccess is started.

Return value:

=-1 -> no process found called "ProcName"

=-2 -> file "EquFile" not found

=-3 -> a line of the equation file "EquFile" are longer as 65535 chars

=-4 -> inside equation file "EquFile" are syntax errors

10.5. XilEnv_AddBehindProcessEquationFromFile

C:

```
int XilEnv_AddBehindProcessEquationFromFile(int Nr, const char *ProcName, const char *EquFile);
```

VB:

```
Public Declare Function XilEnv_AddBehindProcessEquationFromFile (ByVal Nr As Long, ByVal ProcName As String, ByVal EquFile As String) As Long
```

XilEnv_AddBehindProcessEquationFromFile added an equation block "EquFile" behinde a process "ProcName". The block number "Nr" can be defined free in the range 1 and above. If "Nr" is 1 a reference the equation file is writte to the INI file. This equation file will be loaded afterward each time the the proccess is started.

Return value:

=-1 -> no process found called "ProcName"

=-2 -> file "EquFile" not found

=-3 -> a line of the equation file "EquFile" are longer as 65535 chars

=-4 -> inside equation file "EquFile" are syntax errors

10.5.1. XilEnv_DelBeforeProcessEquations

C:

```
int XilEnv_DelBeforeProcessEquations (int Nr, const char *ProcName);
```

VB:

```
Public Declare Function XilEnv_DelBeforeProcessEquations (ByVal Nr As Long,  
    ByVal ProcName As String) As Long
```

XilEnv_DelBeforeProcessEquations delete an equation block which is loaded with the block number "Nr". If the blocknumber is set to -1 all equations are deleted.

10.5.2. XilEnv_DelBehindProcessEquations

C:

```
int XilEnv_DelBehindProcessEquations (int Nr, const char *ProcName);
```

VB:

```
Public Declare Function XilEnv_DelBehindProcessEquations (ByVal Nr As Long,  
    ByVal ProcName As String) As Long
```

XilEnv_DelBehindProcessEquations delete an equation block which is loaded with the block number "Nr". If the blocknumber is set to -1 all equations are deleted.

10.5.3. XilEnv_WaitUntil

C:

```
int XilEnv_WaitUntil (const char \*Equation, int Cycles);
```

The function waits for the equation returning 'TRUE' as result or the time (number of cycles) has passed.

The return value is the rest of cycles (not elapsed cycles).

10.6. Control internal processes

10.6.1. XilEnv_StartScript

C:

```
int XilEnv_StartScript(unsigned char\* scrfile);
```

VB:

```
Public Declare Function XilEnv_StartScript (ByVal scrfile As String) As Long
```

XilEnv_StartScript starts the script-file "scrfile" through the OpenXilEnv-internal script-interpreter. Only one single script may be active at the same time. If OpenXilEnv is used remote-controlled from another PC, the path-specification in "scrfile" must be defined by the view of the OpenXilEnv-PC.

Return value is always '0'.

10.6.2. XilEnv_StopScript

C:

```
int XilEnv_StopScript(void);
```

VB:

```
Public Declare Function XilEnv_StopScript () As Long
```

XilEnv_StopScript stops a running script-file.

10.6.3. XilEnv_StartRecorder

C:

```
int XilEnv_StartRecorder(unsigned char\* cfgfile);
```

VB:

```
Public Declare Function XilEnv_StartRecorder (ByVal cfgfile As String) As Long
```

XilEnv_StartRecorder starts the recorder with the configuration "cfgfile". The Recorder can only be activated once, several parallel running recorders are not possible (same situation for the internal script-interpreter and for manual recordings). If OpenXilEnv is used remote-controlled from another PC, the path-specification in "cfgfile" must be defined by the view of the OpenXilEnv-PC.

XilEnv_StartRecorder returns immediately without waiting for the Recorder to be started (this may take a few cycles).

Return value is always '0'.

10.6.4. XilEnv_StopRecorder**C:**

```
int XilEnv_StopRecorder(void);
```

VB:

```
Public Declare Function XilEnv_StopRecorder () As Long
```

XilEnv_StopRecorder stops an active recording.

10.6.5. XilEnv_StartPlayer**C:**

```
int XilEnv_StartPlayer(unsigned char\* cfgfile)
```

VB:

```
Public Declare Function XilEnv_StartPlayer (ByVal cfgfile As String) As Long
```

XilEnv_StartPlayer starts the Stimuli-player with the configuration "cfgfile".³ The Stimuli-player can only be activated once, several parallel running Stimuli-files are not possible (same situation for the internal script-interpreter and for manual recordings). If OpenXilEnv is used remote-controlled by another PC, the path-specification in "cfgfile" must be defined by the view of the OpenXilEnv-PC.

XilEnv_StartPlayer returns immediately without waiting for the Stimuli-player to be started (this may take a few cycles).

Return value is always '0'.

Visual-Basic example:

Starts a Stimuli-Player and waits for the end of the Stimuli-file.

```
Sub Stimuli()
    Dim Ret As Long
    Dim vid As Long

    vid = XilEnv_AddVari("HD_Play", 1, "")
    Ret = XilEnv_StartGenerator("c:\schrott\dummy.cfg") ' Wait until the player is
running

    Do Until XilEnv_Get(vid) = 2
        Ret = DoEvents()
    Loop ' Wait until Stimuli is ended

    Do Until XilEnv_Get(vid) = 0
        Ret = DoEvents()
    Loop
End Sub
```

10.6.6. XilEnv_StopPlayer

C:

```
int XilEnv_StopPlayer(void);
```

VB:

```
Public Declare Function XilEnv_StopPlayer ()
```

XilEnv_StopPlayer stops an active recording.

10.6.7. XilEnv_StartEquations

C:

```
int XilEnv_StartEquations(unsigned char\* equfile)
```

VB:

```
Public Declare Function XilEnv_StartEquations (ByVal equfile As String) As Long
```

XilEnv_StartEquations starts a OpenXilEnv-equation block. The formula calculator can only be activated once at a moment, several running Stimuli-files are not possible (same situation for the internal script-interpreter and manual started Stimulies). If OpenXilEnv is used remote-controlled by another PC, the path-specification in "equfile" must be defined by the view of the OpenXilEnv-PC.

XilEnv_StartEquations returns immediately without waiting for the Stimuli-Player to be started (this may take a few cycles).

Return value is always '0'.

10.6.8. XilEnv_StopEquations**C:**

```
int XilEnv_StopEquations(void);
```

VB:

```
Public Declare Function XilEnv_StopEquations ()
```

XilEnv_StopEquations stops the cyclic execution of the calculator.

10.6.9. XilEnv_StartGenerator**C:**

```
int XilEnv_startGenerator(unsigned char* genfile);
```

VB:

```
Public Declare Function XilEnv_StartGenerator (ByVal genfile As String) As Long
```

XilEnv_StartGenerator starts the Ramp-generator with the configuration "genfile". The generator can only be activated once at a moment, several running generators are not possible (same situation for the internal script-interpreter and for manual recordings). If OpenXilEnv is used remote-controlled by another PC, the path-specification in "genfile" must be defined by the view of the OpenXilEnv-PC.

XilEnv_StartGenerator returns immediately without waiting for the ramps to be starte (may take a few cycles).

Return value is always '0'.

Visual-Basic example:

Starts the ramp-generator and waits until all ramps are ended.

```
Sub Rampe()
    Dim Ret As Long
    Dim vid As Long

    vid = XilEnv_AddVari("Generator", 1, "")
    Ret = XilEnv_StartGenerator("c:\schrott\dummy.gen")      ' Wait until ramp runs

    Do Until XilEnv_Get(vid) = 2
        Ret = DoEvents()
    Loop                                                    ' Wait until ramp is ended

    Do Until XilEnv_Get(vid) = 0
        Ret = DoEvents()
    Loop
End Sub
```

10.6.10. XilEnv_StopGenerator

C:

```
int XilEnv_StopGenerator(void);
```

VB:

```
Public Declare Function XilEnv_StopGenerator ()
```

XilEnv_StopGenerator stops all ramp calculations.

10.7. Control the user interface

10.7.1. XilEnv_LoadDsktop

C:

```
int XilEnv_LoadDsktop(unsigned char\* file);
```

VB:

```
Public Declare Function XilEnv_LoadDesktop (ByVal file As String) As Long
```

XilEnv_LoadDesktop closes and removes all OpenXilEnv-windows, afterwards all windows that are defined in the desktop-file are loaded and opened. If OpenXilEnv is used remote-controlled from another PC, the path-specification in "file" must be defined by the view of the OpenXilEnv-PC.

Return value:

==0 -> everything OK

!=0 -> Error

10.7.2. XilEnv_SaveDesktop**C:**

```
int XilEnv_SaveDesktop(unsigned char\* file);
```

VB:

```
Public Declare Function XilEnv_SaveDesktop (ByVal file As String) As Long
```

XilEnv_SaveDesktop saves the current desktop in "file". If OpenXilEnv is remote-controlled from another PC, the path-specification "file" must be defined by the view of the OpenXilEnv-PC.

Return value:

==0 -> everything OK

!=0 -> Error

10.7.3. XilEnv_CreateDialog**C:**

```
int XilEnv_CreateDialog(const char\* DialogName);
```

VB:

```
Public Declare Function XilEnv_CreateDialog (ByVal DialogName As String) As Long
```

XilEnv_CreateDialog creates a OpenXilEnv-dialogue for the query of values from a blackboard variable.

Query-elements can be added to the dialogue by using 'XilEnv_AddDialogItem'. The string „DialogName" is entered into the window bar of the dialogue. Attention: The dialogue is not visible until the intruction 'XilEnv_ShowDialog' was executed.

Return value

==0 -> everything OK

!=0 -> Error

10.7.4. XilEnv_AddDialogItem

C:

```
int XilEnv_AddDialogItem(const char\* Description, const char \*VariName);
```

VB:

```
Public Declare Function XilEnv_AddDialogItem (ByVal Description As String, ByVal  
VariName As String) As Long
```

XilEnv_AddDialogItem inserts an input-element for values of blackboard variables to a OpenXilEnv-dialogue. "Description" specifies the description of the input-element and "VariName" specifies the blackboard variable whose value should be changed.

By using 'XilEnv_CreateDialog' a dialogue must be created previously. Attention: The dialogue is not visible until the instruction 'XilEnv_CreateDialog' was executed.

Return value:

==0 -> everything OK

!=0 -> Error

10.7.5. XilEnv_ShowDialog

C:

```
int XilEnv_ShowDialog(void);
```

VB:

```
Public Declare Function XilEnv_ShowDialog () As Long
```


XilEnv_ShowDialog opens the dialogue of the that was created previously by 'XilEnv_CreateDialogue' and 'XilEnv_AddDialogItem'. The dialogue is opened until the user closes it.

Attention: the function returns immediately without waiting for the dialogue to be closed.

By using 'XilEnv_IsDialogClosed' it can be checked if the dialogue was closed.

Return value:

=0 -> everything OK

!=0 -> Error

10.7.6. XilEnv_IsDialogClosed

C:

```
int XilEnv_IsDialogClosed(void);
```

VB:

```
Public Declare Function XilEnv_IsDialogClosed () As Long
```

'XilEnv_DialogClosed' checks if the dialogue is opened (return- value = 0) or closed (return-value = 1)

Return value:

=0 -> Dialogue is open (visible)

=1 -> Dialogue was closed by the user

10.7.7. XilEnv_SelectSheet

C:

```
int XilEnv_SelectSheet (const char\* SheetName);
```

Switches the display to the sheet "SheetName".

Return value:

=0 -> Sheet was switched

=-1 -> Sheet-name unknown

10.7.8. XilEnv_AddSheet

C:

```
int XilEnv_AddSheet (const char* SheetName);
```

Adds a new sheet „SheetName" to the display-

Return value:

=0 -> Sheet was added

=-1 -> Sheet Name already exists

10.7.9. XilEnv_DeleteSheet

C:

```
int XilEnv_DeleteSheet (const char* SheetName);
```

Deletes an existing sheet called "SheetName".

Return value:

=0 -> Sheet was deleted

=-1 -> no sheet existing having this name

10.7.10. XilEnv_RenameSheet

C:

```
int XilEnv_RenameSheet (const char* OldSheetName, const char* NewSheetName);
```

Changes the name of an already existing sheet from "OldSheetName" into "NewSheetName".

Return value:

0 -> Sheet was renamed

-1 -> no sheet existing having this name

10.7.11. XilEnv_OpenWindow

C:

```
int XilEnv_OpenWindow (const char* WindowName);
```

Opens a display-window called "WindowName". The window must be existing, that means it was opened once before. Wildcards (*,?) are allowed.

Return value:

Always '0'.

10.7.12. XilEnv_CloseWindow

C:

```
int XilEnv_CloseWindow (const char* WindowName);
```

Closes an opened display-window „WindowName". Wildcards (*,?) are allowed.

Return value:

Always '0'.

10.7.13. XilEnv_DeleteWindow

C:

```
int XilEnv_DeleteWindow (const char* WindowName);
```

Deletes a display-window called „WindowName" from the INI-file. If the window is opened, it will be closed before deleting it. Wildcards (*,?) are allowed.

Return value:

Always '0'.

10.7.14. XilEnv_ImportWindow

C:

```
int XilEnv_ImportWindow (const char* WindowName, const char* FileName);
```

Imports windows from the file „FileName" and opens them in the current sheet. Wildcards (*,?) are allowed.

Return value:

0-> Windows were imported

-1: -> File could not be read

10.7.15. XilEnv_ExportWindow

C:

```
int XilEnv_ExportWindow (const char\* SheetName, const char\* WindowName, const char\* FileName);
```

Exports windows into the file „FileName“. Wildcards (*,?) are allowed.

Return value:

Always '0'.

10.8. Blackboard

10.8.1. XilEnv_AddVari

C:

```
int XilEnv_AddVari(unsigned char\* label, int type, unsigned char\* unit)
```

VB:

```
Public Declare Function XilEnv_AddVari (ByVal label As String, ByVal ltype As Long, ByVal unit As String) As Long
```

XilEnv_AddVari registers a variable "label" with "unit" in the blackboard. The data type is specified with "type".

Following types are allowed:

Count	Type
0	Signed Byte (signed char)
1	Unsigned Byte (unsigned char)
2	Signed Word (signed short)
3	Unsigned Word (unsigned short)
4	Signed Double Word (signed long)
5	Unsigned Double Word (unsigned long)
6	32bit Float (float)
7	64bit Float (double)
8	Unknown; variable must be registered in the blackboard
9	Unknown if the variable is existing in the blackboard, otherwise: 64bit Float

Count	Type
34	Signed Quadruple Word (signed long long)
35	Unsigned Quadruple Word (unsigned long long)

Return value:

>0 -> Variable-identifier for later read-/write access

<0 -> Error: variable could not be registered.

10.8.2. XilEnv_RemoveVari

C:

```
int XilEnv_RemoveVari(int vid);
```

VB:

```
Public Declare Function XilEnv_RemoveVari Lib (ByVal vid As Long) As Long
```

XilEnv_RemoveVari removes a variable referenced through "vid" from the blackboard. The variable has to be created by 'XilEnv_AddVari' previously. The variable can be removed only when no other process uses this variable. The number of XilEnv_RemoveVari must be equal to the number of XilEnv_AddVari.

Example:

```
Dim vid As Long

vid = XilEnv_AddVari("test", 7, "_A")
vid = XilEnv_AddVari("test", 7, "mA")
vid = XilEnv_AddVari("test", 7, "mA")
XilEnv_RemoveVari(vid);
XilEnv_RemoveVari(vid);
XilEnv_RemoveVari(vid);           '←- here the variable is removed
```

10.8.3. XilEnv_AttachVari

C:

```
int XilEnv_AttachVari(unsigned char\* label);
```

VB:

```
Public Declare Function XilEnv_AttachVari Lib (ByVal label As String) As Long
```

XilEnv_AttachVari increases the access-counter of a variable and returns its variable-ID.

Example:

```
Dim vid As Long

vid = XilEnv_AddVari("test", 7e "mA")
vid = XilEnv_AttachVari("test")
vid = XilEnv_AttachVari("test")
XilEnv_RemoveVari(vid);
XilEnv_RemoveVari(vid);
XilEnv_RemoveVari(vid);           '<- here the variable is deleted
```

10.8.4. XilEnv_Get

C:

```
double XilEnv_Get(int vid);
```

VB:

```
Public Declare Function XilEnv_Get Lib (ByVal vid As Long) As Double
```

XilEnv_Get returns the current value of the variable, referenced through "vid".

Example:

```
Dim vid As Long
Dim value As Double

vid = XilEnv_AddVari("test", 7, "mA")
value = XilEnv_Get(vid);
```

10.8.5. XilEnv_GetPhys

C:

```
double XilEnv_GetPhys(int vid);
```

VB:

```
Public Declare Function XilEnv_GetPhys Lib (ByVal vid As Long) As Double
```

XilEnv_GetPhys returns the current physical value (converted) of the variable, referenced through "vid". There must be a conversion formula specified for this variable in the blackboard.

10.8.6. XilEnv_Set

C:

```
void XilEnv_Set(int vid, double value);
```

VB:

```
Public Declare Sub XilEnv_Set (ByVal vid As Long, ByVal value As Double)
```

XilEnv_Set writes the "value" into the "vid"-referenced variable.

10.8.7. XilEnv_SetPhys

C:

```
int XilEnv_SetPhys(int vid, double value);
```

VB:

```
Public Declare Function XilEnv_SetPhys (ByVal vid As Long, ByVal value As Double)  
As Long
```

XilEnv_SetPhys converts "value" from physical into decimal and writes the result into the "vid"-referenced variable. There must be a linear conversion formula specified for this variable in the blackboard.

Return value:

==0 -> OK

!=0 -> Error (conversion formula is not-linear, e.g.: 1/#)

10.8.8. XilEnv_Equ

C:

```
double XilEnv_Equ(unsigned char\* equ);
```

VB:

```
Public Declare Function XilEnv_Equ (ByVal equ As String) As Double
```

XilEnv_Equ calculates the formula "equ" and returns the result. The formula can contain any blackboard variable.

E.g.: $\text{Signal1} * 10.0 + (\text{Signal2} - \text{Signal3}) * \text{Offset}$

10.8.9. XilEnv_WrVariEnable

C:

```
int XilEnv_WrVariEnable(unsigned char\* label, unsigned char\* process);
```

VB:

```
Public Declare Function XilEnv_WrVariEnable (ByVal label As String, ByVal process As String) As Long
```

XilEnv_WrVariEnable allows the writing of a process "process" into a blackboard variable "label", that was blocked by 'XilEnv_WrVariDisable' previously.

Return value:

$\text{==0} \rightarrow \text{OK}$

$\text{!=0} \rightarrow \text{Error}$

10.8.10. XilEnv_WrVariDisable

C:

```
int XilEnv_WrVariDisable(unsigned char\* label, unsigned char\*process);
```

VB:


```
Public Declare Function XilEnv_WrVariDisable (ByVal label As String, ByVal process As String) As Long
```

XilEnv_WrVariDisable blocks the writing of a process "process" into a blackboard-variable "label".

Return value:

==0 -> OK

!=0 -> Error

10.8.11. XilEnv_IsWrVariEnable

C:

```
int XilEnv_IsWrVariEnabled(unsigned char* label, unsigned char* process);
```

VB:

```
Public Declare Function XilEnv_IsWrVariEnabled (ByVal label As String, ByVal process As String) As Long
```

XilEnv_WrVariDisable blocks the writing of a process "process" into a blackboard-variable "label".

Return value:

==0 -> OK

!=0 -> Error

10.8.12. XilEnv_LoadRefList

C:

```
int XilEnv_LoadRefList(unsigned char* reflist, unsigned char* process);
```

VB:

```
Public Declare Function XilEnv_LoadRefList (ByVal reflist As String, ByVal process As String) As Long
```

XilEnv_LoadRefList loads a reference list "reflist" of an external process "process".

If OpenXilEnv is remoted-controlled by another PC, the path-specification in "reflist" must be defined by the view of the OpenXilEnv-PC.

Return value:

==0 -> everything OK

!=0 -> Error

10.8.13. XilEnv_SaveRefList

C:

```
int XilEnv_SaveRefList(unsigned char* reflist, unsigned char* process);
```

VB:

```
Public Declare Function XilEnv_SaveRefList (ByVal reflist As String, ByVal process As String) As Long
```

XilEnv_SaveRefList saves the current reference list "reflist" of an external process "process".

If OpenXilEnv is remoted-controlled by another PC, the path-specification in "reflist" must be defined by the view of the OpenXilEnv-PC.

Return value:

==0 -> everything OK

!=0 -> Error

10.8.14. XilEnv_ExportRobFile

C:

```
int XilEnv_ExpoCtRobFile(unsigned char* robfile);
```

VB:

```
Public Declare Function XilEnv_ExportRobFile (ByVal robfile As String) As Long
```

XilEnv_ExportRobFile exports a ROB-/ASAP-file of all blackboard-variables. If the "robfile" ends with *.rob, a ROB-file is created, if the ending is *.a2l, a ASAP-file is created. If OpenXilEnv is remoted-controlled by another PC, the path-specification in "reflist" must be defined by the view of the OpenXilEnv-PC.

Return value:

==0 -> everything OK

!=0 -> Error

10.8.15. XilEnv_GetVariConversionType

C:

```
int XilEnv_GetVariConversionType(int vid);
```

VB:

```
Public Declare Function XilEnv_GetVariConversionType (ByVal vid As Long) As Long
```

XilEnv_GetVariConversionType returns the conversion-type of a blackboard-variable. Return value:

< 0 -> Error, variable does not exist

==0 -> no conversion defined

==1 -> conversion defined

==2 -> an enum text replacement is defined

10.8.16. XilEnv_GetVariConversionString

C:

```
char\* XilEnv_GetVariConversionString(int vid);
```

VB:

```
Public Declare Function XilEnv_GetVariConversionString (ByVal vid As Long) As String
```

XilEnv_GetVariConversionString returns the conversion-string of a blackboard variable. Return value:

==NULL -> error, variable does not exist or no conversion is defined

==String -> conversion-String e.g.: „2*#+100“

10.8.17. XilEnv_SetVariConversion

C:

```
int XilEnv_SetVariConversion (int vid, int type, const char conv_string);
```

VB:

```
Public Declare Function XilEnv_SetVariConversion (ByVal vid As Long, ByVal type As Long, ByVal conv_string As String) As Long
```

XilEnv_SetVariConversion sets the conversion of a blackboard-variable.

Return value:

!= 0 -> error, variable does not exist

==0 -> OK

10.8.18. XilEnv_GetVariType

C:

```
int XilEnv_GetVariType(int vid);
```

VB:

```
Public Declare Function XilEnv_GetVariType (ByVal vid As Long) As Long
```

XilEnv_GetVariType returns the current data type of a blackboard-variable.

Return value:

< 0 -> error, variable does not exist

== 0 -> Signed Byte (signed char)

== 1 -> Unsigned Byte (unsigned char)

== 2 -> Signed Word (signed short)

== 3 -> Unsigned Word (unsigned short)

== 4 -> Signed Double Word (signed long)

== 5 -> Unsigned Double Word (unsigned long)

== 6 -> 32bit Float (float)

== 7 -> 64bit Float (double)

== 8 -> Unknown variable has to be registered in the blackboard.

10.8.19. XilEnv_GetVariUnit

C:

```
char\* XilEnv_GetVariUnit(int vid);
```

VB:

```
Public Declare Function XilEnv_GetVariUnit (ByVal vid As Long) As String
```

XilEnv_GetVariUnit returns the unit of a blackboard-variable.

Return value:

!= 0 -> error, variable does not exist

== 0 -> OK

10.8.20. XilEnv_SetVariUnit

C:

```
char\* XilEnv_SetVariUnit(int vid, char \*unit);
```

VB:

```
Public Declare Function XilEnv_SetVariUnit (ByVal vid As Long ByVal unit As String) As Long
```

XilEnv_SetVariUnit sets the unit of a blackboard-variable „unit“

Return value:

!= 0 -> Error, variable does not exist

== 0 -> OK

10.8.21. XilEnv_GetVariMin

C:

```
double XilEnv_GetVariMin(int vid);
```

VB:

```
Public Declare Function XilEnv_GetVariMin (ByVal vid As Long) As Double
```

XilEnv_GetVariMin returns the minimum threshold of a blackboard-variable.

10.8.22. XilEnv_GetVariMax

C:

```
double XilEnv_GetVariMax(int vid);
```

VB:

```
Public Declare Function XilEnv_GetVariMax (ByVal vid As Long) As Double
```

XilEnv_GetVariMax returns the maximum threshold of a blackboard-variable.

10.9. XilEnv_SetVariMin

C:

```
int XilEnv_SetVariMin(int vid, double min);
```

VB:

```
Public Declare Function XilEnv_SetVariMin (ByVal vid As Long, ByVal min As Double)  
As Long
```

XilEnv_SetVariMin sets the minimum threshold of a blackboard-variable.

Return value:

!= 0 -> error, variable does not exist

== 0 -> OK

10.9.1. XilEnv_SetVariMax

C:

```
int XilEnv_SetVariMax(int vid, double max);
```

VB:

```
Public Declare Function XilEnv_SetVariMax (ByVal vid As Long, ByVal max As Double)
As Long
```

XilEnv_SetVariMax sets the maximum threshold of a blackboard-variable.

Return value:

!= 0 -> error, variable does not exist

== 0 -> OK

10.9.2. XilEnv_GetNextVari

C:

```
char\* XilEnv_GetNextVari(int flag, char\* filter);
```

VB:

```
Public Declare Function XilEnv_GetNextVari (ByVal flag As Long, ByVal filter As
String) As String
```

XilEnv_GetNextVari returns the name of a blackboard-variable.

If flag = 1 it will be started with the first variable, afterwards flag should be set to '0', to read all other variables. A filter can be specified with "filter", e.g.: „*" for all.

Return value:

=NULL -> no (other) variable found

=String -> variable name

10.9.3. XilEnv_GetNextVariEx

C:

```
char\* XilEnv_GetNextVariEx(int flag, char\* filter, char\*process, int
AccessFlags);
```

VB:

```
Public Declare Function XilEnv_GetNextVariEx (ByVal flag As Long, ByVal filter As
String, , ByVal process As String, ByVal AccessFlags As Long) As String
```

XilEnv_GetNextVariEx returns the name of a blackboard-variable.

If flag = 1 it will be started with the first variable, afterwards flag should be set to '0', to read all other variables. A filter can be specified with "filter", e.g.: „*" for all.

Only the variables of the process "process" with the following access rights will be found:

AccessFlags = 0 -> Process "process" has write access to that variable

AccessFlags = 1 -> Process "process" has no write access to that variable

AccessFlags = 2 -> Process "process" has access (write or no write) to that variable

All others of AccessFlags -> no variable will be found

Return value:

=NULL -> no (other) variable found

=String -> variable name

10.9.4. XilEnv_GetVariEnum**C:**

```
char\* XilEnv_GetVariEnum(int vid, double value);
```

VB:

```
Public Declare Function XilEnv_GetVariEnum (ByVal vid As Long, ByVal value As
double) As String
```

XilEnv_GetVariEnum returns the text replacement (enum) that was defined for the "value".

Return value:

=NULL -> Variable not found

=String -> Text replacement (enum)

10.9.5. XilEnv_WriteFrame

C:

```
int XilEnv_WriteFrame (int \*Vids, double \*ValueFrame, int Size);
```

XilEnv_WriteFrame writes an array of variables into the blackboard. Thereby it is ensured that all variables are written in one cycle without an interrupt. This is also much faster than single writing through 'XilEnv_Set'.

Return value:

=-1 -> no connection to OpenXilEnv(RT)

=0 -> OK

Python example:

```
Vids = sc2py.VidsArray(3)
```

```
Vids[0] = sc2py.AddVari ("Variable1", 0, "")
```

```
Vids[1] = sc2py.AddVari ("Variable2", 0, "")
```

```
Vids[2] = sc2py.AddVari ("Variable3", 0, "")
```

```
Values = sc2py.ValuesArray(3)
```

```
Values[0] = 1.0
```

```
Values[1] = 2.0
```

```
Values[2] = 3.0
```

```
sc2py.WriteFrame (Vids, Values, 3)
```

10.9.6. XilEnv_GetFrame

C:

```
int XilEnv_GetFrame (int \*Vids, double \*RetValFrame, int Size);
```

XilEnv_GetFrame reads an array of variables from the blackboard at once. Thereby it is ensured that all variables are read in one cycle without an interrupt. This is much faster than single reading through 'XilEnv_Get'.

Return value:

=-1 -> no connection to OpenXilEnv(RT)

=0 -> OK

Python example:

```
Vids = sc2py.VidsArray(3)

Vids[0] = sc2py.AddVari ("Variable1", 0, "")

Vids[1] = sc2py.AddVari ("Variable2", 0, "")

Vids[2] = sc2py.AddVari ("Variable3", 0, "")

Values = sc2py.ValuesArray(3)

sc2py.WriteFrame (Vids, Values, 3)

print Values[0]

print Values[1]

print Values[2]
```

10.9.7. XilEnv_WriteFrameWaitReadFrame

C:

```
int XilEnv_WriteFrameWaitReadFrame (int \*WriteVids, double \*WriteValues, int
WriteSize, int \*ReadVids, double \*ReadValuesRet, int ReadSize);
```

XilEnv_WriteFrameWaitReadFrame writes an array of variables into the blackboard. Thereby it is ensured that all variables are written in one cycle without an interrupt. At OpenXilEnv, the Scheduler is started for one cycle, which requires that the Scheduler was stopped before. At OpenXilEnv for HiL, the function waits until at least one Scheduler-cycle was performed. Afterwards an array of variables is read from the blackboard. Thereby it is ensured that all variables are read in one cycle without interrupt.

Return value:

=-1 -> no connection to OpenXilEnv(RT)

=0 -> OK

Python example:

```
VidsOut = sc2py.VidsArray(3)

VidsOut[0] = sc2py.AddVari ("Out1", 0, "")

VidsOut[1] = sc2py.AddVari ("Out2", 0, "")

VidsOut[2] = sc2py.AddVari ("Out3", 0, "")

Out = sc2py.ValuesArray(3)
```

```

Out[0] = 1.0

Out[1] = 2.0

Out[2] = 3.0

VidsIn = sc2py.VidsArray(2)

VidsIn[0] = sc2py.AddVari ("In1", 0, "")

VidsIn[1] = sc2py.AddVari ("In2", 0, "")

In = sc2py.ValuesArray(2)

sc2py.WriteFrameWaitReadFrame (VidsOut, Out, 3, VidsIn, In, 2)

print In[0]

print In[1]

```

10.9.8. XilEnv_ImportVariProperties

C:

```
int XilEnv_ImportVariProperties (const char\* Filename);
```

Imports the settings of all blackboard-variables that are included in the file "Filename".

Thereby the following settings are overwritten: unit, min-/max-thresholds, conversion, description, step size and default color.

10.9.9. XilEnv_EnableRangeControl

C:

```
int XilEnv_EnableRangeControl (const char\* ProcessNameFilter, const char\*
VariableNameFilter);
```

This instruction switches on the range monitoring of a process regarding certain variables. The range monitoring is switched on at process-start by default.

Wildcards (*) are allowed for process name as well as variabels.

10.9.10. XilEnv_DisableRangeControl

C:

```
int XilEnv_DisableRangeControl (const char\* ProcessNameFilter, const char\*
VariableNameFilter);
```

This instruction switches off the range monitoring of a process regarding certain variables. The range monitoring is switched on at process-start by default.

Wildcards (*) are allowed for process name as well as variables.

10.9.11. XilEnv_GetRaw

C:

```
enum BB_DATA_TYPES XilEnv_GetRaw(int vid, union BB_VARI \*ret_Value);
```

'XilEnv_GetRaw' get a raw value of a blackboard variable with the identifier 'vid'. The fetched value are stored at the address given with the parameter 'ret_Value'. The pointer 'ret_Value' must be reference a memory object equal or larger 8 bytes so all possible base data types can be stored.

```
union BB_VARI {
    int8_t b;           // BB_BYTE
    uint8_t ub;         // BB_UBYTE
    int16_t w;          // BB_WORD
    uint16_t uw;        // BB_UWORD
    int32_t dw;         // BB_DWORD
    uint32_t udw;       // BB_UDWORD
    int64_t qw;         // BB_QWORD
    uint64_t uqw;       // BB_UQWORD
    float f;           // BB_FLOAT
    double d;          // BB_DOUBLE
};
```

Return value

>=0 - everything is OK, the value represent the data type of the fetched value.

```
enum BB_DATA_TYPES {BB_BYTE=0, BB_UBYTE=1, BB_WORD=2, BB_UWORD=3, BB_DWORD=4,
                    BB_UDWORD=5, BB_FLOAT=6, BB_DOUBLE=7, BB_UNKNOWN=8,
                    BB_UNKNOWN_DOUBLE=9, BB_UNKNOWN_WAIT=10, BB_QWORD=34,
                    BB_UQWORD=35};
```

<0 - error

10.9.12. XilEnv_SetRaw

C:

```
int XilEnv_SetRaw(int vid, enum BB_DATA_TYPES Type, union BB_VARI Value, int
Flags);
```

'XilEnv_SetRaw' will set the raw value of a blackboard variable with the identifier 'vid'. The parameter 'Type' give the data type of the raw value to set. The parameter 'value' must be set with the correct data type match to the value of the parameter 'Type'.

```
enum BB_DATA_TYPES {BB_BYTE=0, BB_UBYTE=1, BB_WORD=2, BB_UWORD=3, BB_DWORD=4,
                    BB_UDWORD=5, BB_FLOAT=6, BB_DOUBLE=7, BB_UNKNOWN=8,
                    BB_UNKNOWN_DOUBLE=9, BB_UNKNOWN_WAIT=10, BB_QWORD=34,
                    BB_UQWORD=35};

union BB_VARI {
    int8_t b;           // BB_BYTE
    uint8_t ub;         // BB_UBYTE
    int16_t w;          // BB_WORD
    uint16_t uw;        // BB_UWORD
    int32_t dw;         // BB_DWORD
    uint32_t udw;       // BB_UDWORD
    int64_t qw;         // BB_QWORD
    uint64_t uqw;       // BB_UQWORD
    float f;            // BB_FLOAT
    double d;           // BB_DOUBLE
};
```

Remark: OpenXilEnv will check the match of the 'Type' parameter and the data type of the blackboard variable. It will convert if necessary.

Return value

>=0 - everything is OK.

<0 - error

10.10. Calibration

10.10.1. XilEnv_LoadSvl

C:

```
int XilEnv_LoadSvl(unsigned char\* svlfile, unsigned char\* process);
```

VB:

```
Public Declare Function XilEnv_LoadSvl (ByVal svlfile As String, ByVal process As String) As Long
```

'XilEnv_LoadSvl' loads a SVL-file "svlfile" into an external process "process". A description of the SVL-format (**Softcar Value List**). If OpenXilEnv is remote-controlled from another PC, the path-specification in the "svlfile" must be defined by the view of the OpenXilEnv-PC.

Return value:

==1 -> everything OK

!=1 -> error

10.10.2. XilEnv_SaveSvl**C:**

```
int XilEnv_SaveSvl(unsigned char\* svlfile, unsigned char\* process, unsigned char\* filter);
```

VB:

```
Public Declare Function XilEnv_SaveSvl (ByVal svlfile As String, ByVal process As String, ByVal filter As String) As Long
```

'XilEnv_SaveSvl' saves global data of an external process "process" in a SVL-file "svlfile".

Thereby the use of a filter is permitted. E.g.: "*" for all labels or "KL_*" for labels which start with "KL_". If OpenXilEnv is remote-controlled from another PC, the path-specification in the "svlfile" must be defined by the view of the OpenXilEnv-PC.

Return value:

==1 -> everything OK

!=1 -> error

10.10.3. XilEnv_ReferenceSymbol**C:**

```
int XilEnv_ReferenceSymbol(const char\* Symbol, const char \*DisplayName, const char\* Process, const char \*Unit, int ConversionType, const char \*Conversion,
```

```
double Min, double Max, int Color, int Width, int Precision, int Flags);
```

'XiEnv_ReferenceSymbol' will be referenced a symbol 'Symbol' inside an external process 'Process' to the blackboard. The symbol name must be a entry inside the debug information of the externen process. The symbol is a path to a static memory location inside the external process. For example:

```
Label1.member.array[3].member
```

The symbol must be a base data type. If the 'DisplayName' is not an empty string or a null pointer this will define the name inside the blackboard, otherwise it is the symbol itself. The parameter 'Unit' sets the unit string inside the blackboard. It can be an emty string or a null pointer. Than the unit would be not changed or will be empty. The parameter 'ConversionType' define the type of the display conversion. It can have the following values:

```
/* no conversion 1:1 */
#define CONVTYPE_NOTHING      0
/* a equation as conversion for example "10.0*#+1000.0" */
#define CONVTYPE_EQUATION     1
/* an enum conversion for example "0 0 \"off\"; 1 1 \"on\";" */
#define CONVTYPE_TEXTREPLACE  2
/* do not change the conversion */
#define CONVTYPE_UNDEFINED    255
```

The parameter 'Conversion' define the conversion string of the display conversion. this cn be a formula or a text replacement definition.

The parameter 'Min' and 'Max' define the range of the variable. If this should not be set NAN (not a number) can be used. The parameter 'Color' defines the default color of the variable. It must a RGB value or COLOR_UNDEFINED.

The parameter 'Width' and 'Precison' defines the display format representation. 'Width' defines the whole lenght and 'Precision' the position after decimal point.

The parameter 'Flags' defines the direction of the reference.

```
/* BB -> extern process */
#define REFERENCE_SYMBOL_BB2EP_FLAG      0x0001
/* extern process -> BB */
#define REFERENCE_SYMBOL_EP2BB_FLAG      0x0002
/* BB <-> extern process */
#define REFERENCE_SYMBOL_READWRITE_FLAG  0x0003
#define REFERENCE_SYMBOL_ADD_TO_LIST     0x0004
```

Return value

>0 - everything is OK, the value are the variable id, this can be used to access the value of this reference inside the blackboard.

<0 - error

10.10.4. XilEnv_DereferenceSymbol

C:

```
int XilEnv_DereferenceSymbol(const char* Symbol, const char* Process, int
Flags);
```

'XilEnv_ReferenceSymbol' will be referenced a symbol 'Symbol' inside an external process 'Process' to the blackboard. The symbol name must be a entry inside the debug information of the external process. The symbol is a path to a static memory location inside the external process. For example:

```
Label1.member.array[3].member
```

The symbol must be a base data type.

The parameter 'Flags' can have following value or 0.

```
#define DEREFERENCE_SYMBOL_REMOVE_FROM_LIST_FLAG 0x0004
```

Return value

>0 - everything is OK.

<0 - error

10.10.5. XilEnv_GetSymbolRaw

C:

```
enum _\_BB_DATA_TYPES XilEnv_GetSymbolRaw(const char* Symbol, const char*
Process, int Flags, union BB_VARI *ret_Value);
```

'XilEnv_GetSymbolRaw' get a raw value of a symbol 'Symbol' from an external process 'Process'. The symbol name must be a entry inside the debug information of the external process. The symbol is a path to a static memory location inside the external process. For example:

```
Label1.member.array[3].member
```


The symbol must be a base data type. The fetched value are stored at the address given with the parameter 'ret_Value'. The pointer 'ret_Value' must be reference a memory object equal or larger 8 bytes so all possible base data types can be stored.

```
union BB_VARI {
    int8_t b;           // BB_BYTE
    uint8_t ub;         // BB_UBYTE
    int16_t w;          // BB_WORD
    uint16_t uw;        // BB_UWORD
    int32_t dw;         // BB_DWORD
    uint32_t udw;       // BB_UDWORD
    int64_t qw;         // BB_QWORD
    uint64_t uqw;       // BB_UQWORD
    float f;            // BB_FLOAT
    double d;           // BB_DOUBLE
};
```

The parameter 'Flags' should be always 0.

Return value

>=0 - everything is OK, the value represent the data type of the fetched value.

```
enum BB_DATA_TYPES {BB_BYTE=0, BB_UBYTE=1, BB_WORD=2, BB_UWORD=3, BB_DWORD=4,
                    BB_UDWORD=5, BB_FLOAT=6, BB_DOUBLE=7, BB_UNKNOWN=8,
                    BB_UNKNOWN_DOUBLE=9, BB_UNKNOWN_WAIT=10, BB_QWORD=34,
                    BB_UQWORD=35};
```

<0 - error

10.10.6. XilEnv_SetSymbolRaw

C:

```
int XilEnv_SetSymbolRaw(const char* Symbol, const char* Process, int Flags,enum
    \_BB_DATA_TYPES DataType, union BB_VARI Value);
```

'XilEnv_GetSymbolRaw' will set a raw value of a symbol 'Symbol' inside an external process 'Process'. The symbol name must be a entry inside the debug information of the external process. The symbol should be a path to a static memory location inside the external process. For example:

```
Label1.member.array[3].member
```

The symbol must be a base data type. The value to store at the given symbol address are defined with the parameter 'DataType and Value'.

```
enum BB_DATA_TYPES {BB_BYTE=0, BB_UBYTE=1, BB_WORD=2, BB_UWORD=3, BB_DWORD=4,
                    BB_UDWORD=5, BB_FLOAT=6, BB_DOUBLE=7, BB_UNKNOWN=8,
                    BB_UNKNOWN_DOUBLE=9, BB_UNKNOWN_WAIT=10, BB_QWORD=34,
                    BB_UQWORD=35};

union BB_VARI {
    int8_t b;           // BB_BYTE
    uint8_t ub;         // BB_UBYTE
    int16_t w;          // BB_WORD
    uint16_t uw;        // BB_UWORD
    int32_t dw;         // BB_DWORD
    uint32_t udw;       // BB_UDWORD
    int64_t qw;         // BB_QWORD
    uint64_t uqw;       // BB_UQWORD
    float f;           // BB_FLOAT
    double d;          // BB_DOUBLE
};
```

The parameter 'Flags' should be always 0.

Return value

==0 - everything is OK, the value represent the data type of the fetched value.

<0 - error

10.11. Calibration with A2L file

10.11.1. XilEnv_SetupLinkToExternProcess

C:

```
int XilEnv_SetupLinkToExternProcess(const char *A2LFileName, const char
    *ProcessName, int UpdateFlag);
```

'XilEnv_SetupLinkToExternProcess' will be setup a link between an external process and an A2L file. The parameter 'A2LFileName' will define the A2L file to load. The parameter 'ProcessName' defines to which process the A2L file should be linked. Additionally it can be defined some flags with the parameter 'UpdateFlag'. It can have following bit values:

```
#define A2L_LINK_NO_FLAGS                0x0
#define A2L_LINK_UPDATE_FLAG            0x1
```

```
#define A2L_LINK_UPDATE_IGNORE_FLAG          0x2
#define A2L_LINK_UPDATE_ZERO_FLAG           0x4
#define A2L_LINK_ADDRESS_TRANSLATION_DLL_FLAG 0x8
#define A2L_LINK_ADDRESS_TRANSLATION_MULTI_DLL_FLAG 0x10
#define A2L_LINK_REMEMBER_REFERENCED_LABELS_FLAG 0x20
```

Return value

== 0 - everything is OK, The A2L file is loaded and parsed and an update of the addresses taken place as defined with 'UpdateFlag'.

< 0 - error

10.11.2. XilEnv_GetLinkToExternProcess

C:

```
int XilEnv_GetLinkToExternProcess(const char \*ProcessName);;
```

'XilEnv_GetLinkToExternProcess' will give back a link number of an external process. This external process must be linked with an A2L file otherwise the function call fails. Set link can be setup with the command XilEnv_SetupLinkToExternProcess or inside the 'Configuration process' dialog from the control panel.

Return value

> 0 - everything is OK. The return value is the link number.

< 0 - error

10.11.3. XilEnv_GetIndexFromLink

C:

```
int XilEnv_GetIndexFromLink(int LinkNr, const char \*Label, int TypeMask);
```

The function 'XilEnv_GetIndexFromLink' will be return a index of a characteristic or measurements. The first parameter 'LinkNr' must be the link number returned from the 'XilEnv_GetLinkToExternProcess' function. The second parameter 'Label' is the simbol name to search inside the A2L database. The third parameter defines which type should be searched it can be a combination of following bit values:

```
// This are the possible vlues of the parameter TypeMask
#define A2L_LABEL_TYPE_ALL          0xFFFF
#define A2L_LABEL_TYPE_MEASUREMENT 0xFF
#define A2L_LABEL_TYPE_SINGEL_VALUE_MEASUREMENT 0x1
#define A2L_LABEL_TYPE_1_DIM_ARRAY_MEASUREMENT 0x2
#define A2L_LABEL_TYPE_2_DIM_ARRAY_MEASUREMENT 0x4
```

```

#define A2L_LABEL_TYPE_3_DIM_ARRAY_MEASUREMENT      0x8
#define A2L_LABEL_TYPE_NOT_REFERENCED_MEASUREMENT  0x10
#define A2L_LABEL_TYPE_REFERENCED_MEASUREMENT      0x20

#define A2L_LABEL_TYPE_CALIBRATION                   0xFF00
#define A2L_LABEL_TYPE_SINGLE_VALUE_CALIBRATION     0x100
#define A2L_LABEL_TYPE_ASCII_CALIBRATION            0x200
#define A2L_LABEL_TYPE_VAL_BLK_CALIBRATION          0x400
#define A2L_LABEL_TYPE_CURVE_CALIBRATION            0x800
#define A2L_LABEL_TYPE_MAP_CALIBRATION              0x1000
#define A2L_LABEL_TYPE_CUBOID_CALIBRATION           0x2000
#define A2L_LABEL_TYPE_CUBE_4_CALIBRATION           0x4000
#define A2L_LABEL_TYPE_CUBE_5_CALIBRATION           0x8000
#define A2L_LABEL_TYPE_AXIS_CALIBRATION             0x10000

```

Return value

>=0 - everything is OK, and a matching characteristic/measurement was found. The value are an index for later access to the found characteristic/measurement. Remark: an index >= 0x40000000 are an characteristic and index < 0x40000000 are a measurement.

<0 - error or no entry with the name 'Label' found.

Example to read one measurements with the name "MyMeasurement" :

```

int Index = XilEnv_GetIndexFromLink(LinkNr, "MyMeasurement", 0xFF);
if (Index >= 0) {
    XilEnv_LINK_DATA *Data = XilEnv_GetDataFromLink(LinkNr, Index, Data, 0x3,
&Error);
    if (Data != NULL) {
        XilEnv_PrintLinkData(Data);
        XilEnv_SetLinkArrayValueDataString(Data, 1, 0, "E_Range_ER2_High");
        if (XilEnv_SetDataToLink(LinkNr, Index, Data, &Error)) {
            printf ("Error %s\n", Error);
        }
        XilEnv_FreeLinkData(Data);
    }
    Data = XilEnv_GetDataFromLink(LinkNr, Index, Data, 0x7, &Error);
    if (Data != NULL) {
        XilEnv_PrintLinkData(Data);
    }
}

```

10.11.4. XilEnv_GetNextSymbolFromLink

C:

```
int XilEnv_GetNextSymbolFromLink(int LinkNr, int Index, int TypeMask, const char
    \*Filter, char \*ret_Label, int MaxChar);
```

The function 'XilEnv_GetNextSymbolFromLink' can be used to iterate through to all characteristics and/or measurements. The first parameter 'LinkNr' must be the link number returned from the 'XilEnv_GetLinkToExternProcess' function. The second parameter 'Index' is the starting point to start the search from. If it is -1 the search is started from the beginning. The third parameter defines which type should be searched it can be a combination of following bit values:

```
// This are the possible vlues of the parameter TypeMask
#define A2L_LABEL_TYPE_ALL 0xFFFF
#define A2L_LABEL_TYPE_MEASUREMENT 0xFF
#define A2L_LABEL_TYPE_SINGEL_VALUE_MEASUREMENT 0x1
#define A2L_LABEL_TYPE_1_DIM_ARRAY_MEASUREMENT 0x2
#define A2L_LABEL_TYPE_2_DIM_ARRAY_MEASUREMENT 0x4
#define A2L_LABEL_TYPE_3_DIM_ARRAY_MEASUREMENT 0x8
#define A2L_LABEL_TYPE_NOT_REFERENCED_MEASUREMENT 0x10
#define A2L_LABEL_TYPE_REFERENCED_MEASUREMENT 0x20

#define A2L_LABEL_TYPE_CALIBRATION 0xFF00
#define A2L_LABEL_TYPE_SINGLE_VALUE_CALIBRATION 0x100
#define A2L_LABEL_TYPE_ASCII_CALIBRATION 0x200
#define A2L_LABEL_TYPE_VAL_BLK_CALIBRATION 0x400
#define A2L_LABEL_TYPE_CURVE_CALIBRATION 0x800
#define A2L_LABEL_TYPE_MAP_CALIBRATION 0x1000
#define A2L_LABEL_TYPE_CUBOID_CALIBRATION 0x2000
#define A2L_LABEL_TYPE_CUBE_4_CALIBRATION 0x4000
#define A2L_LABEL_TYPE_CUBE_5_CALIBRATION 0x8000
```

The fourth parameter 'Filter' defines the wildcard search filter. for example '*' for everything. Or 'KL_*' for everything starting with 'KL_'. The fifth parameter 'ret_Label' should be a pointer to a place where the result of the seach can be copied to. The sixth parameter 'MaxChar' defines the maximum size of the result which will copy to the address of the pointer.

Return value

>=0 - everything is OK, and a matching characteristic/measurement was found. The value are an index for later access to the found characteristic/measurement. Remark: an index >= 0x40000000 are an characteristic and index < 0x40000000 are a measurement. For continuous search use the return value as parameter 'Index' for the next calling of XilEnv_GetNextSymbolFromLink.

<0 - error or no more entrys.

Example to iterate through all measurents:

```

int x = -1;
char Label[512];
while ((x = XilEnv_GetNextSymbolFromLink(LinkNr, x, A2L_LABEL_TYPE_MEASUREMENT,
"*, Label, 512)) >= 0) {
    printf("Measurement label %s:\n", Label);
    Index = XilEnv_GetIndexFromLink(LinkNr, Label, A2L_LABEL_TYPE_MEASUREMENT);
    if (Index >= 0) {
        XilEnv_LINK_DATA *Data = XilEnv_GetDataFromLink(LinkNr, Index, NULL, 0x7,
&Error);
        if (Data != NULL) {
            XilEnv_PrintLinkData(Data);
            if (XilEnv_SetDataToLink(LinkNr, Index, Data, &Error)) {
                printf ("Error %s\n", Error);
            }
            XilEnv_FreeLinkData(Data);
        } else {
            printf ("Error %s\n", Error);
        }
    }
}
}

```

10.11.5. XilEnv_GetDataFromLink

C:

```

XilEnv_LINK_DATA* XilEnv_GetDataFromLink(int LinkNr, int Index, XilEnv_LINK_DATA
\*Reuse, int PhysFlag, const char \**ret_Error);

```

'XilEnv_GetDataFromLink' will fetch the data from an external process linked with an A2L file. The first parameter 'LinkNr' defines the Link number. 'LinkNr' is returned from the 'XilEnv_GetLinkToExternProcess' function. The second parameter 'Index' is the identifier of the characteristic/measurement to read from. 'Index' are the return value of the function XilEnv_GetIndexFromLink or XilEnv_GetNextSymbolFromLink. With the third parameter 'Reuse' the memory of a before fetched data can be reused. If a new should be allocated set this parameter to NULL. The fourth parameter 'PhysFlag' defines For example:

```

#define A2L_GET_PHYS_FLAG          0x1
#define A2L_GET_TEXT_REPLACE_FLAG 0x2
#define A2L_GET_UNIT_FLAG          0x4

```

The fifth parameter 'ret_Error' will be a pointer to a pointer. It will define a location where a pointer to an error string can be stored. If an error occurred and this pointer is not NULL it will be set to a error message.

Return value

!= NULL - everything is OK, the value represent a pointer/handle the data fetched.

== NULL - error, see the 'ret_Error' pointer.

An example reading one measurement/characteristic:

```
XilEnv_LINK_DATA *Data = XilEnv_GetDataFromLink(LinkNr, Index, Data, 0x7, &Error);
if (Data != NULL) {
    XilEnv_PrintLinkData(Data);
    if (XilEnv_SetDataToLink(LinkNr, Index, NULL, &Error)) {
        printf ("Error %s\n", Error);
    }
    XilEnv_FreeLinkData(Data);
} else {
    printf ("Error %s\n", Error);
}
```

10.11.6. XilEnv_SetDataToLink

C:

```
int XilEnv_SetDataToLink(int LinkNr, int Index, XilEnv_LINK_DATA \*Data, const
char \*\*ret_Error);
```

'XilEnv_SetDataToLink' will write back the data to an external process linked with an A2L file. The first parameter 'LinkNr' defines the Link number. 'LinkNr' is returned from the 'XilEnv_GetLinkToExternProcess' function. The second parameter 'Index' is the identifier of the characteristic/measurement to read from. 'Index' are the return value of the function XilEnv_GetIndexFromLink or XilEnv_GetNextSymbolFromLink. The third parameter 'Data' is a pointer to the data which should be write back to an external process. The fourth parameter 'ret_Error' will be a pointer to a pointer. It will define a location where a pointer to an error string can be stored. If an error accured and this pointer is not NULL it will be set to a error message.

Remark: It will be only write back values which was changed inside the 'Data' structure with one of the XilEnv_SetLinkSingleValueDataXXX or XilEnv_SetLinkArrayValueDataXXX functions.

Return value

== 0 - everything is OK, the value are successfully written back to the external process

!= 0 - error, see the 'ret_Error' pointer.

An example reading one measurement/characteristic:

```
int Index = XilEnv_GetIndexFromLink(LinkNr, "NyCharacteristic", 0xFF00);
if (Index >= 0) {
    XilEnv_LINK_DATA *Data = XilEnv_GetDataFromLink(LinkNr, Index, Data, 0x3,
&Error);
```

```

    if (Data != NULL) {
        XilEnv_PrintLinkData(Data);
        XilEnv_SetLinkSingleValueDataInt(Data, 1234);
        XilEnv_PrintLinkData(Data);
        if (XilEnv_SetDataToLink(LinkNr, Index, Data, &Error)) {
            printf ("Error %s\n", Error);
        }
        XilEnv_FreeLinkData(Data);
    }
}

```

10.11.7. XilEnv_ReferenceMeasurementToBlackboard

C:

```
int XilEnv_ReferenceMeasurementToBlackboard(int LinkNr, int Index, int DirFlags);
```

'XilEnv_ReferenceMeasurementToBlackboard' will reference a measurement out of the A2L database to the blackboard. Afterward this signal is synchronized between an the external process and the blackboard each cycle. The first parameter 'LinkNr' defines the Link number. 'LinkNr' is returned from the 'XilEnv_GetLinkToExternProcess' function. The second parameter 'Index' is the identifier of the characteristic/measurement to read from. 'Index' are the return value of the function XilEnv_GetIndexFromLink or XilEnv_GetNextSymbolFromLink. The third parameter 'DirFlags' defines the synchronize direction. It can have the following bit values:

```

/* BB -> extern process */
#define LINK_REFERENCE_SYMBOL_BB2EP_FLAG      0x0001
/* extern process -> BB */
#define LINK_REFERENCE_SYMBOL_EP2BB_FLAG      0x0002
/* BB <-> extern process */
#define LINK_REFERENCE_SYMBOL_READWRITE_FLAG  0x0003
/* default as defined inside the A2L file */
#define LINK_REFERENCE_SYMBOL_DEFAULT_FLAG    0x0000

```

Remark: It will be only write back values which was changed inside the 'Data' structure with one of the XilEnv_SetLinkSingleValueDataXXX or XilEnv_SetLinkArrayValueDataXXX functions.

Return value

> 0 - the signal was successfully referenced, the value are variable identifier inside the blackboard.

< 0 - an error is occurred.

10.11.8. XilEnv_DereferenceMeasurementFromBlackboard

C:


```
XilEnv_DereferenceMeasurementFromBlackboard(int LinkNr, int Index);
```

'XilEnv_DereferenceMeasurementFromBlackboard' will remove the reference of a measurement to the blackboard. Afterward the synchronizing between an the external process and the blackboard is stopped. The first parameter 'LinkNr' defines the Link number. 'LinkNr' is returned from the 'XilEnv_GetLinkToExternProcess' function. The second parameter 'Index' is the identifier of the characteristic/measurement to read from. 'Index' are the return value of the function XilEnv_GetIndexFromLink or XilEnv_GetNextSymbolFromLink.

Return value

== 0 - the signal was successfully dereferenced.

< 0 - an error is occurred.

10.11.9. XilEnv_GetLinkDataType

C:

```
enum A2L_DATA_TYPE XilEnv_GetLinkDataType(XilEnv_LINK_DATA \*Data);
```

'XilEnv_GetLinkDataType' will give back the type of the data 'Data'. With this function it is possible to distinguish the type of a data.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before.

Return value

>= 0 - successfull, the value represent the type of the data.

```
enum A2L_DATA_TYPE { A2L_DATA_TYPE_MEASUREMENT = 0, A2L_DATA_TYPE_VALUE = 1,
A2L_DATA_TYPE_ASCII = 2,
                    A2L_DATA_TYPE_VAL_BLK = 3, A2L_DATA_TYPE_CURVE = 4,
A2L_DATA_TYPE_MAP = 5,
                    A2L_DATA_TYPE_CUBOID = 6, A2L_DATA_TYPE_CUBE_4 = 7,
A2L_DATA_TYPE_CUBE_5 = 8,
                    A2L_DATA_TYPE_ERROR = -1 };
```

!= 0 - an error occur.

10.11.10. XilEnv_GetLinkDataArrayCount

C:

```
int XilEnv_GetLinkDataArrayCount(XilEnv_LINK_DATA \*Data);
```

'XilEnv_GetLinkDataArrayCount' will give back the number of arrays inside data 'Data'. If data is a single value like A2L_DATA_TYPE_MEASUREMENT or A2L_DATA_TYPE_VALUE with no dimensions defined it should be 0. The return value for a A2L_DATA_TYPE_VALUE should be 1, for A2L_DATA_TYPE_CURVE 2 and so on.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before.

Return value

>= 0 - successfull, the value represent the number of arrays inside data.

!= 0 - an error occur.

10.11.11. XilEnv_GetLinkDataArraySize

C:

```
int XilEnv_GetLinkDataArraySize(XilEnv_LINK_DATA \*Data, int ArrayNo);
```

'XilEnv_GetLinkDataArraySize' will give back the size of an arrays inside data 'Data'. The parameter 'ArrayNo' should be smaller as the return value of XilEnv_GetLinkDataArrayCount function.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before.

Return value

>= 0 - successfull, the value represent the size of an arrays inside data.

!= 0 - an error occur.

10.11.12. XilEnv_CopyLinkData

C:

```
XilEnv_LINK_DATA\* XilEnv_CopyLinkData(XilEnv_LINK_DATA \*Data)
```

'XilEnv_CopyLinkData' will give back a copy of the given data 'Data'.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The copy have to freed with XilEnv_FreeLinkData to avoid memory leaks.

Return value

!= NULL - successfull, the value represent a pointer to the copied data.

== NULL - an error occur.

10.11.13. XilEnv_FreeLinkData

C:

```
XilEnv_LINK_DATA\* XilEnv_FreeLinkData(XilEnv_LINK_DATA \*Data);
```

'XilEnv_FreeLinkData' will delete the given data 'Data'.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. All datas fetched with XilEnv_GetDataFromLink or copied with XilEnv_CopyLinkData should be freed to avoid memory leaks.

Return value

!= NULL - successfull, the value represent a pointer to the copied data.

== NULL - an error occur.

10.11.14. XilEnv_GetLinkSingleValueDataType

C:

```
enum A2L_ELEM_TYPE XilEnv_GetLinkSingleValueDataType(XilEnv_LINK_DATA \*Data);
```

'XilEnv_GetLinkSingleValueDataType' will give back the data type of a single value data.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType must be 0 or 1.

Return value

>= 0 - successfull, the value represent the data type of 'Data'.

```
enum A2L_ELEM_TYPE { A2L_ELEM_TYPE_INT = 0, A2L_ELEM_TYPE_UINT = 1,  
A2L_ELEM_TYPE_DOUBLE = 2,  
A2L_ELEM_TYPE_PHYS_DOUBLE = 3, A2L_ELEM_TYPE_TEXT_REPLACE =  
4, A2L_ELEM_TYPE_ERROR = -1 };
```

< 0 - an error occur.

10.11.15. XilEnv_GetLinkSingleValueTargetDataType

C:

```
enum A2L_ELEM_TARGET_TYPE XilEnv_GetLinkSingleValueTargetDataType(XilEnv_LINK_DATA  
\*Data);
```

'XilEnv_GetLinkSingleValueTargetDataTypes' will give back the target data type of a single value data.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType must be 0 or 1.

Return value

>= 0 - successfull, the value represent the data type of 'Data'.

```
enum A2L_ELEM_TARGET_TYPE { A2L_ELEM_TARGET_TYPE_INT8 = 0,
A2L_ELEM_TARGET_TYPE_UINT8 = 1,
                        A2L_ELEM_TARGET_TYPE_INT16 = 2,
A2L_ELEM_TARGET_TYPE_UINT16 = 3,
                        A2L_ELEM_TARGET_TYPE_INT32 = 4,
A2L_ELEM_TARGET_TYPE_UINT32 = 5,
                        A2L_ELEM_TARGET_TYPE_FLOAT32_IEEE = 6,
A2L_ELEM_TARGET_TYPE_FLOAT64_IEEE = 7,
                        A2L_ELEM_TARGET_TYPE_INT64 = 34,
A2L_ELEM_TARGET_TYPE_UINT64 = 35,
                        A2L_ELEM_TARGET_TYPE_FLOAT16_IEEE = 36,
                        A2L_ELEM_TARGET_TYPE_NO_TYPE = 100,
A2L_ELEM_TARGET_TYPE_ERROR = -1 };
```

< 0 - an error occur.

10.11.16. XilEnv_GetLinkSingleValueFlags

C:

```
uint32_t XilEnv_GetLinkSingleValueFlags(XilEnv_LINK_DATA \*Data);
```

'XilEnv_GetLinkSingleValueFlags' will give back all flags set of a single value data.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType must be 0 or 1.

Return value

>= 0 - successfull, the value represent the flags of 'Data'.

```
// this is the possible return value of the xxxxFlags() functions
#define A2L_VALUE_FLAG_CALIBRATION          0x1
#define A2L_VALUE_FLAG_MEASUREMENT          0x2
#define A2L_VALUE_FLAG_PHYS                 0x4
#define A2L_VALUE_FLAG_READ_ONLY            0x8
#define A2L_VALUE_FLAG_ONLY_VIRTUAL         0x10
#define A2L_VALUE_FLAG_UPDATE               0x1000
```

< 0 - an error occur.

10.11.17. XilEnv_GetLinkSingleValueAddress

C:

```
uint64_t XilEnv_GetLinkSingleValueAddress(XilEnv_LINK_DATA \*Data);
```

'XilEnv_GetLinkSingleValueAddress' will give back the target address of a single value data.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType must be 0 or 1.

Return value

> 0 - successfull, the value represent the address of 'Data'.

== 0 - an error occur.

10.11.18. XilEnv_GetLinkSingleValueDimensionCount

C:

```
int XilEnv_GetLinkSingleValueDimensionCount(XilEnv_LINK_DATA \*Data);
```

'XilEnv_GetLinkSingleValueDimensionCount' will give back the number of dimension of a single value data. Normally this value should be 0, but if a measurement or characteristic has a dimension optional parameter inside the A2L file description the return value can have a value between 1 and 3.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType must be 0 or 1.

Return value

>= 0 - successfull, the value represent the dimension numbers of 'Data'.

< 0 - an error occur.

10.11.19. XilEnv_GetLinkSingleValueDimension

C:

```
int XilEnv_GetLinkSingleValueDimension(XilEnv_LINK_DATA \*Data, int DimNo);
```

'XilEnv_GetLinkSingleValueDimension' will give back the size of a dimension of a single value data. The parameter 'DimNo' must be smaller as the return value of 'XilEnv_GetLinkSingleValueDimensionCount' and can have a value between 0...2.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType must be 0 or 1.

Return value

>= 0 - successfull, the value represent the size of a dimension of 'Data'.

< 0 - an error occur.

10.11.20. XilEnv_GetLinkSingleValueData[Double/Int/Uint]

C:

```
double XilEnv_GetLinkSingleValueDataDouble(XilEnv_LINK_DATA \*Data);
```

C:

```
int64_t XilEnv_GetLinkSingleValueDataInt(XilEnv_LINK_DATA \*Data);
```

C:

```
uint64_t XilEnv_GetLinkSingleValueDataUint(XilEnv_LINK_DATA \*Data);
```

'XilEnv_GetLinkSingleValueData[Double/Int/Uint]' will give back the value of a single value data converted to data type double/Int/Uint. The data type double are a 64 float, the data type Int are a 64 bit integer and the data type Uint are a 64 bit unsigned integer.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be 0 or 1.

Return value is the value of data converted to double.

10.11.21. XilEnv_SetLinkSingleValueData[Double/Int/Uint]

C:

```
int XilEnv_SetLinkSingleValueDataDouble(XilEnv_LINK_DATA \*Data, double Value);
```

C:

```
int XilEnv_SetLinkSingleValueDataInt(XilEnv_LINK_DATA \*Data, int64_t Value);
```

C:

```
int XilEnv_SetLinkSingleValueDataUint(XilEnv_LINK_DATA \*Data, uint64_t Value);
```

'XilEnv_SetLinkSingleValueData[Double/Int/Uint]' will set the value of data. It will set the A2L_VALUE_FLAG_UPDATE flag inside data, so it will be writeback during the XilEnv_SetDataToLink method.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be 0 or 1.

Return value:

== 0 - successfully set.

< 0 - an error occur.

10.11.22. XilEnv_GetLinkSingleValueDataString

C:

```
int XilEnv_GetLinkSingleValueDataString(XilEnv_LINK_DATA \*Data, char \*ret_Value,  
int MaxLen);
```

'XilEnv_GetLinkSingleValueDataString' will give back the value of a single value data converted to a string. The buffer maximum size must be defined with the parameter 'MaxLen'. This funktion should be used for text replaced values.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be 0 or 1.

Return value:

> 0 - successfully, the return value is the string length including the tailing 0 byte.

< 0 - an error occur.

10.11.23. XilEnv_GetLinkSingleValueDataStringPtr

C:

```
const char\* XilEnv_GetLinkSingleValueDataStringPtr(XilEnv_LINK_DATA \*Data);
```

'XilEnv_GetLinkSingleValueDataStringPtr' will give back the a pointer points to a text repace string. If data is not a text repace the returned pointer points to a empty string. You should not call this function only on data type A2L_ELEM_TYPE_TEXT_REPLACE.

Remark: The returned pointer are invalid after any other XilEnv_xxxx function! If you need a clean copy use XilEnv_GetLinkSingleValueDataString inseat. The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be 0 or 1.

Return value is a pointer to a string

10.11.24. XilEnv_SetLinkSingleValueDataString

C:

```
int __STDCALL__ XilEnv_SetLinkSingleValueDataString(XilEnv_LINK_DATA *Data,
const char *Value);
```

'XilEnv_SetLinkSingleValueDataString' will set a value to a text repace 'Value' string. Data must have thr data type A2L_ELEM_TYPE_TEXT_REPLACE, otherwise the function fails.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be 0 or 1.

Return value:

== 0 - successfully set.

< 0 - an error occur.

10.11.25. XilEnv_GetLinkSingleValueUnit

C:

```
int XilEnv_GetLinkSingleValueUnit(XilEnv_LINK_DATA *Data, char *ret_Value, int
MaxLen);
```

'XilEnv_GetLinkSingleValueUnit' will give back the unit of a single value data. The buffer maximum size must be defined with the parameter 'MaxLen'. This funktion should be used if data is fetched XilEnv_GetDataFromLink has set the A2L_GET_UNIT_FLAG flag inside the parameter 'PhysFlag', otherwise the function fails.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be 0 or 1.

Return value is a pointer to a string

10.11.26. XilEnv_GetLinkSingleValueUnitPtr

C:

```
const char* XilEnv_GetLinkSingleValueUnitPtr(XilEnv_LINK_DATA *Data);
```

'XilEnv_GetLinkSingleValueUnitPtr' will give back a pointer to the unit of a single value data. This funktion should be used if data is fetched XilEnv_GetDataFromLink has set the A2L_GET_UNIT_FLAG flag inside the parameter 'PhysFlag', otherwise the function fails.

Remark: The returned pointer are invalid after any other XilEnv_xxxx function! If you need a clean copy use XilEnv_GetLinkSingleValueDataString instead. The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be 0 or 1.

Return value:

> 0 - successfully, the return value is the string length including the tailing 0 byte.

< 0 - an error occur.

10.11.27. XilEnv_GetLinkArrayValueDataType

C:

```
enum A2L_ELEM_TYPE XilEnv_GetLinkArrayValueDataType(XilEnv_LINK_DATA *Data, int
ArrayNo, int Number);
```

'XilEnv_GetLinkArrayValueDataType' will give back the data type of an array value data. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of XilEnv_GetLinkSingleValueDimensionCount method. The parameter 'Number' defines the array element.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be larger as 1.

Return value

>= 0 - successfull, the value represent the data type of 'Data'.

```
enum A2L_ELEM_TYPE { A2L_ELEM_TYPE_INT = 0, A2L_ELEM_TYPE_UINT = 1,
A2L_ELEM_TYPE_DOUBLE = 2,
A2L_ELEM_TYPE_PHYS_DOUBLE = 3, A2L_ELEM_TYPE_TEXT_REPLACE =
4, A2L_ELEM_TYPE_ERROR = -1 };
```

< 0 - an error occur.

10.11.28. XilEnv_GetLinkArrayValueTargetDataType

C:

```
enum A2L_ELEM_TARGET_TYPE XilEnv_GetLinkArrayValueTargetDataType(XilEnv_LINK_DATA
\*Data, int ArrayNo, int Number);
```

'XilEnv_GetLinkArrayValueTargetDataType' will give back the target data type of an array value data. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of XilEnv_GetLinkSingleValueDimensionCount method. The parameter 'Number' defines the array element.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be larger as 1.

Return value

>= 0 - successfull, the value represent the data type of 'Data'.

```
enum A2L_ELEM_TARGET_TYPE { A2L_ELEM_TARGET_TYPE_INT8 = 0,
A2L_ELEM_TARGET_TYPE_UINT8 = 1,
                        A2L_ELEM_TARGET_TYPE_INT16 = 2,
A2L_ELEM_TARGET_TYPE_UINT16 = 3,
                        A2L_ELEM_TARGET_TYPE_INT32 = 4,
A2L_ELEM_TARGET_TYPE_UINT32 = 5,
                        A2L_ELEM_TARGET_TYPE_FLOAT32_IEEE = 6,
A2L_ELEM_TARGET_TYPE_FLOAT64_IEEE = 7,
                        A2L_ELEM_TARGET_TYPE_INT64 = 34,
A2L_ELEM_TARGET_TYPE_UINT64 = 35,
                        A2L_ELEM_TARGET_TYPE_FLOAT16_IEEE = 36,
                        A2L_ELEM_TARGET_TYPE_NO_TYPE = 100,
A2L_ELEM_TARGET_TYPE_ERROR = -1 };
```

< 0 - an error occur.

10.11.29. XilEnv_GetLinkArrayValueFlags

C:

```
uint32_t XilEnv_GetLinkArrayValueFlags(XilEnv_LINK_DATA \*Data, int ArrayNo, int
Number);
```

'XilEnv_GetLinkArrayValueFlags' will give back all flags set of a single value data. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of XilEnv_GetLinkSingleValueDimensionCount method. The parameter 'Number' defines the array element.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be larger than 1.

Return value

>= 0 - successfull, the value represent the flags of 'Data'.

```
// this is the possible return value of the xxxxFlags() functions
#define A2L_VALUE_FLAG_CALIBRATION          0x1
#define A2L_VALUE_FLAG_MEASUREMENT          0x2
#define A2L_VALUE_FLAG_PHYS                  0x4
#define A2L_VALUE_FLAG_READ_ONLY             0x8
#define A2L_VALUE_FLAG_ONLY_VIRTUAL          0x10
#define A2L_VALUE_FLAG_UPDATE                0x1000
```

< 0 - an error occur.

10.11.30. XilEnv_GetLinkArrayValueAddress

C:

```
uint64_t XilEnv_GetLinkArrayValueAddress(XilEnv_LINK_DATA \*Data, int ArrayNo, int
Number);
```

'XilEnv_GetLinkArrayValueAddress' will give back the target address of a single value data. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of XilEnv_GetLinkSingleValueDimensionCount method. The parameter 'Number' defines the array element.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType must be 0 or 1.

Return value

> 0 - successfull, the value represent the address of 'Data'.

== 0 - an error occur.

10.11.31. XilEnv_GetLinkArrayValueDimensionCount

C:

```
int XilEnv_GetLinkArrayValueDimensionCount(XilEnv_LINK_DATA \*Data);
```

'XilEnv_GetLinkArrayValueDimensionCount' will give back the number of dimension of a single value data.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType must be larger than 1.

Return value

>= 0 - successfull, the value represent the dimension numbers of 'Data'.

< 0 - an error occur.

10.11.32. XilEnv_GetLinkArrayValueDimension

C:

```
int XilEnv_GetLinkArrayValueDimension(XilEnv_LINK_DATA \*Data, int DimNo);
```

'XilEnv_GetLinkArrayValueDimension' will give back the size of a dimension of a single value data.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType must be be larger than 1.

Return value

>= 0 - successfull, the value represent the size of a dimension of 'Data'.

< 0 - an error occur.

10.11.33. XilEnv_GetLinkArrayValueData[Double/Int/Uint]

C:

```
double XilEnv_GetLinkSingleValueDataDouble(XilEnv_LINK_DATA \*Data, int ArrayNo, int Number);
```

C:

```
int64_t XilEnv_GetLinkSingleValueDataInt(XilEnv_LINK_DATA \*Data, int ArrayNo, int Number);
```

C:

```
uint64_t XilEnv_GetLinkSingleValueDataUint(XilEnv_LINK_DATA \*Data, int ArrayNo, int Number);
```

'XilEnv_GetLinkSingleValueData[Double/Int/Uint]' will give back the value of a array data element converted to data type double/Int/Uint. The data type double are a 64 float, the data type Int are a 64 bit integer and the data type Uint are a 64 bit unsigned integer. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of XilEnv_GetLinkSingleValueDimensionCount method. The parameter 'Number' defines the array element.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be larger than 1.

Return value is the value of data converted to double.

10.11.34. XilEnv_SetLinkArrayValueData[Double/Int/UInt]

C:

```
int XilEnv_SetLinkArrayValueDataDouble(XilEnv_LINK_DATA \*Data, int ArrayNo, int
Number, double Value);
```

C:

```
int XilEnv_SetLinkArrayValueDataInt(XilEnv_LINK_DATA \*Data, int ArrayNo, int
Number, int64_t Value);
```

C:

```
int XilEnv_SetLinkArrayValueDataUInt(XilEnv_LINK_DATA \*Data, int ArrayNo, int
Number, uint64_t Value);
```

'XilEnv_SetLinkArrayValueData[Double/Int/UInt]' will set the value of one array data element. It will set the A2L_VALUE_FLAG_UPDATE flag inside data, so it will be writeback during the XilEnv_SetDataToLink method. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of XilEnv_GetLinkSingleValueDimensionCount method. The parameter 'Number' defines the array element.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be larger than 1.

Return value:

== 0 - successfully set.

< 0 - an error occur.

10.11.35. XilEnv_GetLinkArrayValueDataString

C:

```
int XilEnv_GetLinkArrayValueDataString(XilEnv_LINK_DATA \*Data, , int ArrayNo, int
Number, char \*ret_Value, int MaxLen);
```

'XilEnv_GetLinkArrayValueDataString' will give back the value of an array element value data converted to a string. The buffer maximum size must be defined with the parameter 'MaxLen'. This funktion should be used for text replaced values. The parameter 'ArrayNo' would define the array number it should be smaller as the

return value of `XilEnv_GetLinkSingleValueDimensionCount` method. The parameter 'Number' defines the array element.

Remark: The parameter 'Data' must be determine with the `XilEnv_GetDataFromLink` function before. The return value of `XilEnv_GetLinkDataType` should be 0 or 1.

Return value:

> 0 - successfully, the return value is the string length including the tailing 0 byte.

< 0 - an error occur.

10.11.36. `XilEnv_GetLinkArrayValueDataStringPtr`

C:

```
const char* XilEnv_GetLinkArrayValueDataStringPtr(XilEnv_LINK_DATA *Data, int
ArrayNo, int Number);
```

'`XilEnv_GetLinkSingleValueDataStringPtr`' will give back a pointer points to a text repase string of an array element. If data is not a text repase the returned pointer points to a empty string. You should not call this function only on data type `A2L_ELEM_TYPE_TEXT_REPLACE`. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of `XilEnv_GetLinkSingleValueDimensionCount` method. The parameter 'Number' defines the array element.

Remark: The returned pointer are invalid after any other `XilEnv_xxxx` function! If you need a clean copy use `XilEnv_GetLinkSingleValueDataString` inseat. The parameter 'Data' must be determine with the `XilEnv_GetDataFromLink` function before. The return value of `XilEnv_GetLinkDataType` should be 0 or 1.

Return value is a pointer to a string

10.11.37. `XilEnv_SetLinkArrayValueDataString`

C:

```
int XilEnv_SetLinkArrayValueDataString(XilEnv_LINK_DATA *Data, , int ArrayNo, int
Number, const char *Value);
```

'`XilEnv_SetLinkArrayValueDataString`' will set an array element value to a text repase 'Value' string. Data must have thr data type `A2L_ELEM_TYPE_TEXT_REPLACE`, otherwise the function fails. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of `XilEnv_GetLinkSingleValueDimensionCount` method. The parameter 'Number' defines the array element.

Remark: The parameter 'Data' must be determine with the `XilEnv_GetDataFromLink` function before. The return value of `XilEnv_GetLinkDataType` should be 0 or 1.

Return value:

== 0 - successfully set.

< 0 - an error occur.

10.11.38. XilEnv_GetLinkArrayValueUnit

C:

```
int XilEnv_GetLinkArrayValueUnit(XilEnv_LINK_DATA \*Data, , int ArrayNo, int
Number, char \*ret_Value, int MaxLen);
```

'XilEnv_GetLinkArrayValueUnit' will give back the unit of an array element value data. The buffer maximum size must be defined with the parameter 'MaxLen'. This funktion should be used if data is fetched XilEnv_GetDataFromLink has set the A2L_GET_UNIT_FLAG flag inside the parameter 'PhysFlag', otherwise the function fails. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of XilEnv_GetLinkSingleValueDimensionCount method. The parameter 'Number' defines the array element.

Remark: The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be 0 or 1.

Return value is a pointer to a string

10.11.39. XilEnv_GetLinkArrayValueUnitPtr

C:

```
const char\* XilEnv_GetLinkArrayValueUnitPtr(XilEnv_LINK_DATA \*Data, int ArrayNo,
int Number);
```

'XilEnv_GetLinkArrayValueUnitPtr' will give back a pointer to the unit of an array element value data. This funktion should be used if data is fetched XilEnv_GetDataFromLink has set the A2L_GET_UNIT_FLAG flag inside the parameter 'PhysFlag', otherwise the function fails. The parameter 'ArrayNo' would define the array number it should be smaller as the return value of XilEnv_GetLinkSingleValueDimensionCount method. The parameter 'Number' defines the array element.

Remark: The returned pointer are invalid after any other XilEnv_xxxx function! If you need a clean copy use XilEnv_GetLinkSingleValueDataString inseat. The parameter 'Data' must be determine with the XilEnv_GetDataFromLink function before. The return value of XilEnv_GetLinkDataType should be 0 or 1.

Return value:

> 0 - successfully, the return value is the string length including the tailing 0 byte.

< 0 - an error occur.

10.12. OpenXilEnv for HiL

The following functions are useful for OpenXilEnv for HiL only, but they are also defined for OpenXilEnv

10.13. CAN

10.13.1. XilEnv_LoadCanVariante

C:

```
int XilEnv_LoadCanVariante(unsigned char* canfile, int channel);
```

VB:

```
Public Declare Function XilEnv_LoadCanVariante (ByVal canfile As String, ByVal  
channel As Long) As Long
```

XilEnv_LoadCanVariante loads a CAN-description "canfile" and activates it for the CAN-channel "channel". Before selected variants will be removed from the channel .If OpenXilEnv is remote-controlled from another PC, the path-specification in "canfile" has to defined by the view of the OpenXilEnv-PC.

Return value:

==0 -> all OK

!=0 -> error

10.13.2. XilEnv_LoadAndSelCanVariante

C:

```
int XilEnv_LoadAndSelCanVariante(unsigned char* canfile, int channel);
```

VB:

```
Public Declare Function XilEnv_LoadAndSelCanVariante (ByVal canfile As String,  
ByVal channel As Long) As Long
```

XilEnv_LoadAndSelCanVariante loads a CAN-description "canfile" and activates it for the CAN-channel "channel". Before selected variants will not be removed from the channel. If OpenXilEnv is remote-controlled from another PC, the path-specification in "canfile" has to defined bythe view of the OpenXilEnv-PC.

Return value:

==0 -> all OK

!=0 -> error

10.13.3. XilEnv_DellAllCanVariants

C:

```
void XilEnv_DellAllCanVariants(void); **VB:** Public Declare Sub
XilEnv_DellAllCanVariants ()
```

XilEnv_DellAllCanVariants deletes all CAN-variants from the internal CAN-database. Be careful, these variants are delete completely!

10.13.4. XilEnv_TransmitCAN

C:

```
int XilEnv_TransmitCAN (int channel, int id, int ext, int size,
```

unsigned char data0, unsigned char data1,

unsigned char data2, unsigned char data3,

unsigned char data4, unsigned char data5,

unsigned char data6, unsigned char data7);

Sends one CAN-object once through the channel **channel** with identifier **id**. The identifier can be an extended identifier (29bit), for **ext**='1' or for **ext**='0' the identifier is a standard identifier. The size of the CAN-object is specified by **size** (0...8). All 8 databytes of the CAN-object **data0...7** must always be specified even if the object is smaller.

10.13.5. XilEnv_SetCanErr

C:

```
int XilEnv_SetCanErr (int Channel, int Id, int Startbit, int Bitsize, char
\*Byteorder, unsigned long Cycles, unsigned \_\_int64 BitErrValue);
```

'XilEnv_SetCanErr' overwrites the number of bits "Bitsize" from bit-position "Startbit" of the transmit CAN-message with the identifier "Id" on the CAN-channel with the value "BitErrValue" for the duration of "Cycles".

Special case:

If the parameter "Startbit" is set to '-1', the data length of the CAN-object is set to the value of "Bitsize" for the duration of "Cycles". Thereby it must be regarded that Bitsize is a multiple of 8 (8, 16, ..., 64). The parameter

"Byteorder" and "ErrValue" are ignored for this case.

If the parameter "Startbit" is set to '-2', the sending of the CAN-object that is defined by the parameter "Channel" and "Id" is interrupted for the duration of "Cycles".

10.13.6. XilEnv_SetCanErrSignalName

C:

```
int XilEnv_SetCanErrSignalName (int Channel, int Id, char \*Signalname, unsigned long Cycles, unsigned \_\_int64 BitErrValue);
```

'XilEnv_SetCanErrSignalName' overwrites the signal called "Signalname" of the transmit CAN-message, identifier "Id" of the CAN-channel "Channel" with the value "ErrValue" for the duration of "Cycles".

When parameter „Channel" and parameter „Id" have the value '-1' the signal is searched in all objects of all CAN-channels. If the signal name appears several times at different positions, the position/CAN-object/channel found at first, will be changed.

10.13.7. XilEnv_ClearCanErr

C:

```
int XilEnv_ClearCanErr (void);
```

XilEnv_ClearCanErr stops the overwriting of bits in a CAN-message before the end of the specified duration "cycles".

10.13.8. XilEnv_SetCanSignalConversion

C:

```
int XilEnv_SetCanSignalConversion (int Channel, int Id, const char \*Signalname, const char \*Conversion);
```

'XilEnv_SetCanSignalConversion' implements a conversion within the active CAN-simulation of the remaining bus. The changes are only activated as long as the CAN-server is reinitialized or cancelled by 'XilEnv_ResetCanSignalConversion'. Thereby **Channel** is the CAN-channel number (0...3), **Id** is the CAN-identifier of the object, **Signalname** is the signal in the CAN-objekt whose conversion should be changed and **Conversion** is the new calculation method for this signal.

If **Channel** == '-1' searching for the object/signal on all CAN-channels.

If **ID** == '-1' searching of the signal on all CAN-objects.

10.13.9. XilEnv_ResetCanSignalConversion

C:

```
int XilEnv_ResetCanSignalConversion (int Channel, int Id, const char
\*Signalname);
```

'ResetCanSignalConversion' resets the changes of the conversion done by one or several 'XilEnv_SetCanSignalConversion' within the active simulation of the CAN-remaining bus. Thereby **Channel** is the CAN-channel number (0...3), **ID** is the CAN-identifier of the object, **Signalname** is the signal in the CAN-object whose conversion should be reset.

If the signalname == NULL, all changes on the CAN-bus and the chosen objects are reset. If **ID** == '-1' or left out, all changes in the CAN-objects are reset. If **Channel** == '-1' or left out, all changes are reset.

10.13.10. XilEnv_ResetAllCanSignalConversion

C:

```
int XilEnv_ResetAllCanSignalConversion (int Channel, int Id);
```

'XilEnv_ResetAllCanSignalConversion' resets all change(s) of conversion that were set through one or several 'XilEnv_SetCanSignalConversion' instructions in the active simulation of the CAN-remaining bus. Thereby **Channel** is the CAN-channel number (0...3), **ID** is the CAN-identifier of the object. If **ID** == '-1' or left out, all changes in all CAN-objects are reset. If **Channel** == '-1' or left out all changes are reset.

The function is identical to the 'ResetCanSignalConversion' instruction with signal name == NULL.

10.14. CCP-Protocol

10.14.1. XilEnv_LoadCCPConfig

C:

```
int XilEnv_LoadCCPConfig(unsigned char\* xcpfile);
```

VB:

```
Public Declare Function XilEnv_LoadCCPConfig (ByVal ccpfile As String) As Long
```

XilEnv_LoadCCPConfig loads a CCP-configuration "ccpfile". If OpenXilEnv is remote-controlled from another PC, the path-specification in "ccpfile" must be defined by the view of the OpenXilEnv-PC.

Return value:

==0 -> all OK

!=0 -> error

10.14.2. XilEnv_StartCCPBegin

C:

```
int XilEnv_StartCCPBegin(void);
```

VB:

```
Public Declare Function XilEnv_StartCCPBegin () As Long
```

The functions 'XilEnv_StartCCPBegin', 'XilEnv_StartCCPAddVar' and 'XilEnv_StartCCPEnd' are used to start a CCP-connection (measurement). 'XilEnv_StartCCPBegin' initiates a CCP-start instruction whereby a maximum of 100 variables can be enabled by the use of 'XilEnv_StartCCPAddVar'. 'XilEnv_StartCCPEnd' concludes the instruction and starts the CCP-connection (Measurement).

Return value:

==0 -> all OK

!=0 -> error

Example:

```
Dim Ret As Long

Ret = XilEnv_StartCCPBegin
Ret = XilEnv_StartCCPAddVar("Signal1")
Ret = XilEnv_StartCCPAddVar("Signal2")
Ret = XilEnv_StartCCPAddVar("Signal3")
Ret = XilEnv_StartCCPEnd
```

10.14.3. XilEnv_StartCCPAddVar

C:

```
int XilEnv_StartCCPAddVar(unsigned char* label);
```

VB:

```
Public Declare Function XilEnv_StartCCPAddVar (ByVal label As String) As Long
```

10.14.4. XilEnv_StartCCPEnd

C:

```
int XilEnv_StartCCPEnd(void);
```

VB:

```
Public Declare Function XilEnv_StartCCPEnd () As Long
```

10.14.5. XilEnv_StopCCP

C:

```
int XilEnv_StopCCP(void);
```

VB:

```
Public Declare Function XilEnv_StopCCP () As Long
```

XilEnv_StopCCP disconnects the CCP-connection from ECU.

Return value:

==0 -> all OK

!=0 -> error

10.14.6. XilEnv_StartCCPCalBegin

C:

```
int XilEnv_StartCCPCalBegin(void);
```

VB:

```
Public Declare Function XilEnv_StartCCPCalBegin () As Long
```

The functions 'XilEnv_StartCCPCalBegin', 'XilEnv_StartCCPCalAddVar' and 'XilEnv_StartCCPCalEnd' are used to start a CCP-connection (Calibration). 'XilEnv_StartCCPCalBegin' initiates a CCP-start instruction whereby a maximum of 100 variables can be enabled by the use of 'XilEnv_StartCCPCalAddVar'. 'XilEnv_StartCCPCalEnd' concludes the instruction and starts a CCP-connection (Calibration).

Return value:

==0 -> all OK

!=0 -> error

Example:

```
Dim Ret As Long

Ret = XilEnv_StartCCPCalBegin
Ret = XilEnv_StartCCPCalAddVar("PARAMETER1")
Ret = XilEnv_StartCCPCalAddVar("PARAMETER2")
Ret = XilEnv_StartCCPCalAddVar("PARAMETER3")
Ret = XilEnv_StartCCPCalEnd
```

10.14.7. XilEnv_StartCCPCalAddVar

C:

```
int XilEnv_StartCCPCalAddVar(unsigned char* label);
```

VB:

```
Public Declare Function XilEnv_StartCCPCalAddVar (ByVal label As String) As Long
```

10.14.8. XilEnv_StartCCPCalEnd

C:

```
int XilEnv_StartCCPCalEnd(void);
```

VB:

```
Public Declare Function XilEnv_StartCCPCalEnd () As Long
```

10.14.9. XilEnv_StopCCPCal

C:

```
int XilEnv_StopCCP(void);
```

VB:

```
Public Declare Function XilEnv_StopCCPCal () As Long
```

XilEnv_StopCCPCal disconnects the CCP-connection to ECU.

Return value:

==0 -> all OK

!=0 -> error

10.15. XCP-Protocol

10.15.1. XilEnv_LoadXCPConfig

C:

```
int XilEnv_LoadXCPConfig(unsigned char\* xcpfile);
```

VB:

```
Public Declare Function XilEnv_LoadXCPConfig (ByVal xcpfile As String) As Long
```

XilEnv_LoadXCPConfig loads a XCP-configuration "xcpfile". If OpenXilEnv is remote-controlled from another PC, the path-specification in "xcpfile" must be defined by the view of the OpenXilEnv-PC.

Return value:

==0 -> all OK

!=0 -> error

10.15.2. XilEnv_StartXCPBegin

C:

```
int XilEnv_StartXCPBegin(void);
```

VB:

```
Public Declare Function XilEnv_StartXCPBegin () As Long
```

The functions 'XilEnv_StartXCPBegin', 'XilEnv_StartXCPCAddVar' and 'XilEnv_StartXCPEnd' are used to start a XCP-connection (Measurement). 'XilEnv_StartXCPBegin' initiates a XCP-start instruction whereby a maximum of 100 variables can be enabled by 'XilEnv_StartXCPCAddVar'. 'XilEnv_StartXCPEnd' concludes the instruction and starts the XCP-connection (Measurement).

Return value:

==0 -> all OK

!=0 -> error

Example:

```
Dim Ret As Long

Ret = XilEnv_StartXCPBegin
Ret = XilEnv_StartXCPCAddVar("Signal1")
Ret = XilEnv_StartXCPCAddVar("Signal2")
Ret = XilEnv_StartXCPCAddVar("Signal3")
Ret = XilEnv_StartXCPEnd
```

10.15.3. XilEnv_StartXCPCAddVar

C:

```
int XilEnv_StartXCPCAddVar(unsigned char* label);
```

VB:

```
Public Declare Function XilEnv_StartXCPCAddVar (PByVal label As String) As Long
```

10.15.4. XilEnv_StartXCPEnd

C:

```
int XilEnv_StartXCPEnd(void);
```

VB:

```
Public Declare Function XilEnv_StartXCPEnd () As Long
```

10.15.5. XilEnv_StopXCP

C:

```
int XilEnv_StopXCP(void);
```

VB:

```
Public Declare Function XilEnv_StopXCP () As Long
```

XilEnv_StopXCP disconnects the XCP-connection from ECU.

Return value:

==0 -> all OK

!=0 -> error

10.15.6. XilEnv_StartXCPCalBegin

C:

```
int XilEnv_StartXCPCalBegin(void);
```

VB:

```
Public Declare Function XilEnv_StartXCPCalBegin () As Long
```

The functions 'XilEnv_StartXCPCalBegin', 'XilEnv_StartXCPCalAddVar' and 'XilEnv_StartXCPCalEnd' are used to start a XCP-connection (Calibration). 'XilEnv_StartXCPCalBegin' initiates a XCP-start instruction whereby a

maximum of 100 parameters can be enabled by 'XilEnv_StartXCPCalAddVar'. 'XilEnv_StartXCPCalEnd' concludes the instruction and starts a XCP-connection (Calibration).

Return value:

==0 -> all OK

!=0 -> error

Example:

```
Dim Ret As Long

Ret = XilEnv_StartXCPCalBegin
Ret = XilEnv_StartXCPCalAddVar("PARAMETER1")
Ret = XilEnv_StartXCPCalAddVar("PARAMETER2")
Ret = XilEnv_StartXCPCalAddVar("PARAMETER3")
Ret = XilEnv_StartXCPCalEnd
```

10.15.7. XilEnv_StartXCPCalAddVar

C:

```
int XilEnv_StartXCPCalAddVar(unsigned char* label);
```

VB:

```
Public Declare Function XilEnv_StartXCPCalAddVar (ByVal label As String) As Long
```

10.15.8. XilEnv_StartXCPCalEnd

C:

```
int XilEnv_StartXCPCalEnd(void);
```

VB:

```
Public Declare Function XilEnv_StartXCPCalEnd () As Long
```

10.15.9. XilEnv_StopXCPCal

C:

```
int XilEnv_StopXCP(void);
```

VB:

```
Public Declare Function XilEnv_StopXCPCal () As Long
```

XilEnv_StopXCPCal disconnects the XCP-connection from ECU.

Return value:

==0 -> all OK

!=0 -> error

11. Appendix

11.1. Transfer parameter

11.1.1. -ini

OpenXilEnv gets the INI-file that should be used

Example

```
-ini c:\\OpenXilEnv\\my_ini.ini
```

11.1.2. -script

If a script-file should be executed directly after the start of OpenXilEnv, the parameter -script is used.

Example

```
c:\\fkt_test\\not_test.scr
```

11.1.3. -Instance

Use this parameter if you want define an instance name. You can start OpenXilEnv more than one instance if you use difference insance names. The length of the instance name are limited to 259 chars.

Example

```
-Instance AtomatedBackgroundTest
```

11.1.4. -err2msg

Using the parameter -err2msg all OpenXilEnv-error messages can be passed to the script-message output-window (hence in the **script.msg**-file also). OpenXilEnv does not stop to wait for a confirmation on error-messages. This can also be reached by changes in the INI-file. There must be inserted the following line by hand:

```
[BasicSettings]
```

```
Convert Error to Message=1
```

All error messages are protocolled in the file OpenXilEnv.err

Attention: Error messages from external processes are ignored!

11.1.5. --ScriptErrExit

OpenXilEnv will be closed immediately if a syntax-error or run time error occur in the script-file. The exit code of XilEnv_32.exe will be 1 independent of the blackboard variable XilEnv.ExitCode. The parameter --ScriptErrExit are the same as thh both --exit_on_script_syntax_error and -exit_on_script_execution_error together. This paramter is not neccessary for XilEnv.exe this is the default.

11.1.6. --exit_on_script_syntax_error

OpenXilEnv will be closed immediately if a script-file has a syntax-error. The exit code of XilEnv_32.exe will be 1 independent of the blackboard variable XilEnv.ExitCode. This paramter is not neccessary for XilEnv.exe this is the default.

11.1.7. --exit_on_script_execution_error

OpenXilEnv will be closed immediately if a runtime error occure inside a script-file. The exit code of XilEnv_32.exe will be 1 independent of the blackboard variable XilEnv.xitCode. This paramter is not neccessary for XilEnv.exe this is the default.

11.1.8. -icon

Start OpenXilEnv as icon

11.1.9. -nogui

When the transfer parameter -nogui is specified, OpenXilEnv is running as icon permanently.

11.1.10. -Port

When the transfer parameter -Port is specified, OpenXilEnv accepts external process login over sockets. You have to define a port number (for example 1800). This will overwrite the settings from the INI file.

11.1.11. -sys (remote master only)

The transfer parameter -sys is required when you want to transfer an executable to the remote master (linux realtime PC)

11.1.12. Transfer parameter of an external process

An external process is able to start OpenXilEnv automatically. It requires two additional parameters:

--q(Waiting time)

Specifies the time in seconds where the external process searches a running OpenXilEnv. Afterwards it starts OpenXilEnv by using the path-specification described in. (normally 1 to 5 seconds)

--c(CycleDivider:Delay)

Specifies the cycle divider and the delay of the external process.

--Instance or -i

Specifies the instance of OpenXilEnv the external process should logins. You can also define the "XilEnv_Instance" environment variable instead for that.

--Server or -s

Specifies the socket address and port the external process tries to connect to OpenXilEnv. You have also to define the same port inside OpenXilEnv itself. The -Instance parameter cannot be used with this option. If you want to use more instances on the same PC you have to use different port numbers. If you only define a Server address 1800 will be assumed. The format is IpAddress@PortNumber (127.0.0.1@1800).

--DontBreakOutOfJob

If the option -DontBreakOutOfJob is set and OpenXilEnv should be started from the external process it will NOT try to start OpenXilEnv outside the job. Normally it will try to breakout of the job the external process is started inside. This is done to avoid the killing of OpenXilEnv if the external process is started inside a debugging job of eclipse.

--dllpath

If the external process uses the DLL variant of the OpenXilEnv interface this option will define the path where the file XilEnvExtProc32.dll or XilEnvExtProc64.dll will be searched.

--NoXcp

If the option -NoXcp is set the XCP driver inside the external process will be disabled (regardless it is instantiated).

--WriteBackExeToDir

With the option `-WriteBackExeToDir` you can set the executable name to store the executable if a write back is initiated.

Path-specification of the OpenXilEnv-EXE

Path of the OpenXilEnv-EXE-file that should be started.

Example:

```
--q5 c:\\OpenXilEnv\\XilEnv_32.exe --ini c:\\siftcar\\test.ini --script  
c:\\OpenXilEnv\\start.scr
```

11.2. Environment variables

11.2.1. ExternalProcess_CallFrom

If OpenXilEnv or an other process should be called by an external process because of the transfer parameters `-q` or `-w` this will be a comma separated list of the called processes. Example: `Extp1.exe -q1 Extp2.exe -q1 Extp3.exe -q1 XilEnv_32.exe -ini config.ini` will result in `ExternalProcess_CallFrom=Extp1.exe,Extp2.exe,Extp3.exe`

11.2.2. ExternalProcess_Instance

This environment variable will only be set by OpenXilEnv to define the instance for all external processes called from OpenXilEnv itself. it will be set to the value given as transfer parameter `-Instance`. This will only used by external process if the transfer parameter `-Instance` is not given.

11.2.3. XilEnv_NoGui

This environment variable will only be set by OpenXilEnv to "yes" if it is given the transferparameter `-nogui`. All external processes called from OpenXilEnv itself will be started without a icon.

11.2.4. XILENV_NO_XCP

This environment variable will only evaluated by an external process. If it is set to "yes" no XCP over ethernet will be possible even if the function `XCPWrapperInit()` is called.

11.2.5. FMU_EXTRACTOR_CMD

With this environment variable an alternative ZIP file extractor can be defined. The external process `ExtProc_FMUExtract.exe` will use this definition string for FMU extrating if it is defined. The format must include the two strings `TO_REPLACE_WITH_SOURCE_FMU` and `TO_REPLACE_WITH_DESTINATION_PATH`, witch will be replaced with the source FMU file and the destination where it should be extracted.

Example for using 7z for extracting the FMU:

```
set FMU_EXTRACTOR_CMD = c:\path_to_7z\7z.exe x TO_REPLACE_WITH_SOURCE_FMU  
-oTO_REPLACE_WITH_DESTINATION_PATH
```

11.3. Buildin-functions in formula

There are multiple functions for formula in OpenXilEnv:

11.3.1. Mathematical buildin-functions

The mathematical buildin-functions are always calculation in datatype double.

sin(x)

This function returns the sinus of x. X must be specified in radian.

Example:

```
x=3.1415926535 / 2;
```

```
a=sin(x);
```

Result: a= 1.0

cos(x)

This function returns the consine of x. X must be specified in radian.

Example:

```
x=3.1415926535;
```

```
a=cos(x);
```

Result: a= 1.0

tan(x)

This function returns the tangent of x. X must be specfied in radian.

Example:

```
x=3.1415926535 / 4;
```

```
a=tan(x);
```

Result: a= 1.0

asin(x)

This function returns the arc sine of x in radian.

Example:

```
x=1;
```

```
a=asin(x);
```

Result: a= 1,5707963267

acos(x)

This function returns the arc cosine of x in radian.

Example:

```
x=1;
```

```
a=acos(x);
```

Result: a= 0

atan(x)

This function returns the arc tangent of x in radian.

Example:

```
x=1;
```

```
a=atan(x);
```

Result: a= 0.7853981633974483

sinh(x)

This function returns the hyperbolic sine of x.

Example:

```
x=0.5;
```

```
a=sinh(x);
```

Result: a=0.521095

cosh(x)

This function returns the hyperbolic cosine of x.

Example:

```
x=0.5;
```

```
a=cosh(x);
```


Result: a= 1.12763

tanh(x)

This function returns the hyperbolic tangent of x.

Example:

```
x=0.5;
```

```
a=tanh(x);
```

Result: a= 0.462117

exp(x)

This function returns e^x , $e = 2.718281828459$.

Example:

```
x=2,302585092994;
```

```
a=exp(x);
```

Result: a= 10

pow(x,y)

This function returns x^y .

Example:

```
x=2;
```

```
y=3;
```

```
a=pow(x,y);
```

Result: a= 8

sqrt(x)

This function returns the square root of x.

Example:

```
x=2;
```

```
a=sqrt(x);
```

Result: a= 1.414

log(x)

This function calculates the logarithm to the basis e of x, $e = 2.718281828459$.

Example:

```
x=10;
```

```
a=log(x);
```

Result: a= 2.302

log10(x)

This function calculates the logarithm to the basis 10 of x.

Example:

```
x=100;
```

```
a=log10(x);
```

Result: a= 2

modulo(x,y)

This function returns the rest of the division x/y.

Example:

```
x=10;
```

```
y=4;
```

```
a=modulo(x,y);
```

Result: a= 2

abs(x)

This function returns the absolute value of x.

Example:

```
x=-10;
```

```
a=abs(x);
```

Result: a= 10

round(x)

This function rounds x up or down to the next integer value.

Example:

```
x=2.5;
```

```
a=round(x);
```

Result: a= 3

```
x=2.4;
```

```
a=round(x);
```

Result: a= 2

round_up(x)

This function rounds x up to the next integer value.

Example:

```
x=2.1;
```

```
a=round_up(x);
```

Result: a= 3

round_down(x)

This function rounds x down to the next integer value.

Example:

```
x=2.9;
```

```
a=round_down(x);
```

Result: a= 2

overflow(min,max,x)

This function checks 'x > max'. If true, the function returns min, otherwise x.

Example:

```
x=20;
```

```
max=100;
```

```
min=10;
```

```
a=overflow(min,max,x);
```

Result: a= 20

```
x=210;
```

```
max=100;
```

```
min=10;
```

```
a=overflow(min,max,x);
```

Result: a= 10

underflow(min,max,x)

This function checks 'x < min'. If true, the function returns max, otherwise x.

Example:

```
x=20;
```

```
max=100;
```

```
min=10;
```

```
a=underlow(min,max,x);
```

Result: a= 20

```
x=5;
```

```
max=100;
```

```
min=10;
```

```
a=underflow(min,max,x);
```

Result: a= 100

min(x,y)

This function returns the lower of both values (x,y).

Example:

```
x=3;
```

```
y=4;
```

```
a=min(x,y);
```

Result: a= 3

max(x,y)

This function returns the higher of both values (x,y).

Example:

```
x=3;
```

```
y=4;
```

```
a=max(x,y);
```

Result: a= 4

equal(x,y,diff)

This function returns 0 if $|x - y| > |diff|$ otherwise 1.

Example:

```
x=3.03;
```

```
y=3.034;
```

```
a=equal(x,y,0.01);
```

Result: a= 1

11.3.2. Buildin-function for bitoperations

Bitoperations are always calculated in 32bit Integer datatype.

and(x,y)

This function performs an and-operation for x and y bit by bit.

Example:

```
x=0xAA55F0F0;
```

```
y=0x5A5A0FFF;
```

```
a=and(x,y);
```

Result: a= 0x0A5000F0

or(x,y)

This function performs an or-operation for x and y bit by bit.

Example:

```
x=0xAA55F0F0;
```

```
y=0x5A5A0FFF;
```

```
a=or(x,y);
```

Result: a= 0xFA5FFFFF

xor(x,y)

This function links x and y bit by bit with a xor-operation.

Example:

```
x=0xAA55F0F0;
```

```
y=0x5A5A0FFF;
```

```
a=xor(x,y);
```

Result: a= 0xF00FFF0F

invert(x)

This function inverts x bit by bit.

Example:

```
x=0xAA55F0F0;
```

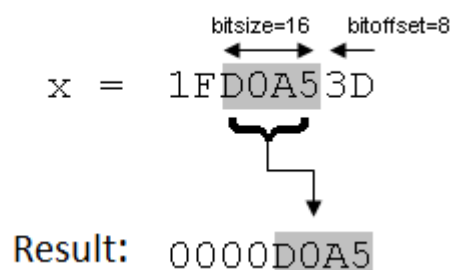
```
a=invert(x);
```

Result: a= 0x55AA0F0F

getbits(x,bitoffset,bitsize)

This function returns the bits from bit-position "bitoffset" up to the bit-position "bitsize" + "bitoffset" of x.

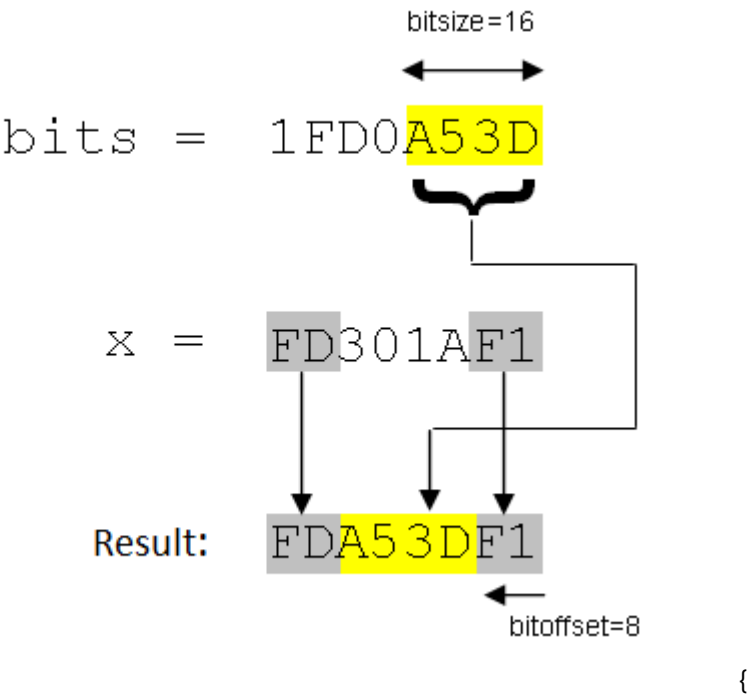
Example:



setbits(x,bitoffset,bitsize,bits)

This function replaces the bits of x from position "bitoffset" up to bit-position "bitsize" + "bitoffset" with "bits" and returns the result.

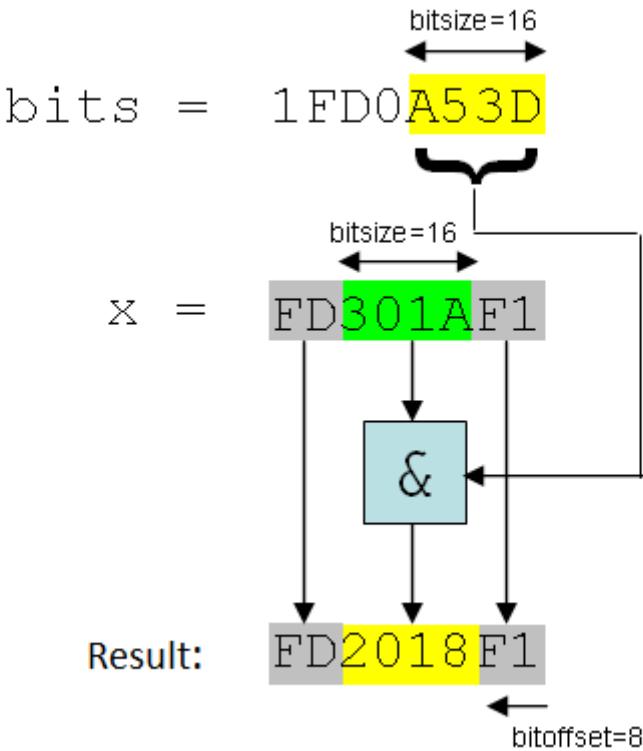
Example:



andbits(x,bitoffset,bitsize,bits)

This function performs an and-operation for the bits of x from bit-position "bitoffset" up to the bit-position "bitsize" + "bitoffset" with "bits" and returns the result.

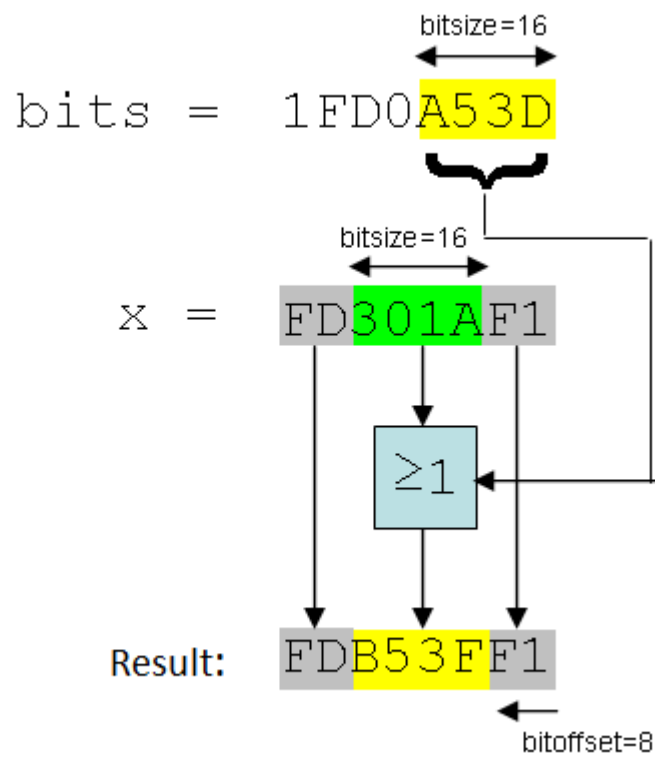
Example:



orbits(x,bitoffset,bitsize,bits)

This function performs an or-operation for the bits of x from bit-position "bitoffset" up to the bit-position "bitsize" + "bitoffset" with "bits" and returns the result.

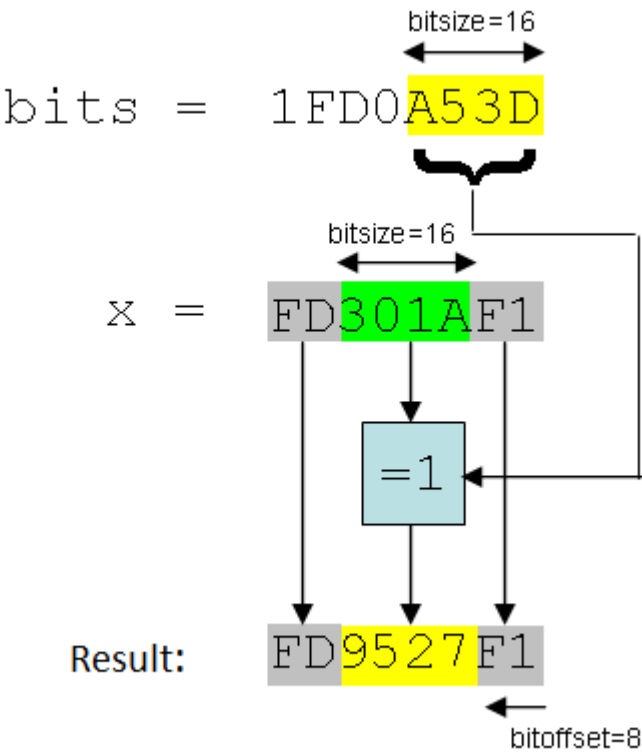
Example:



xorbits(x,bitoffset,bitsize,bits)

This function performs an xor-operation for the bits of x from bit-position "bitoffset" up to the bit-position "bitsize" + "bitoffset" with "bits" and returns the result.

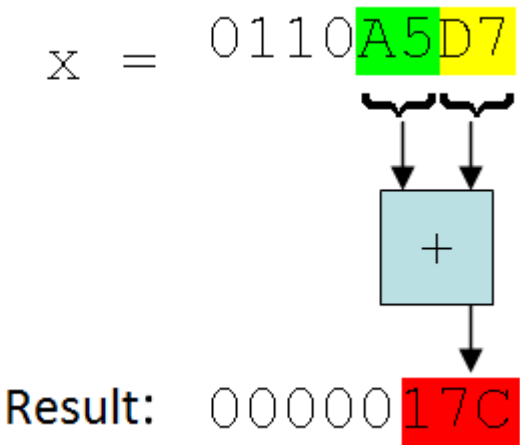
Example:



add_msb_lsb(x)

This function adds the highest byte and the lowest byte, all other bytes are ignored.

Example:



add_msn_lsn(x)

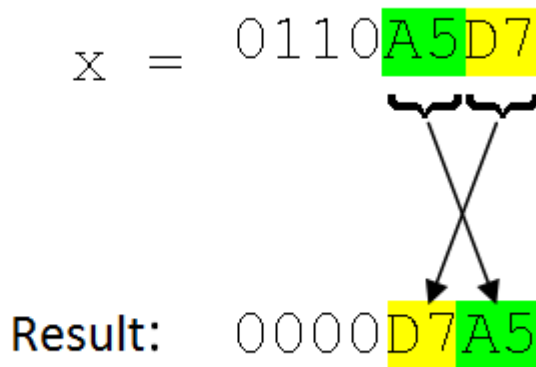
This function adds the highest nibble (4bits) and the lowest nibble, all other bytes are ignored.

Example:

{width="388px" height="255px"}

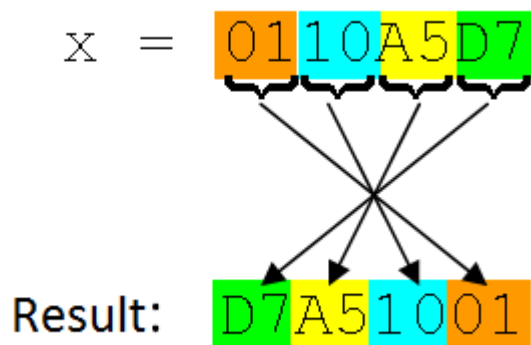
swap16(x)

This function swps the first and the second byte of a word. All other bytes are cut off.



swap32(x)

This function swaps the first and the fourth byte as well as the third and the second byte of a value.



shift_left(x,y)

This function shifts x bit by bit to the left.

shift_right(x,y)

This function shifts x bit by bit to the right.

ones_complement(Variablename)

This function reads the raw-value of a blackboard variable and builds the ones complement. All bits are inverted. Thereby the length of the datatype of the variable is regarded.

Example:

```
ADD_BBVARI (a, BYTE)
SET (a = 0)
SET (b = ones_complement(a))
MESSAGE (a = %z b = %z, a, b)
```

Output: a = 0 B = 255

get_binary(Variablename)

This function returns the binary representation of an Integer-blackboard variable.

Example:

```
ADD_BBVARI (a, BYTE)
SET (a = -2)
SET (b = get_binary(a))
MESSAGE (a = %z b = %z, a, b)
```

Output: a = -2 b = 254

set_binary(Variablename, BinaryValue)

This function sets an Integer-blackboard variable to binary and returns the result in Double.

Example:

```
ADD_BBVARI (a, BYTE)
SET (a = -2)
SET (b = set_binary(a, 0xFF))
MESSAGE (a = %z b = %z, a, b)
```

Output: a = -1 b = -1

11.3.3. CAN Buildin-functions

CAN-Buildin functions may only be used within the CAN-configuration dialogue.

canbyte(x)

This function returns the byte from byte-position x. Its value range is 0...7. If x > 7 the function always returns byte 7.

canword(x)

This function returns the word (16bit) from word-position x. Its value range is 0...3. If x > 3 the functions always returns word 3.

candword(x)

This function returns the Double-word (32bit) from Double-word-position x. Its values are 0 or 1. If $x > 1$ the function always returns the Double-word 1.

can_id()

This function returns the identifier of the current object. This function can only be used inside the CAN configuration

can_size()

This function returns the size of the current object. This function can only be used inside the CAN configuration

can_chksum8_plusid()

This function calculates the checksum for a CAN-object. The low- and high-byte of the identifier are added to all data-bytes. The function returns the lowest 8 bit of the addition.

Attention: CAN-signals that are using the function should stand on the top level. So OpenXilEnv writes this signal into the corresponding CAN-object lastly. Previously all other signals were written into the CAN-object. Otherwise a wrong checksum will be calculated.

can_chksum_xor_id(pos)

This function calculates the checksum of a CAN-object. Thereby all data-bytes and additionally the low- and high-byte of the identifier are linked with a xor-operation. The function returns the lower 8 bit of the link.

Attention: CAN-signals that are using the function should stand on the top level. So OpenXilEnv writes this signal into the corresponding CAN-object lastly. Previously all other signals were written into the CAN-object. Otherwise a wrong checksum will be calculated.

can_chksum4_plusid()

This function calculates the checksum of a CAN-object. All data-bytes are added and the identifier will be added to this result. From that the low- and high-byte are added. From that the lowest and highest 4 bits are added again. The function returns the lowest 4 bit of this calculation.

Attention: CAN-signals that are using the function should stand on the top level. So OpenXilEnv writes this signal into the corresponding CAN-object lastly. Previously all other signals were written into the CAN-object. Otherwise a wrong checksum will be calculated.

can_cyclic()

This function returns 1 if the corresponding CAN-object should be sent cyclic. Therefore the setting 'Cycles' and 'Delay' within the CAN-dialog are used. This function can be used for the setting "transmit event" to enable the cyclic sending, additional to sending controlled by result.

can_data_changed(pos,size)

This function returns '1' when a bit changed in the CAN-object. The function can be used for sending a CAN-message only when its content changed. This can be set by the setting 'transmit event'. The parameter 'pos' and 'size' are specified in bytes. If 'pos' or 'size' are less than zero the entire CAN object is checked.

can_chksum16_8()

This function calculates the checksum for a CAN-object. Thereby all data-words (16 bit) are added. From the result the low- and high-byte are added. The function returns the lowest 8 bit of the addition.

Attention: CAN-signals that are using the function should stand on the top level. So OpenXilEnv writes this signal into the corresponding CAN-object lastly. Previously all other signals were written into the CAN-object. Otherwise a wrong checksum will be calculated.

can_crc8 (AdditionalByte, Polynom, IdxCrcByte)

This function calculates the CRC-checksum as follows:

```
unsigned char can_crc8 (unsigned char AdditionalByte,
                        unsigned char Polynom,
                        unsigned char IdxCrcByte)
{
    unsigned char CrcValue;
    int ByteCount;
    int BitCount;

    CrcValue = 0xFF;
    for (ByteCount = 0; ByteCount < CANMessage.Size; ByteCount++) {
        if (ByteCount != IdxCrcByte) { /* Leave out CRC-Byte */
            CrcValue ^= CANMessage.Data[ByteCount];
            for (BitCount = 0; BitCount < 8; BitCount++) {
                if ((CrcValue & 0x80) != 0) {
                    CrcValue = (CrcValue << 1) ^ Polynom;
                } else {
                    CrcValue = (CrcValue << 1);
                }
            }
        }
    }
    CrcValue ^= AdditionalByte;
    for (BitCount = 0; BitCount < 8; BitCount++) {
        if ((CrcValue & 0x80) != 0) {
            CrcValue = (CrcValue << 1) ^ Polynom;
        } else {
            CrcValue = (CrcValue << 1);
        }
    }
    CrcValue = ~CrcValue;
    return CrcValue;
}
```

Attention: CAN-signals that are using the function should stand on the top level. So OpenXilEnv writes this signal into the corresponding CAN-object lastly. Previously all other signals were written into the CAN-object. Otherwise a wrong checksum will be calculated.

can_crc8_rev_in_out (Polynom, IdxCrcByte, ReverseInput, ReverseOutput)

This function calculates the CRC-checksum as follows:

```

unsigned char can_crc8RevInOut (unsigned char Polynom,
                                unsigned char IdxCrcByte
                                unsigned char ReverseInput)
                                unsigned char ReverseOutput) {

    unsigned char CrcValue, TestByte;
    int ByteCount;
    int BitCount;

    CrcValue = 0;
    for (ByteCount = 0; ByteCount < CANMessage.Size; ByteCount++) {
        if (ByteCount != IdxCrcByte) { /* Das CRC-Byte aussparen */
            for (BitCount = 0; BitCount < 8; BitCount++) {
                if (ReverseInput)
                    TestByte = (CANMessage.Data[ByteCount] & (1<<BitCount))<<(7-
BitCount);
                else
                    TestByte = (CANMessage.Data[ByteCount] & (1<<(7-BitCount)))
<<BitCount;

                if (TestByte != (CrcValue & 0x80))
                    CrcValue = (unsigned char)((((unsigned int)CrcValue * 2) ^
(unsigned int)Polynom));
                else
                    CrcValue = (unsigned char)((unsigned int)CrcValue * 2);
            }
        }
    }
    if (ReverseOutput) {
        TestByte = CrcValue << 7;
        for (BitCount = 1; BitCount < 4; BitCount++) {
            TestByte |= (CrcValue & (1 << BitCount)) << (7 - BitCount*2);
        }
        for (BitCount = 4; BitCount < 8; BitCount++) {
            TestByte |= (CrcValue & (1 << BitCount)) >> (BitCount*2 - 7);
        }
        CrcValue = TestByte;
    }
    return CrcValue;
}

```

Attention: CAN-signals that are using the function should stand on the top level. So OpenXilEnv writes this signal into the corresponding CAN-object lastly. Previously all other signals were written into the CAN-object. Otherwise a wrong checksum will be calculated.

can_transmit_cycle_rate (Channel-Nummer, Objekt-Name)

The function returns the configured cycle-rate of a CAN-send object. Thereby channelnumber (0...7) and object name must be passed as they were defined in the CAN-database.

can_object_id (Channel-Nummer, Objekt-Name)

The function returns the configured identifier of a CAN object. Thereby channelnumber (0...7) and object name must be passed as they were defined in the CAN-database.

11.3.4. other buildin-functions

get_process_state(Processname)

This function provides the possibility of querying the state of the external processes within the script. 'get_process_state' returns the state of an external process:

0-> process does not exist

1-> process has state 'reference' (reference_varis-fct)

2-> process has state 'init' (init_xxxx-fct)

3-> process has state 'running' (cyclic_xxxx-fct)

4-> process has state 'terminate' (terminate_xxxx-fct)

Example:

```
START_PROCESS (%XILENV_SCRIPT_DIR%\EXTPT_32.EXE)
WHILE (get_process_state (EXTPT_32.EXE) != 3)
    MESSAGE (Wait)
ENDWHILE
```

exist(Blackboard-variablename)

This function provides the possibility of querying in the script if a blackboard variable does exist. The function 'exist' returns '1' when a variable exists or '0' when no variable exists.

Example:

```
IF (!exist(dummy_variable))
    ADD_BBVARI (dummy_variable, BYTE)
```

```
ENDIF
```

env_exist(environmentvariable)

This function can be used for testing the script on the existence of the environment variable 'environmentvariable'. The function 'env_exist' returns '1' if the variable exists or '0' if not.

file_exist(file name)

This function can be used for testing the script on the existence of a file 'file name'. The function 'file_exist' returns '1' if the variable exists or '0' if not.

enum(Blackboard-variablename@enum-text)

This function returns the value that is specified in a blackboard-variable as text replacement.

The return value is the start-value of the text replacement.

Alternatively use '.' instead of '@'.

Example:

Text replacement String for blackboard-variable 'x': 0 0 "OFF"; 1 3 "ON";

```
x = enum (x@ON);
```

Result: x= 1

phys (Blackboard-variablename)

This function returns the physical value of a blackboard-variable. Also this function must have a conversion formula, if it has none there will be an error message. A physical variable can also be written when it is standing at the left side of an assignment. Than the conversion formula has to be linear.

Example:

```
phys_value = phys (variable);
phys (variable) = Phys_value;
```

strcmp (String1, String2)

This function compares two Strings. It returns '0' for equal Strings, otherwise the return value is '!= 0'.

stricmp (String1, String2)

This function compares two Strings not case-sensitive. It returns '0' for equal Strings, otherwise the return value is '!= 0'.

strncmp (String1, String2, count)

This function compares two Strings to a maximum of [**count**]{.Source .Char} characters. It returns '0' for equal Strings, otherwise the return value is '!= 0'.

strnicmp (String1, String2, count)

This function compares two strings not case-sensitive and to a maximum of [**count**]{.Source .Char} characters. It returns '0' for equal Strings, otherwise the return value is '!= 0'.

crc_0x1d (Byte0, Byte1, ..., Byte_n)

This function calculates the crc value from the Byte0 to Byte_n with the polynom 0x1d.

Sample:

```
`crc_0x1d (pdubyte(0), pdubyte(1) pdubyte(2), pdubyte(3))`
```

This will calculate the CRC checksum on the byte 0 to byte 3 of the actual PDU.

crc8_user_poly (StartValue,PolynomValue,Byte0, Byte1, ..., Byte_n)

This function calculates the crc8 value from the Byte0 to Byte_n with the polynom "PolynomValue" and the start value "StartValue". The value would be inverted at the end of the calculation.

Sample:

```
`crc8_user_poly (0xFF, 0x1D, pdubyte(0), pdubyte(1) pdubyte(2), pdubyte(3))`
```

This will calculate the CRC checksum on the byte 0 to byte 3 with the polynom 0x1D and the start value 0xFF of the actual PDU. This should have the same result as `crc_0x1d (pdubyte(0), pdubyte(1) pdubyte(2), pdubyte(3))`.

crc16_user_poly (StartValue,PolynomValue,Byte0, Byte1, ..., Byte_n)

This function calculates the crc8 value from the Byte0 to Byte_n with the polynom "PolynomValue" and the start value "StartValue". The value would be inverted at the end of the calculation.

Sample:

```
`crc16_user_poly (0xFFFF, 0x8005, pdubyte(0), pdubyte(1) pdubyte(2), pdubyte(3))`
```

This will calculate the CRC checksum on the byte 0 to byte 3 with the polynom 0x8005 and the start value 0xFFFF of the actual PDU.

has_changed (Value)

This function checks if the Value has changed between two calls. The return value is zero if unchangend and one if changed. This method is only available inside cyclic calculated formulas like: CAN transmit event, equation blocks, ... but not IF-, SET-script commands.

Sample:

```
has_changed (signal)
```

For example this function is usefull as trigger event for a CAN message.

slope_up (Value)

This function checks if the Value rises between two calls The return value is zero if the value don't rise and one if it rise. This method is only available inside cyclic calculated formulas like: CAN transmit event, equation blocks, ... but not IF-, SET-script commands.

Sample:

```
slope_up (signal)
```

slope_down (Value)

This function checks if the Value drops between two calls The return value is zero if the value don't drop and one if it drops. This method is only available inside cyclic calculated formulas like: CAN transmit event, equation blocks, ... but not IF-, SET-script commands.

Sample:

```
slope_down (signal)
```

11.4. INI-/DSK-Files

All settings are seved in the INI-file. Which INI-file should be used can be specified by the option -ini at program start. If no INI-file was selected, OpenXilEnv uses the file that was opened last. DSK-files (desktop-files) are an extract from an INI-file. They contain only items, labeled by '*'.

[BasicSettings]

Name	Meaning
Projectname=Project name	Name of the display window
xdim=1032	Width of the display window
ydim=776	High of the display window
ControlpanelXpos= 167	X-oosition of the Control Panel
ControlpanelYpos=61	Y-position of the Control Panel

Name	Meaning
CountofCycles=1000	Number of cycles when the next-button is pressed.
Breakpointactiv=0	not evaluated
SelectedControl=Script	enabled process in the "Internal Process Control"
BlackBoardSize=512	max. number of variables in the Blackboard
Editor=notepad.exe	Which editor should be used (Config-button)
Scriptmessages=1	open message-display window for script-outputs
Work Dir=folder	workspace of OpenXilEnv, switches to after start
switch off the automatic save of the NI-File=0	Switches off the automatic saving of the INI-file when exit OpenXilEnv. All changes are lost.
Priority	The following priorities can be specified: PRIORITY_NORMAL: normal use PRIORITY_BELOW_NORMAL: lower than other applications PRIORITY_LOWEST: lowest of all applications PRIORITY_IDLE: OpenXilEnv get the computing time that other applications do not use (mostly enough)
display unit for none physical values	When set to '1' the units of none physical values are displayed.
StatusbarCar	When set to '1' a little car drives back and forth in the statusbar.
StatusbarCarSpeed	The speed of the car can be tied to a blackboard-variable by using a formula.e.g.: $n_{ab}/100$
Selected sheet	Specifies the sheet in the foreground
not faster than real time	When set '1' the simulated time is roughly real time (at least not faster)
remember manual referenced labels=1	Memorizes all manual labels that were referenced through the application window. It references these labels again automatically when restarting OpenXilEnv and the corresponding process.

[Scheduler]

Name	Meaning
Period=0.010000	period duration of a cycle

ProcessName.EXE=20,3,-1

When the basic settings of process order, number of cycles between two calls and timeout not satisfactory, it can be changed here. Basic setting is 150,0,-1

History of the latest 10 files respectively breakpoints:

[last_10_Breakpoints]

[last_10_Scriptfiles]

[last_10_Generatorfiles]

[last_10_Recorderfiles]

[last_10_Playerfiles]

[last_10_Triggerfills]

Processes to be enabled at program start (internal and external). P0 is started firstly. A maximum of 31 processes are possible.

[start processes]

write protect=1

P0=EquationCompiler

P1=EquationCalculator

P2=StimuliPlayer

P3=StimuliQueue

P4=SignalGeneratorCompiler

P5=SignalGeneratorCalculator

P6=ProcessName1.EXE

P7=ProcessName2.EXE

P8=TraceQueue

P9=TraceRecorder

P10=Oscilloscope

P11=Script

Databas of all variable settings.

Variablenname = settings seperated by commas with the following meaning:

Pos.	Meaning
1	Data type (encoding see table at chapter 3.3.3)
2	Unit (max. 15 characters)
3	Min.-value (as Float)

Pos.	Meaning
4	Max.-value (as Float)
5	Display format: number of characters (1...11)
6	Display format: decimal place (0...11)
7	Step size interpretation for \pm button (0 --> linear: $x=x+step$ 1 --> procentage: $x=x*(1+step/100)$)
8	Step size for \pm button (as Float)
9	Conversion type: 0 --> none, 1 --> Conversion formula, 2 --> Text replacement
10	Conversion: when conversion type == 0 empty, when == 1 conversion formula, when == 2 text replacement description
11	Color settings (blue, green, red)

Always all settings must be specified.

There is a tool 'rob_imp.exe' to assume unit, min.-, max.-value, display format and conversion from a ROB-file.

[Variables]

Signal1=7,HZ,0.000000,0.000000,4,1,0,1.000000,1,10*Signal1 ,(116,116,116)

Signal2=7,s,0.000000,0.000000,4,3,0,1.000000,0,,(0,0,116)

Signal3=3,-,0,0,11,0,0,1.000000,0,,(0,0,0)

[all osziwindows]* contains a list of all oscilloscope-windows

ow0=	My oscilloscope
[My Oscilloscope]*	Name of the oscilloscope-window
type=oscilloscope	it is an oscilloscope-window
left=0	X-window position (left upper corner)
top=0	Y-window position (left upper corner)
width=582	width of the window
height=499	height of the window
icon=0	display window as icon
left_axis_on=1	display left axis
right_axis_on=1	display right axis
sel_left_right=0	left or right axis selected
sel_pos_left=0	which axis-label is selected at left
sel_pos_right=0	which axis-label is selected at right

[My Oscilloscope]*	Name of the oscilloscope-window
buffer_depth=1024	how far can be moved backwards (in cycles)
sample_steps=1	Always 1 at the moment
window_step_width=20	how much the window scrolls when the egde of the window was reached (in percentage)
time_window=10.000000	Window section in seconds
y_zoom=1.000000	zoom factor of all y-axes
y_off=0.000000	Offset of all y-axes
vl0=signal, 0.000000, 110.000000, (255,0,0), phys	Variable that should be displayed on first position on the left, etc.
font.lfHeight=-10	Font setting
font.lfWeight=400	Font setting
font.lfOutPrecision=1	Font setting
font.lfClipPrecision=1	Font setting
font.lfQuality=1	Font setting
font.lfPitchAndFamily=34	Font setting
font.lfFaceName=Small Fonts	Font setting

[all text windows]* contains a list of all text-windows

Ow0= My text output

[My text output]*	Name of the text-window
type=text	it is a text-window handelt sich um ein Text-Fenster
xpos=292	X-window position (left upper corner)
ypos=33	Y-window position (left upper corner)
xdim=312	width of the window
ydim=698	height of the window
E1=0,m_bremse	display variabe m_bremse at first row in decimal.
E2=0,m_brラスト	display variabe m_brラスト at first row in decimal.
E3=0,m_brsoll	display variabe m_brsoll at first row in decimal.
E4=0,m_kvbr	display variabe m_kvbr at first row in decimal.
E5=0,m_kvbrbetrag	display variabe m_kvbrbetrag at first row in decimal.
E6=0,m_kvlast	display variabe m_kvlast at first row in decimal.

[My text output]*	Name of the text-window
E7=0,m_kvsoll	display variabe m_kvsoll at first row in decimal.

[all tacho windows]* contains a list of all tacho windows

Ow0=	My tacho
[My tacho]	Name of the tacho-window
type=tachometer	it is a tacho-window
left=549	X-window position (left upper corner)
top=3	Y-window position (left upper corner)
width=122	Width of the window
height=135	height of the window
icon=0	display window as icon
variable=m_mot,-1000.000000, 1000.000000, (255,255,128), dec	
font.IfHeight=-11	Font setting
font.IfWeight=400	Font setting
font.IfOutPrecision=3	Font setting
font.IfClipPrecision=2	Font setting
font.IfQuality=1	Font setting
font.IfPitchAndFamily=34	Font setting
font.IfFaceName=Arial	Font setting

[all application windows] contains a list of all text-windows

ow0=application 1	
[application 1]	Name of the application-window
type=application	it is an application-window
process = ProcessName.EXE	Process name for application-window
filter=* const only=0	Label-filter mask (with wildcards *?)
left=206	X-window position (left upper corner)
top=225	Y-window position (left upper corner)
width=355	Width of the window
height=534	Height of the window
icon=0	display window as icon

[OpenWindows for Sheet 1]^*^ List of all windows that are opened in the first sheet

[OpenWindows for Sheet 1]^*^ List of all windows that are opened in the first sheet	
Sheet name=Ströme	Name of the sheet
F1=Ströme	First opened window in this sheet
F2=Zustände global	Second opened window in this sheet
[OpenWindows]* List of all opened windows in the foreground-sheet	
F1=Textfenster 1	First opened window in this sheet
F2=Oszifenster 2	Second opened window in this sheet

11.5. CAN files

Inside a CAN file a complete definition of one CAN variant (node) or one object or one signal can be stored. If one node is stored all object with all signals inside are also included. If only an object is stored all signals of this object are included.The basic format is an INI-file. The position of sections inside this INI file is not important. Each section must be unique inside the whole INI file. Inside one section an entry must be unique but it positions is irrelevant. All sections must be start with [CAN/...].

Each CAN file must have the section 'CAN/Global' with the entry 'copy_buffer_type'. This entry describe the type of element stored inside the CAN file. If it set to 2 it is a variant (node), if it set to 3 it is an object and if it set to 4 it is a signal.

For Example:

```
[CAN/Global]
copy_buffer_type=2
```

If the CAN file is a node definition (copy_buffer_type=2) all following section starts with [CAN/Variante_9999....]. If the CAN file is a object definition (copy_buffer_type=3) all following section starts with [CAN/Variante_9999/Object_9999....]. If the CAN file is a signal definition (copy_buffer_type=4) all following section starts with [CAN/Variante_9999/Object_9999/Signal_9999].

If the CAN file is a node definition (copy_buffer_type=2) the section [CAN/Variante_9999] describe the basic behaviour of one node.

Entry name	must exist	default value	description
name	yes		The name of the node (variante).
desc	no		Description of the node. This is only for displaying inside the CAN config dialog.
baud rate	yes	500	Set the baud rate of this node. Remark all nodes connected to on channel should have the same baud rate. If CAN FD is enabled this is the arbitration baud rate.

Entry name	must exist	default value	description
sample point	no	0.75	Defines the sample point of the hardware signal detection. (not used at the moment)
data baud rate	no	4000	Set the data phase baud rate for a CAN FD of this node. Remark all nodes connected to on channel should have the same baud rate. Only if CAN FD is enabled this is the data phase baud rate otherwise it will be ignored.
data sample point	no	0.8	Defines the sample point of the hardware signal detection. during data phase with CAN FD enabled (not used at the moment)
can fd	no	no	This will enable 'yes' or disable 'no' the CAN FD extension. This must be enabled if any CAN FD object will be defined inside this node. Default is disabled.
j1939	no	no	This will enable the j1939-21 multipackage handling . If this is enabled, OpenXilEnv can receive and transmit multi package objects.
j1939 src addr 0	no	0	If j1939-21 multi package is enabled here can be defined 4 source addresses OpenXilEnv will responds.
j1939 src addr 1	no	0	If j1939-21 multi package is enabled here can be defined 4 source addresses OpenXilEnv will responds.
j1939 src addr 2	no	0	If j1939-21 multi package is enabled here can be defined 4 source addresses OpenXilEnv will responds.
j1939 src addr 3	no	0	If j1939-21 multi package is enabled here can be defined 4 source addresses OpenXilEnv will responds.
ControlBbName	no	PrefixId	This will define the CAN object control signal names inside the blackboard. If it not set or set to 'PrefixId'. it will use following naming shema "OpenXilEnv for HiL.CAN0.[Id]". If it is set to "NoPrefixObjName" the naming shema would be "[object name]". If it is set to "PrefixObjName" the naming shema would be "OpenXilEnv for HiL.CAN0.[object name]". If it is set to "NoPrefixVarObjName" the naming shema is "[variante name].[object name]". If it is set to "PrefixVarObjName" the naming shema is "OpenXilEnv for HiL.[variante name].[object name]".
can_object_count	yes	0	This defines how many objects (receive and transmit) are inside this node. It must match the number of [CAN/Variante_9999/Object_X] sections.

: [CAN/Varante_9999]

For example:

```
[CAN/Variante_9999]
  name=VariantenName
  desc=
  baud rate=500
  sample point=0.75
  can fd=no
  data baud rate=4000
  data sample point=0.8
  j1939=yes
  j1939 src addr 0=32
  ControlBbName=PrefixId
  can_object_count=2
```

Entry name	must exist	default value	description
name	yes		The name of the object.
desc	no		Description of the object. This is only for displaying inside the CAN config dialog.
id	yes		Set the identifier of this object. If direction is set to write_variable_id this is only the init value of the identifier.
extended	no	no	Defines if the identifier is an extended identifier 'yes' or not 'no'. Default is 'no'.
bit_rate_switch	no	no	Defines if the object switch to the higher data phase baud rate. It can only that to 'yes' if the node have set the 'can fd=yes'.
fd_frame_format_switch	no	no	Defines if the object have the CAN FD frame format. It can oly that to 'yes' if the node have set the 'can fd=yes'.
direction	yes	read	This defines the direction (transmit/receive) of a CAN message. If it set to 'read' OpenXilEnv receive the message from the bus. If it is set to 'write' it will be transmit the message. If it set to 'write_variable_id' it will read the identifier from the blackboard variable 'OpenXilEnv for HiL.CAN0.[object name]' and than transmit the object with this identifier.
size	yes	8	Defines the object size it can be 1...8 for a normal CAN object 1...64 for a CAN FD object and 1...1785 for a j1939-21 object.
byteorder	yes	intel	Defines the byte order of the object. The possible values are 'intel' or 'motorola'. This byte order are used for multiplexing objects. Each normal signal have it own byteorder definition.

Entry name	must exist	default value	description
type	no	normal	Defines the object type. The value can be 'normal' for a CAN (FD) object. Or 'j1939' for a j1939-21 multi package object. Or 'j1939_multi_c_pg' for a j1939-22 multi C_PG object. Or 'j1939_c_pg' for a j1939-22 C_PG object. Or "mux" for a multiplexed object.
variable_dlc	no		If 'type=j1939' is set a blackboard signal can be define to read ore write the lenght of a transmited or received j1939 multi package message can be changed or displayed.
variable_dst_addr	no		If 'type=j1939' is set a blackboard signal can be define to read ore write the destination address of a transmited or received j1939 multi package message can be changed or displayed.
C_PGs	no		This entry can be a comma separated list of identifier of C_PG objects which should be located inside this multi C_PG object. 'type=j1939_multi_c_pg' must be set.
mux_startbit	no	0	If 'type=mux' is set this would define the start bit of the object multipler. The byte oder of the object would be consult.
mux_bit_size	no	1	If 'type=mux' is set this would define the start bit of the multipler
mux_value	no	0	If 'type=mux' is set this is the value which would be compared with the multipler inside the receiving message. Only if it match the multiplexer the receiving will be continue. For transmitting this value would be insert into the message data.
CyclicOrEvent	no	Cyclic	If the object is a transmit object this describes if the object should be send cyclic 'Cyclic' or 'Event'. If it is a cyclic object the cycle and delay entry would describe the transmit timing. if it is an event object an equation describes a transmit condition.
cycles	no	1	If 'CyclicOrEvent=Cyclic' is defined this will give the repetition cycle time. If it is 0 or 1 the object would be transmited each cycle the CAN server is calling. If it is 2 the object would be transmited each second cycle, and so on. This can be defined with an equation, this will be calculated each CAN server cycle.
delay	no	0	If 'CyclicOrEvent=Cyclic' is defined a delay time the CAN server waits after starting till the firt transmission of this object take place.

Entry name	must exist	default value	description
EventEquation	no		If 'CyclicOrEvent=Event' is defined this should be an equation which will be ececuted each cycle by the CAN server. If the return value is not zero the object would be transmited. You can use all buildin function.
signal_count	yes	0	This defines how many signals are inside this object. It must match the number of [CAN/Variante_9999/Object_X/Signal_Y] sections.
InitData	no	0x0	For an transmit object this is the init value of the object data befor any signal are placed inside.
additional_variable_#	no		With this entrys you can define additional blackboard variables which are not part of the object signals. The '#' char is a value between 0 and 99, less are possible, but no gap are allowe. First is always the datatype 'BYTE, UBYTE, WORD, UWORD, DWORD, UDWORD, QWORD, UQWORD, FLOAT or DOUBLE are possible values. Followed by the variable name.
equ_before_#	no		With this entrys you can define additional equation which should be executed before the object are transmited. The '#' char is a value between 0 and 99, less are possible but no gap are allowed. The equation can include all buildin functions.
equ_behind_#	no		With this entrys you can define additional equation which should be executed after the object are transmited. The '#' char is a value between 0 and 99, less are possible but no gap are allowed. The equation can include all buildin functions.

: [CAN/Varante_9999/Object_X]

An example of an object description of an exported CAN variant (node):

```
[CAN/Variante_9999/Object_0]
  name=TestObject
  desc=this is only a example
  id=0x123
  extended=no
  bit_rate_switch=no
  fd_frame_format_switch=no
  byteorder=intel
  size=8
  direction=write
  type=normal
```

```

mux_startbit=0
mux_bitsize=1
mux_value=0
CyclicOrEvent=Cyclic
cycles=10
delay=100
InitData=0x0
signal_count=3

```

Entry name	must exist	default value	description
name	yes		The name of the Signal. This name would be used inside the blackboard.
desc	no		Description of the signal. This is only for displaying inside the CAN config dialog.
unit	no		Set the unit of the signal. For example "mA"
bbtype	yes	UNKNOWN_DOUBLE	Defines if the data type inside the blackboard. Possible values are: BYTE, UBYTE, WORD, UWORD, DWORD, UDWORD, QWORD, UQWORD, FLOAT, DOUBLE, UNKNOWN_DOUBLE.
convtype	no	mx+b	Defines how the signal from the CAN message are converted to a blackboard signal or a blackboard signal is converted to a CAN message signal. Possible values are 'mx+b': the value would be multiply with the value of 'convert' afterwards added the value of 'offset'. 'm(x+b)': the 'offset' would be added first to the value afterwards it would be multiplied with the 'convert' value. 'curve': inside the 'convstring' are a curve definition with x/y value pairs. 'equation': inside the 'convstring' are a conversion formula defined. For example: "10*#+100". The # char represent the singnal to convert.
convert	no	1.0	Defines the factor for converting a signal if 'contype=mx+b' or 'convtype=m(x+b)'.
offset	no	0.0	Defines the offset for converting a signal if 'contype=mx+b' or 'convtype=m(x+b)'
convstring	yes		Defines a conversion string if 'contype=curve' for example "10/0 100/10 255/15". Or if 'convtype=equation' for example: "10*#+100". The # char represent the signal to convert. The equation can include all buildin functions.

Entry name	must exist	default value	description
startbit	yes	0	Defines the object size it can be 1...8 for a normal CAN object 1...64 for a CAN FD object and 1...1785 for a j1939-21 object.
bitsize	yes	8	
byteorder	yes	intel	Defines the byte order of the object. The possible values are 'intel' or 'motorola'. This byte order are used for multiplexing objects. Each normal signal have it own byteorder definition.
type	no	normal	Defines the signal type. The value can be 'normal' for a normal signal. Or "mux" for a multiplexed object. All 'mux' signals of one object with the same 'mux_startbit' and 'mux_bit_size' will be build a mux group. If 'type=mux_by' the multiplexer must be defined with the 'mux_by_signal' entry as a blackboard variable.
sign	yes	unsigned	Defines if the signal is signed or not.
mux_startbit	no	0	If 'type=mux' is set this would define the start bit of the object multiplier. The byte order of the object would be consult.
mux_bit_size	no	1	If 'type=mux' is set this would define the start bit of the multiplier
mux_value	no	0	If 'type=mux' is set this is the value which would be compared with the multiplier inside the receiving message. Only if it match the multiplexer the receiving will be continue. For transmitting this value would be insert into the message data.
mux_by_signal	no		If 'type=mux_by' this defines the multiplexer this will be added as a blackboard variable with the data type UNKNOWN_WAIT.
signal_count	yes	0	This defines how many signals are inside this object. It must match the number of [CAN/Variante_9999/Object_X/Signal_Y] sections.
startvalue	no	0.0	For an transmit object this is the init value of the object data before any signal are placed inside.
startvalue active	no	yes	if set to 'yes' the startvalue will be used as init. value of the blackboard variable each time the CAN server is restarted, otherwise it will be ignored.

: [CAN/Varante_9999/Object_X/Signal_Y]

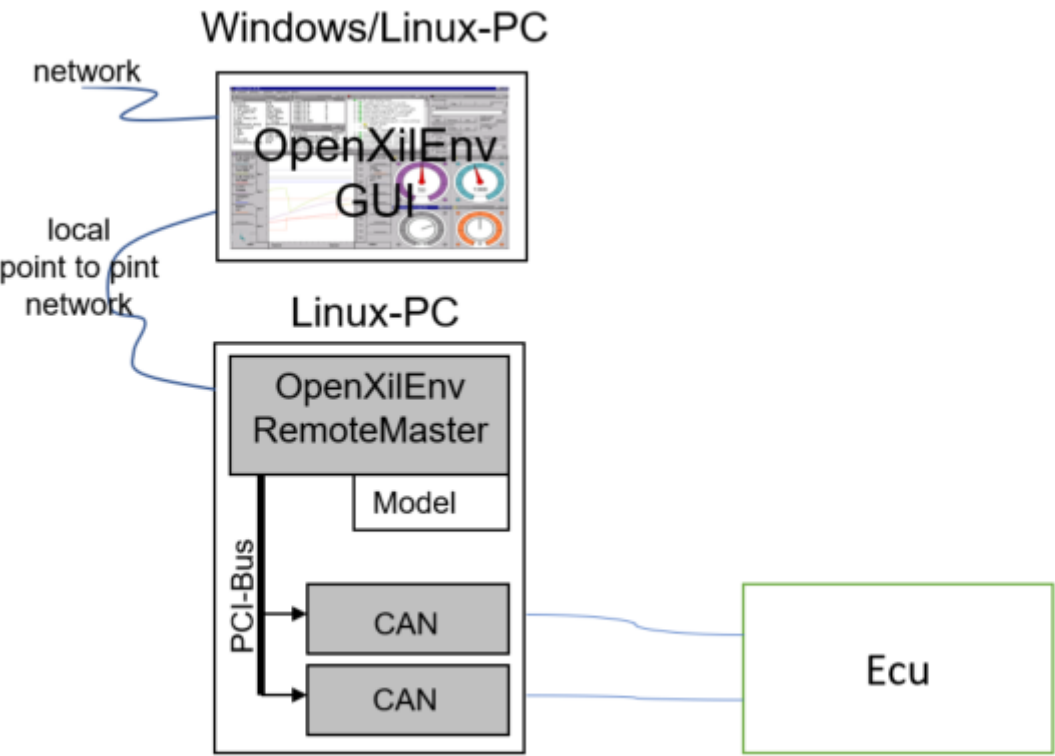
An example of signal inside an object description of an exported CAN variant (node):

```
[CAN/Variante_9999/Object_0/Signal_0]
  name=TestSignal
  desc=this is only a test signal
  unit=mA
  convtype=equation
  convert=10
  offset=1000
  convstring=
  startbit=36
  bitsize=16
  byteorder=intel
  startvalue active=yes
  startvalue=0
  type=normal
  bbtype=UWORD
  sign=unsigned
```

12. OpenXiL-Env for HiL

12.1. General information of OpenXiLEnv for HiL

OpenXiLEnv for HiL basically corresponds to OpenXiLEnv, expanded with realtime extension. The realtime extension need a second PC with a special linux image running (see section installing OpenXiLEnv for HiL). The realtime extension redirect signal generation through CAN-cards and can be include a user defined model.



- Windows10 desktop PC
- A second PC with a linux installation (see section installation OpenXilEnv for HiL)
- RAM >= 4GByte
- at least one CAN-Card or Flexcard

If an own model should be integrated, additional a Visual Studio Professional >= 2017 are usefull for remote devoleping and debugging the model as a remote linux application.

12.2. Differences between OpenXilEnv and OpenXilEnv for HiL

- No external processes possible.
- Todo

Additional processes:

Name	Meaning
CANServer	Operates the CAN-interface(s)

Addiotional variables:

Name	Datatype	Unit	Meaning
XilEnv.Schedulers.RealtimeScheduler.CyclePeriod	BB_DOUBLE	s	Current cycle time (measured)

Name	Datatype	Unit	Meaning
XilEnv.Schedulers.RealtimeScheduler.CycleCounter	BB_UDWORD	-	Current cycle count till now
XilEnv.Schedulers.RealtimeScheduler.CycleMinPeriod	BB_DOUBLE	s	Minimal cycle time (measured) till now
XilEnv.Schedulers.RealtimeScheduler.CycleMaxPeriod	BB_DOUBLE	s	Maximal cycle time (measured) till now
XilEnv.Schedulers.RealtimeScheduler.CycleCountMinPeriod	BB_UDWORD	-	Cycle count with the minimal cycle time (measured).
XilEnv.Schedulers.RealtimeScheduler.CycleCountMaxPeriod	BB_UDWORD	-	Cycle count with the maximal cycle time (measured).
XilEnv.Schedulers.RealtimeScheduler.TimeReset	BB_UDWORD	-	It the value will be changed to ≥ 1 the min/max value are reseted and the value itself will also immediately reseted to 0.
XilEnv.CAN?.0x???	BB_UDWORD	-	CAN-object: receive-counter or switch for on/off
XilEnv.CAN?	BB_UDWORD	-	CAN-Bus switch for on/off 0 -> bus is off (no messages willbe send) 1 -> bus is on
XilEnv.Realtime	BB_UWORD	-	In contrast to OpenXilEnv: always '1'

12.3. Hardware-Requirements

12.3.1. OpenXilEnv for HiL

- You need a second PC with a Linux (with realtime patch) installed.
- Todo

12.4. CAN-Bus

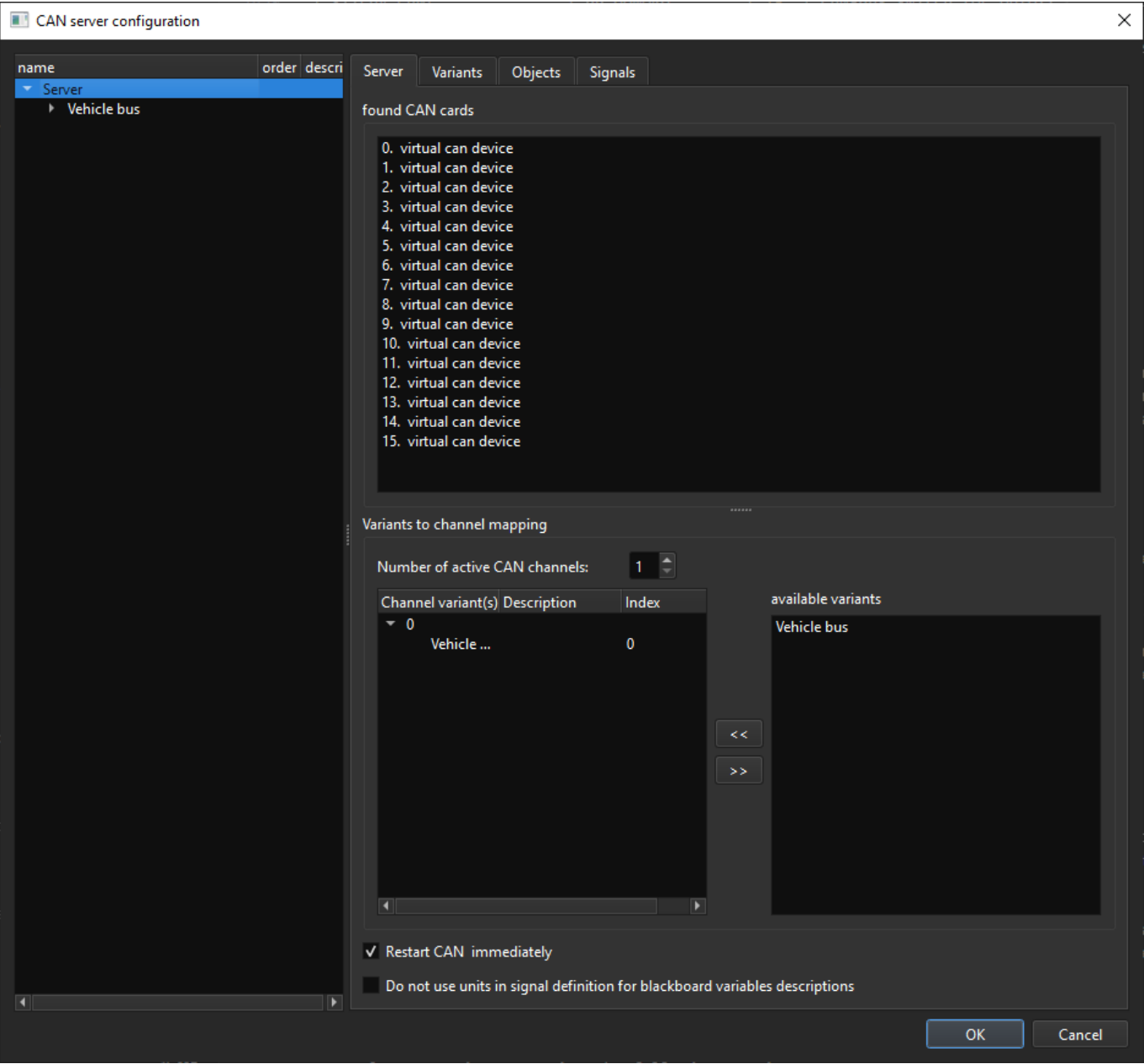
12.4.1. CAN-configuration

The settings for CAN are available at the menu **CAN --> Config**. The dialogue box consits of two parts. The left part is the configuration as tree view. The right part provides the settings of the selceted path for it. The root of the tree is always the CAN-server. The basic settings like: used CAN-card (automatically entered), number of channels and connection to a CAN-layout, can be made here.

The following connection is valid:

CAN-card	Bus	11bit and 29bit Ids mixed	Max. number of CAN-objects	Comment
todo	yes			

When selecting the CAN-server and doing a right-click with the mouse, a context menu opens. By **Add new variante** a new variant of the CAN-layout can be created, by **Paste variante** a previously copied variant can be inserted or by **Import variante** a variant from a file can be imported.

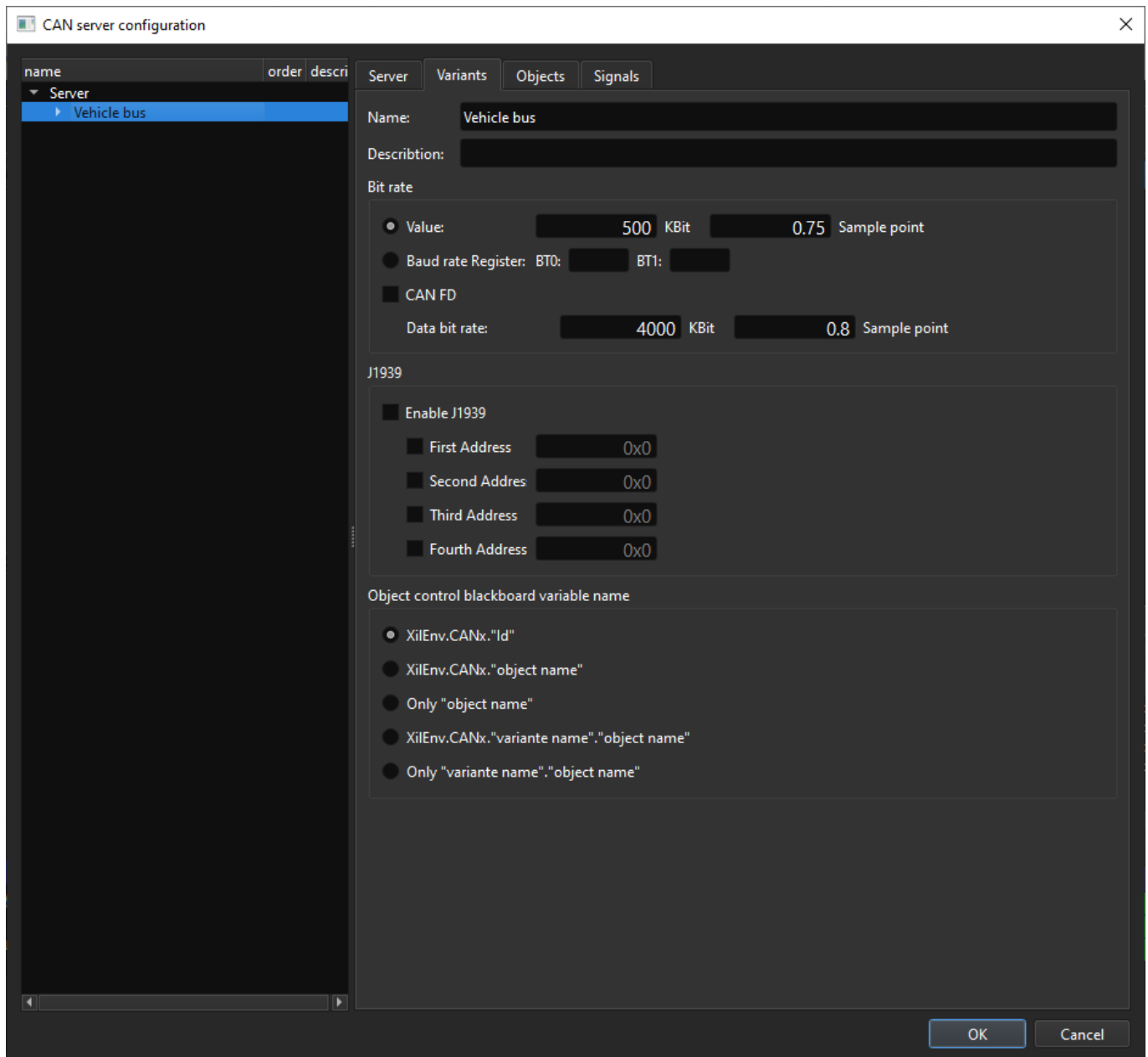


A CAN-Layout variant contains the following settings:

A variant name, a description, a baud-rate the CAN operates with, as well as all containing CAN-objects which will be visible when opening the branch. At the moment baud rates 1000,500, 250, 125 are supported. The

specification is always 'KiloBaud'.

When selecting the variant and doing a right-click with the mouse, a context menu opens. By **Add new object** a new CAN-object can be created. by **Delete variante** delete the variant, by **Copy variante** copy a variant into the clipboard, by **Paste Object** insert a CAN-object that was copied before, by **Export variante** export the selected variant into a file or import by **Import object** from a file.



A CAN-layout variant contains the following settings:

- Name, the variant is visible
- Description
- Baud rate the CAN operates with. At the moment the baud rates 1000, 500, 250, 125 are supported. The specification is always KiloBaud.

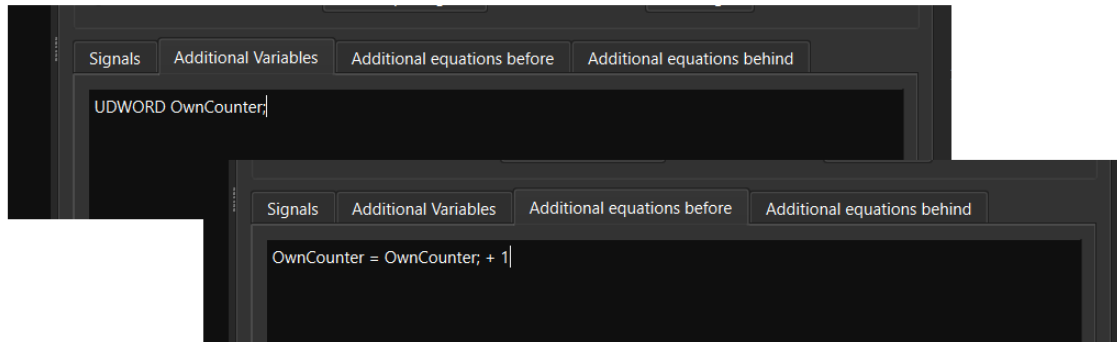
When selecting the variant and doing a right-click with the mouse, a context menu opens. By **Add new object** a new CAN-object can be created. by **Delete variante** delete the variant, by **Copy variante** copy a variant

into the clipboard, by **Paste Object** insert a CAN-object that was copied before, by **Export variante** export the selected variant into a file or import by **Import object** from a file.

The CAN-object allocated to the variant are visible when the branch is opened. A CAN-object contains the following settings:

- Name, the object is visible
- Description
- Identifier in hexa decimal representation (Extended 29Bit otherwise 11Bit)
- Number of bytes (1...8)
- Direction (read or write from the view of the PC)
- Send-condition (for „write“-objects). A cyclic sending with the settings "cycles" and "delay" and a result-controlled sending can be defined through a condition as formula. The settings "cycles" and "delay" refer to the calling-rate of the CAN-server. E.g.: Scheduling periode = 1ms, calling time-slot of the CAN-server = 10, "cycles" and "delay" have the unit 10ms. For result-controlled sending via condition, the buildin-function.
- Type of CAN-bbjects (three types):
 - **Normal:** A "normal" CAN-object is existing with the specified identifier, which is read or written cyclical.
 - **MUX:** serveral CAN-objects having the same identifier can be specified. es können mehrer CAN-Objekte mit dem selben Identifier angegeben werden. The differentiation between single objects is performed by additional bits in the data section of the object. The value of each bit must be specified with the settings MUX start bit, MUX bit size and MUX value.
 - **J1939:** This setting is only a marker and has **no** effects (so far). It is treat like a "normal" object, so it has only 8 data-bytes.
- Additional calculations can be defined before or after the sending of an object. Variables that are use in this formulas should always be declared previously in the mask „additional variables" deklariert werden. The right datatype must be specified (possible: BYTE, UBYTE, WORD, UWORD, DWORD, UDWORD, FLOAT and DOUBLE). The declaration is structured equal to C: Datatype, variablename with concluding semicolon. Every calculation has to end with a semicolon. It can also contain the buildin-functions

described in



- Format, this setting is only used for the display and has no effects on the position of single signals, these need a separate specification of the format setting. There are two object formats (Intel / Motorola), that are send through CAN-bus as follows: Motorola: 63 62 61 60 59 58 57 56 55 54 ... 13 12 11 10 9 8 7 6 5 4 3 2 1 0 Intel: 7 6 5 4 3 2 1 0 15 14 13 12 11 10 ... 49 48 63 62 61 60 59 58 57 56
- Representation bit by bit.
- Each send-CAN-object can be pre-initialized with a bit-pattern before the signals are entered.

When selecting a CAN-object and doing a right-click with the mouse, a context menu opens. By **Add new signal** a new signal can be created in the object, by **Delete object** delete the selected object, by **Copy object** copy the selected object to the clipboard, by **Paste signal** insert a previously copied signal, by **Export object** export the selected object into a file or by **Import signal** import a signal from a file.

The screenshot shows the 'CAN server configuration' window. The left sidebar lists the hierarchy: Server > Vehicle bus > [0x400] AcceleratorP... (selected). The main area has tabs for Server, Variants, Objects, and Signals. The 'Objects' tab is active, showing the configuration for the 'AcceleratorPosition' object.

Object Configuration:

- Name: AcceleratorPosition
- Description: (empty)
- Identifier: 0x400 (Extended)
- FD frame format (FDF): (unchecked)
- Bit rate switch (BRS): (unchecked)
- Size (max): 8
- Direction: write
- Transmit event:
 - Cyclic: Cycles: 1, Delay: 0
 - Equation: (empty)
- Type of object:
 - normal none multiplexed object (selected)
 - Multiplexed: Start bit: 0, Bit size: 1, Value: 0
 - J1939: 21 Multi package (dropdown), Config button

Signals Tab:

Display in: lsb_first, format: (dropdown)

	7	6	5	4	3	2	1	0
0	7	6	5	4	3	2	1	0
1	15	14	13	12	11	10	9	8
2	23	22	21	20	19	18	17	16
3	31	30	29	28	27	26	25	24
4	39	38	37	36	35	34	33	32
5	47	46	45	44	43	42	41	40
6	55	54	53	52	51	50	49	48
7	63	62	61	60	59	58	57	56

List of signals:

- PlugIn
- FireUp
- AcceleratorPosition

Initialise object with: 0x0, All 0, All 1

Buttons: OK, Cancel

Each CAN-object contains a list of signals that will be visible when opening the branch of the CAN-object.

A signal contains the following settings:

- Namen, that is used for the registration in the blackboard (no whitespaces / special characters)
- Description
- Unit
- Conversion from transfer of the CAN-bus and blackboard-value
- Startbit, the step count is dependend on the format-setting
- bit rate
- Format (Intel/Motorola)
- sign

- start value,
- Data type in the blackboard
- Signal type, there are three types of CAN-objects:
 - Normal: the signal has a fix position in the object and is read or written cyclic.
 - MUX: serveral signals can have the same bit-position. The differentiation, which signal is meant, is done by additional bits of the object. The value of the bits must be specified by the settings MUX start bit, MUX bit size, MUX value. When it is a sege, all MUX-signals are sent in sequence, dependend on "MUX value".

When selecting a CAN-object and doing a right-click with the mouse, a context menu opens. By **Delete signal** delete the selected Signal, by **Copy signal** copy the selected signal to the clipboard, by **Export signal** export the selected signal into a file.

CAN server configuration

name	order	descri
Server		
Vehicle bus		
-> [0x400] AcceleratorP...		
PlugIn	0	
FireUp	1	
AcceleratorPosition	2	

Server | Variants | Objects | **Signals**

Name: AcceleratorPosition

Description:

Unit:

Conversion

☒ $m * x + b$ CAN = * Blackboard +

☐ $m * (x + b)$

☐ Curve

☐ Equation

Start bit address:

Bit size:

Byte order:

signed/unsigned/float:

☒ Start value:

Blackboard data type:

Signal type

☒ Normal (not multiplexed) signal

☐ Multiplexed signal

Start bit: Bit size: Value:

☐ Multiplexed signal by other signal

Multiplexer Signal: Value:

OK Cancel

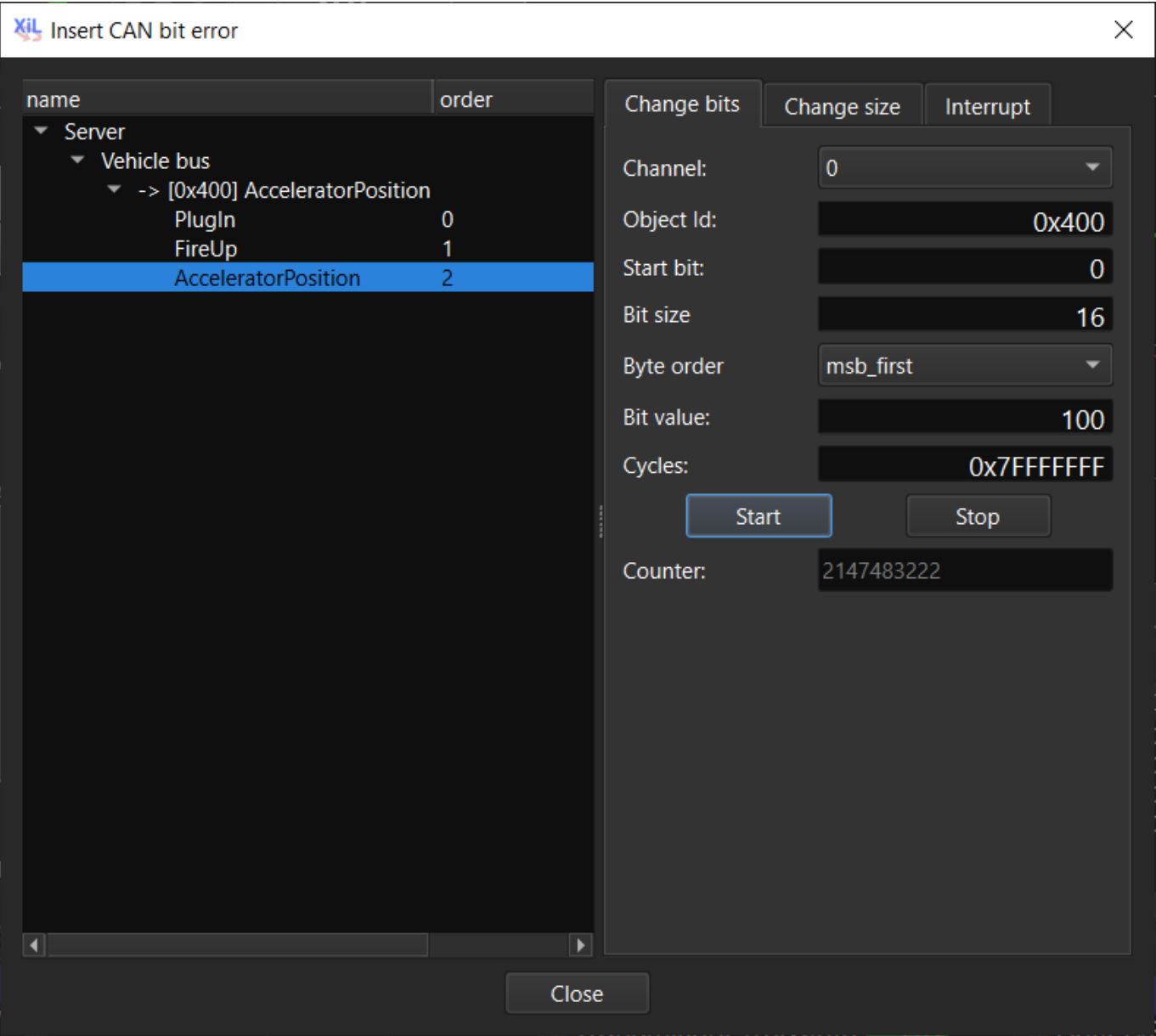
All settings are saved to the current used INI-file.

A blackboard variable is created for each CAN-object. The name consists of: „XilEnv.CAN" + Channel number + „." + Hex. Identifier.

E.g.: OpenXilEnv for HiL.CAN0.0x620 complys with the CAN-object having the ID 0x620, that was sent or read through channel 0. This variable is a message-counter for receiving-objects and a on-/off-switch for send-objects (0 -> no sending, 1-> sending).

12.4.2. Insert data-error on the CAN-bus

Through the menu „CAN" -> „Insert CAN error" the data-bits of a selectable CAN-objects can be overwritten. It can be used to overwrite an alive-counter or a checksum and set them to an invalid value.



On the left part of the tree view from the dialogue, the signal whose content should be overwritten is selected. By using "bit value", the new value of the bits from bit-position "start bit" and length "bit size" is defined. The signal conversion is **not** considered. After entering all signals in the CAN-object, the overwriting starts. The number of cycles where bits are overwritten is defined by "cycles". One cycle conforms to the send-cycle of the corresponding CAN-object.

12.4.3. CCP-application

The CCP -- protocol (V2.1) provides the possibility of reading (measure) and change (calibrate) data from the control unit.

The basic information must be specified in the CCP - configuration dialogue (CAN -> CCP Config).

The list of used variables/ parameters can be edit through the radio-buttons „Measurements" and „Calibrations". It must be regarded that the "Add"-button is pressed when entering a new label, otherwise it will be lost.

Measure

When measuring there are no special things to look for, excepting the fact of configuring variables and a minimum of one variable from window „all variables" was moved to „online variables" (see window below).

Calibration

When calibrating, it is important that the "CCP-Config" was specified on the position of data. It must be selected if the control unit has the data in its application-RAM or if OpenXiLEnv starts the copy process (Move ROM to RAM).

The calibration is only possible at the application RAM.

12.5. Integration of a model

In contrast to OpenXiLEnv a model can not be integrated as external process, but you can link it to the executable for the Linux pc side