

Unity Pro

Concept Application Converter

User Manual

04/2015

The information provided in this documentation contains general descriptions and/or technical characteristics of the performance of the products contained herein. This documentation is not intended as a substitute for and is not to be used for determining suitability or reliability of these products for specific user applications. It is the duty of any such user or integrator to perform the appropriate and complete risk analysis, evaluation and testing of the products with respect to the relevant specific application or use thereof. Neither Schneider Electric nor any of its affiliates or subsidiaries shall be responsible or liable for misuse of the information contained herein. If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

No part of this document may be reproduced in any form or by any means, electronic or mechanical, including photocopying, without express written permission of Schneider Electric.

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

© 2015 Schneider Electric. All rights reserved.

Table of Contents



	Safety Information	7
	About the Book	9
Part I	Requirements and conversion	11
Chapter 1	General Description of the Unity Pro Concept Converter	13
	General Description	14
	Conversion with the Conversion Wizard	16
Chapter 2	Requirements	19
	Concept Version	20
	Supported Hardware Platforms	21
	Configuration	22
	System	23
	EFBs	32
	Programming Language SFC	35
	Programming Language LD	36
	Programming Language ST/IL	50
	Programming Language LL984	52
	Programming Language FBD	53
Chapter 3	Language Differences	55
	Functions Not Present in Unity	56
	EFB Replaced by Function	57
	FFBs Not Available For All Platforms	58
	INOUT Parameters	62
	Parameter Type Changed	63
	ANY_ARRAY_WORD Parameters	64
	Unique Naming required	65
	Incomplete LD Generation	66
	LD Execution Order Changed	67
	Constants	71
	Indices in ST and IL	72
	Calculate with TIME and REAL	73
	WORD Assignments to BOOL Arrays	74
	Topological Address Overlapping	75
	Substitute %QD by %MF	76
	Structure Alignment Changed	77

	Undefined Output on Disabled EFs	78
	Variables at Empty Pins	80
	The set action remains active, even when the associated step becomes inactive	81
	SFC Section Retains its State When Performing an Online Modification	82
	SFCCNTRL Function Block in Unity Behaves Different to Concept	83
	Weekday Numbering	84
	System Timer	85
	Initial Values	86
	Macros	88
Chapter 4	Possible application behavior change	89
	General	90
	Concept Behavior	91
	IEC Demands	92
	Unity Behavior	95
	Consequences	97
Chapter 5	The Conversion Process	103
	Conversion Process	103
Chapter 6	Conversion Procedure	105
	Exporting a Project from Concept	106
	Importing a Project into Unity Pro	107
	Missing Datatypes at the Beginning of the Import	108
	Converting Only Parts of a Concept Application	109
	Removing Accidentally Included Concept Macros	110
	Initialization Values	111
	If the convertedMomentum application contain more than one XMIT block	112
Part II	Blocks from Concept to Unity Pro	113
Chapter 7	BYTE_TO_BIT_DFB: Type conversion	115
	Description	115
Chapter 8	CREADREG: Continuous register reading	119
	Description	120
	Mode of Functioning	123
	Parameter description	124
	Modbus Plus Error Codes	125
Chapter 9	CWRITREG: Continuous register writing	127
	Description	128
	Mode of Functioning	131
	Parameter description	132

Chapter 10	DINT_AS_WORD_DFB: Type conversion.	133
	Description.	133
Chapter 11	DIOSTAT: Module function status (DIO)	135
	Description.	135
Chapter 12	GET_TOD: Reading the hardware clock (Time Of Day)	137
	Description.	137
Chapter 13	LIMIT_IND_DFB: Limit with indicator	141
	Description.	141
Chapter 14	LOOKUP_TABLE1_DFB: Traverse progression with 1st degree interpolation.	145
	Description.	146
	Detailed description.	148
Chapter 15	PLCSTAT: PLC function status.	151
	Description.	152
	Derived Data Types.	154
	PLC status (PLC_STAT)	156
	RIO status (RIO_STAT) for Quantum	158
	DIO status (DIO_STAT).	160
Chapter 16	READREG: Read register	167
	Description.	168
	Mode of Functioning	171
	Parameter description	172
Chapter 17	RIOSTAT: Module function status (RIO)	175
	Description.	175
Chapter 18	SET_TOD: Setting the hardware clock (Time Of Day).	179
	Description.	179
Chapter 19	WORD_AS_BYTE_DFB: Type conversion	183
	Description.	183
Chapter 20	WORD_TO_BIT_DFB: Type conversion	185
	Description.	185
Chapter 21	WRITEREG: Write register.	189
	Description.	190
	Mode of Functioning	193
	Parameter description	194
Appendices		197

Appendix A	FAQ Build Errors	199
	General	200
	Object Link Creation Error	201
	Object Must be Connected to a Successor	202
	Link Together with Variable isn't Allowed.	204
	Data Type 'xxxx' Expected	205
	Empty DFB to Replace Obsolete EFB	210
	Undefined Symbol 'xxxx'	211
	Call of Non-Function Block	213
	Parameter 'xxxx' Has to Be Assigned	216
	'xxxx' Is Not a Parameter of 'yyyy'	217
	DDT Component Is Missing	218
	EHC Parameters Out of Range	219
	Not a Valid Address	220
	140 NOG 111 00 Configuration Not Converted	221
	E1163 Use of Unconfigured Direct Address	222
	The Instance Is Located on an Address That Is Not Configured	223
Appendix B	FAQ Conversion Errors	225
	FAQ Conversion Errors	225
Index		231

Safety Information



Important Information

NOTICE

Read these instructions carefully, and look at the equipment to become familiar with the device before trying to install, operate, or maintain it. The following special messages may appear throughout this documentation or on the equipment to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.



The addition of this symbol to a “Danger” or “Warning” safety label indicates that an electrical hazard exists which will result in personal injury if the instructions are not followed.



This is the safety alert symbol. It is used to alert you to potential personal injury hazards. Obey all safety messages that follow this symbol to avoid possible injury or death.

DANGER

DANGER indicates a hazardous situation which, if not avoided, **will result in** death or serious injury.

WARNING

WARNING indicates a hazardous situation which, if not avoided, **could result in** death or serious injury.

CAUTION

CAUTION indicates a hazardous situation which, if not avoided, **could result in** minor or moderate injury.

NOTICE

NOTICE is used to address practices not related to physical injury.

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

A qualified person is one who has skills and knowledge related to the construction and operation of electrical equipment and its installation, and has received safety training to recognize and avoid the hazards involved.

About the Book



At a Glance

Document Scope

This document describes the functionality and performance scope of the Concept Application Converter for Unity Pro.

Validity Note

This document is valid for Unity Pro 10.0 or later.

Related Documents

Title of Documentation	Reference Number
Unity Pro Program Languages and Structure Reference Manual	35006144 (English), 35006145 (French), 35006146 (German), 35013361 (Italian), 35006147 (Spanish), 35013362 (Chinese)
Unity Pro Operating Modes	33003101 (English), 33003102 (French),33003103 (German), 33003696 (Italian),33003104 (Spanish),33003697 (Chinese)
Modicon Modbus Plus Network Planning and Installation Guide	31003525

You can download these technical publications and other technical information from our website at www.schneider-electric.com.

Part I

Requirements and conversion

Overview

This section contains requirements and information about the conversion.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
1	General Description of the Unity Pro Concept Converter	13
2	Requirements	19
3	Language Differences	55
4	Possible application behavior change	89
5	The Conversion Process	103
6	Conversion Procedure	105

Chapter 1

General Description of the Unity Pro Concept Converter

Overview

This chapter contains a general description of the Unity Pro Concept Converter.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
General Description	14
Conversion with the Conversion Wizard	16

General Description

Brief description

The Concept Converter is an integrated function in Unity Pro, which is used to convert Concept applications into Unity Pro. This means that Concept programs can also be operated in Unity Pro.

Substitute objects are used in place of objects that cannot be converted. The Unity Pro project can be analyzed using the main menu **Create → Analyze Project**. Subsequently messages are displayed in the output window to find the substitute objects.

Elements on the Concept application that can not be converted are logged in the conversion report.

Descriptions of the respective procedures are provided in chapter Conversion procedure (*see page 105*).

NOTE: Back conversion from Unity Pro to Concept is not possible.

WARNING

UNEXPECTED APPLICATION BEHAVIOR

The Concept Converter translates the application but does not ensure its correct operation. Test the application after the conversion.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Conversion

The conversion is carried out in 4 steps:

- 1. In Concept:** Export the Concept application using the Concept converter which creates an ASCII file (*.ASC).
NOTE: Do not use the **project with used DFBs (Re-Connect to Equal)** option when creating the *.ASC file. Unity Pro cannot import the application if this option is used.
- 2. In Unity Pro:** Open the exported ASCII file (*.ASC) in Unity Pro.
- 3. In Unity Pro:** Automatic conversion of the ASCII file into Unity Pro source file format.
- 4. In Unity Pro:** Automatic import of the Unity Pro source file.

Conversion options for Concept projects

You can enter conversion options (*see Unity Pro, Operating Modes*) in Unity Pro before the conversion that have effects on the conversion result.

Atrium cannot be converted

Atrium configurations cannot be converted into Unity Pro.

Conversion Wizard

Please refer to Conversion with the Conversion Wizard ([see page 16](#)).

Conversion with the Conversion Wizard

Application Conversion as a Whole

To convert an application as a whole, while keeping the same PLC family, and selection of application parts or remapping of I/O objects is not needed, use the Concept Application Converter directly via the Unity Pro menu **File →Open**.

Partial Application Conversion

To convert an application partially and/or the PLC family must be changed or remapping of I/O objects is needed, use the conversion wizard via the Unity Pro menu **Tools →Convert Partially**.

For detailed information, please refer to the Introduction (see *Unity Pro, Operating Modes*) to the Conversion Wizard.

Conversion Wizard

The conversion wizard is an integrated part of Unity Pro.

You can use it to

- convert applications, exported out of legacy applications (Concept and PL7) to Unity Pro
- convert legacy applications partially or as a whole
- remap I/O objects (channels, variables etc.) during conversion by means of the wizard
- adapt concurrently the hardware configuration of the new application in Unity Pro
- modify the amount of used memory in the CPU

The conversion wizard is available if you have chosen to install a converter (e.g. Concept Application Converter) during the setup of Unity Pro.

General Procedure

General procedure to convert a legacy application to Unit Pro

Step	Action
1	Export your application out of your legacy programming system (e.g. as an ASC file out of Concept).
2	Create a new application in Unity Pro selecting a CPU with enough memory and the I/O access capabilities needed. Optionally you can configure the I/O modules expected to be needed but you can modify the hardware configuration even later (see step 6).
3	Launch the conversion wizard in Unity Pro via Tools →Convert Partially . Result: The conversion wizard asks you to select the exported legacy source file.
4	Select the exported legacy source file. Result: The converter analyzes the source file and displays the result in the 3 tabs of the conversion wizard.

Step	Action
5	Select the parts of the application (or the complete application) to be converted in the Structure tab.
6	Remap the I/O objects for getting them compliant with the new hardware configuration. Concurrently you can modify the hardware configuration of the new application in Unity Pro. Note: To save a backup file of your intermediate I/O mapping you can use the Save button. With Load you can reload your latest saved intermediate I/O mapping.
7	After finishing all your selections and manual modifications click OK . Result: The converter applies the defined remapping to the selected parts of the source file and imports the results into the opened Unity Pro Application.
8	Continue working on the opened application, save it or export as an XEF file.

Conversion Wizard Documentation

For detailed information on the conversion wizard, please refer to the *Operating Modes Manual -> Conversion Wizard*.

Chapter 2

Requirements

Overview

This chapter contains the requirements for converting a Concept project into a Unity Pro project.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Concept Version	20
Supported Hardware Platforms	21
Configuration	22
System	23
EFBs	32
Programming Language SFC	35
Programming Language LD	36
Programming Language ST/IL	50
Programming Language LL984	52
Programming Language FBD	53

Concept Version

General

Projects from Concept versions 2.11 and 2.5 and 2.6 can be converted to Unity Pro projects.

Preconversion

If an older version of a Concept project should be converted to Unity Pro, the project must be first converted within Concept to bring it to version 2.6 status for security reasons.

Supported Hardware Platforms

General

The Concept Converter accepts applications using the following hardware platforms:

- Quantum
- Compact
- Momentum

Manual Corrections

NOTE: The Concept Converter converts as far as possible the modules when equivalencies are existing. It is mandatory to check the result according to the process needs. The settings of the hardware modules (parameters) are not converted but set to default values and must be entered for each module in Unity Pro. Channel objects are converted as far as possible. Nevertheless, the program may have to be adapted according to the different behavior with the original module.

Quantum Applications

Concept Quantum applications are converted to Unity Pro Quantum applications.

Compact Applications

With global conversion, Concept Compact applications are converted to Quantum applications with a default hardware configuration containing a CPU (140 CPU 534 14A/U) and a power supply (140 CPS 424 00).

With partial conversion (conversion wizard) it is recommended to prepare a Modicon M340 hardware configuration.

Momentum Applications

With global conversion, Concept Momentum applications are converted to Quantum applications with a default hardware configuration containing a CPU (140 CPU 534 14A/U) and a power supply (140 CPS 424 00).

With partial conversion (conversion wizard) it is recommended to prepare a Modicon M340 hardware configuration.

Safety PLC

NOTE: It is not possible to recover an application from Concept to Unity Pro Safety PLC. To build a safety system, refer to the *Safety Manual*.

Configuration

General

Concept sections created using the LL984 programming language appear as well as non-IEC LL984 sections in Unity.

Restrictions for old LL984 configurations

The following points from LL984 configurations are no longer supported by Unity Pro:

Not supported by Unity Pro	Supported by Unity Pro
Loadables	Necessary functionality of system loadables has been integrated into Unity Pro. Unity Pro does NOT provide equivalents for all other loadables.
ASCII messages	Will not be converted.
6x range (register in expanded memory)	Will not be converted.
Data protection configuration extension	Will not be supported.

Hot Standby (HSBY)

There are the following differences for converting the Concept Hot Standby to Unity Pro:

Concept	Unity Pro
The Hot Standby system in Concept is based on the 140 CHS 111 00 module.	This module is no longer supported by Unity Pro.
The 140 CHS 111 00 module is purely a Hot Standby Module for a single slot. The power is supplied via the rack.	The CPU 671 60 module is a CPU module for two slots with a fixed assigned connection for data exchange. The Hot Standby system is integrated into the CPU 671 60 module.

The Concept converter replaces the CPU from Concept with the new Hot Standby CPU 671 60 and the Concept Hot Standby Module 140 CHS 111 00 is removed. All Hot Standby parameters will be transferred to the Unity application.

NOTE: It is not possible to recover an application from Concept to Unity Pro safety PLC. To build a safety system, refer to the *safety manual*.

NOTE: As the CPU in Concept only requires **one** slot, but the new Unity CPU requires **two**, overlaps in the rack may arise. These must be resolved manually by the user.

System

Security

The access authorizations defined in Concept are **not** converted to Unity Pro.

Security under Unity Pro does **not** - refer to the corresponding installation as it does under Concept.

Program Execution

Program execution using Concept and Unity Pro are different. It can lead to different behavior during the first program run after a restart.

Program execution for Concept:

1. Write the outputs (program run n-1)
2. Read the inputs (program run n)
3. Program processing

Program execution for Unity Pro:

1. Read the inputs
2. Program processing
3. Write the outputs

Example:

In Concept, you have assigned a 4x register to a digital output and stopped the PLC when the value is "true". After a restart, the value remains "True" during the first program run even if you have modified the process conditions.

Specified Execution Order

The execution order in the function block language in Concept is determined first of all by how the FFBs are positioned. If the FFBs are then linked graphically, the execution order is determined by the data flow. After this the execution order can be changed based on the intention.

In Unity Pro after conversion it is not possible to see in what order the FFBs were positioned. Therefore, whenever the order cannot be determined unambiguously from the data flow rule alone, the order is defined by the Concept project.

The defined execution sequence is shown by means of a rectangle with the step number in the upper right-hand corner of the FFB.

Single Sweep Function

The single sweep function is no longer supported by Unity Pro.

The corresponding functionality can be realized in Unity Pro using the Debug function "Breakpoints".

EFB Download

Using Concept, all platform dependent EFBs can be placed at any time and loaded in all PLC platforms. Any errors detected during the program execution are written to the message memory.

In Unity Pro, only valid EFBs can be placed. Download to the PLC is only possible if the EFBs used are consistent with the PLC platform.

Reference Data Editor (RDE)

RDE tables created in Concept are converted to Unity Pro when they are placed in the same directory as the Concept ASCII file.

Global Variable Values

Because of different restart behaviors after a power outage, it is possible that the global variable states of two PLCs that restart differently are not the same after the first program run.

There are two different types of restart behavior:

1. All 16 bit PLCs (all Momentum, Quantum 113, 213, 424) continue executing the program at the point at which it was interrupted.
2. All 32 bit PLCs (Quantum 434, 534) start the program run at the beginning.

Unity Pro supports the 1st type of restart behavior described above.

State RAM

The Concept State RAM register addresses are assigned to IEC conforming addresses in Unity Pro.

I/O module addresses are converted either to "flat" addresses or to topological addresses.

State RAM Register Without I/O Module

State RAM register addresses **without** assigned I/O modules are represented with "flat" addresses:

Concept	Unity Pro
4x	%MWx
3x	%IWx ⁽¹⁾
0x	%Mx

Concept	Unity Pro
1x	%Ix
(1) = If Modicon M340 is the target platform, there is no equivalent for input State RAM registers (%IWx). The addresses are converted formally to flat addresses and must be corrected by the user.	

For this, the register number is added to the end of the introduction.

The address reads as follows:

`%[IM][W]Register number`

State RAM Register With I/O Module

State RAM register addresses **with** assigned I/O modules can either be represented on Quantum with "flat" addressing as described above or with topological addressing.

To define that State RAM register addresses will be converted to topological addressing, open the **Conversion Settings** tab via **Tools → Options** in Unity Pro and activate the **Generate Topological Addresses for Quantum** check box before converting.

If the check box is not activated, the State RAM register addresses are converted to "flat" addresses (for Quantum only).

If Compact or Momentum applications are converted with the conversion wizard, topological addressing is used by default, regardless whether the check box is activated or not.

State RAM register addresses with assigned I/O modules (topological)

Concept	Unity Pro
4x	%QWt
3x	%IWt
0x	%Qt
1x	%It
t = topological description	

The following information is read from the configuration to provide a sufficient topological description of State RAM register addresses with assigned I/O modules:

- Bus number (corresponds to drop head in Concept)
- Drop
- Rack
- Module
- Channel

The complete address reads as follows:

`%[IQ][W]<\Busnumber.Drop>Rack.Module.Channel`

State RAM Assignment Using Derived Data Types

In Concept, data structure elements begin at BYTE limits.

In Unity Pro, data structure elements begin at WORD limits.

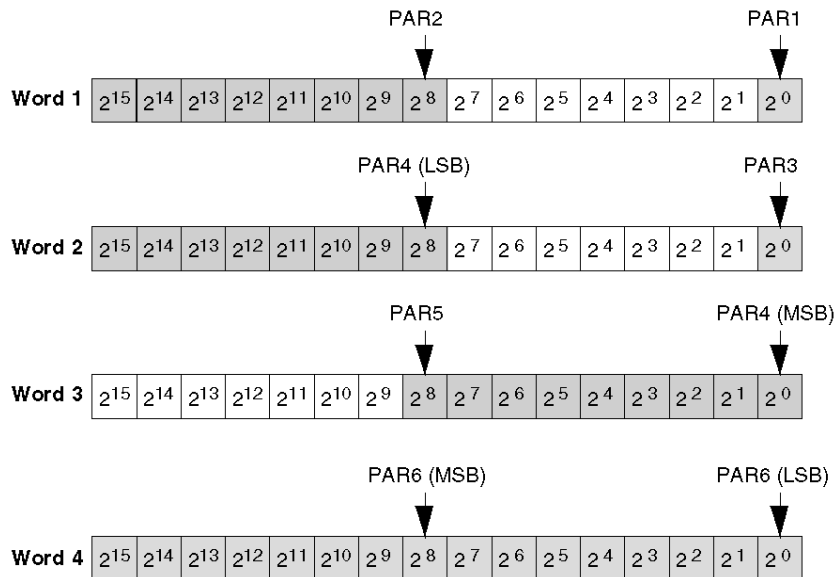
Example of a derived data type:

```

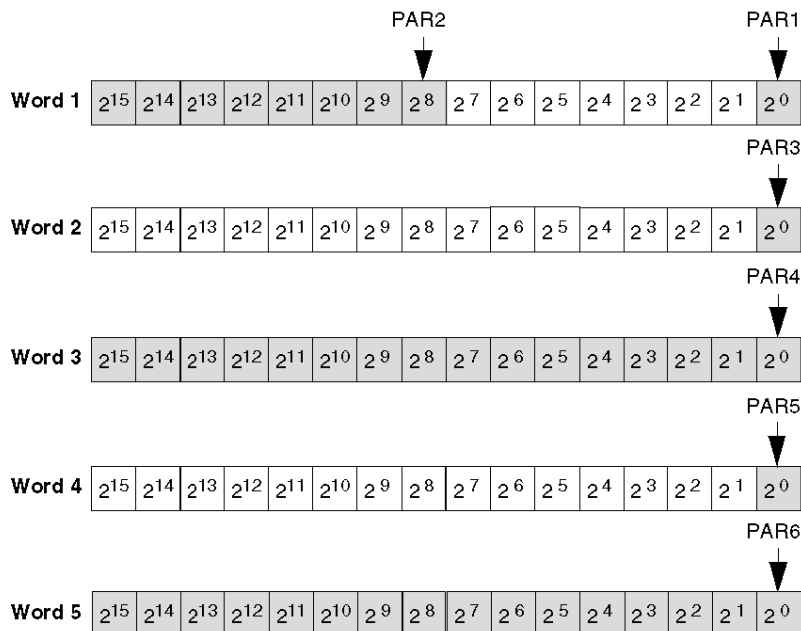
TYPE
  SKOE:
    STRUCT
      PAR1: BOOL;
      PAR2: BYTE;
      PAR3: BOOL;
      PAR4: WORD;
      PAR5: BOOL;
      PAR6: WORD;
    END_STRUCT;
END_TYPE

```

The derived data types are stored in the state RAM when using **Concept**:



The same derived data types are stored in the state RAM when using **Unity Pro**:



Timer, Date, Battery Monitoring

Timer address, date/time of day and the battery monitoring can no longer be assigned to the State RAM with Unity Pro. All required information can be accessed via the control panel.

When Concept is converted to Unity Pro, DFBs are created which can be simulated in Unity Pro without further manual modifications of these functionalities.

NOTE: The Concept Timer Register is 16 bits long and has an accuracy of 10 ms. The equivalent system word %SW18 in Unity Pro is 32 bits long and has an accuracy of 100 ms. If this accuracy is not sufficient, the FREERUN function from the System library can be used, which delivers accuracy of up to 1 ms.

NOTE: When dealing with days of the week, the value 1 corresponds to **Sunday** in Concept and **Monday** in Unity Pro.

Quantum Diagnostics Words

In Unity, the diagnostics words are specified to be a certain number:

- Local I/O: 16 Words
- RIO I/O: 16 Words
- DIO I/O: 16 Words

In Concept it was also possible to specify a smaller number of diagnostics words for the individual I/Os.

Keep this difference in mind, since it can cause problems.

Topological Addresses

The topological addresses are assigned so that if the hardware configuration remains the same, they occupy the same I/O connections as they were assigned in Concept.

The user sees the hardware addresses in Unity Pro that they are using, without having to carry out the intermediate step via the State RAM.

Located Variable

Located BOOL variables in Concept are converted to EBOOL variables in Unity Pro.

Unity Pro provides this new EBOOL variable for the detection of transitions (edges). This "Elementary BOOL type" is used for %Ix, %Mx and unlocated variables.

EBOOL variables can be forced.

The EBOOL variable provides three informational items:

- Current value
- Historical value
- Force information.

Only the current value can be accessed, the other values can only be accessed via product specific functions.

Longer Cycle Time via EBOOL

In Unity, as opposed to Concept, the edge and force information is updated from EBOOL variables during program runtime.

For this reason on the Quantum CPU 434, CPU 534 and CPU 311 platforms the assignment of EBOOL variables is only **half as fast** as the assignment of BOOL variables.

NOTE: If you need variables in the signal memory, use BOOL variables and assign them to the memory area %MW (e.g. BoolVar : BOOL AT %MW10). Otherwise use unlocated BOOL variables.

Constants

Constants in Concept are converted to write-protected variables in Unity Pro.

Unity Pro does not provide constants. Comparable functionality is achieved using write-protected variables.

%Mx Register

In Concept, the 0x registers are **not buffered**. They are reset to zero with every warm restart.

In Unity Pro, the %Mx registers are **buffered** ("RETENTIVE", "VAR_RETAIN"), i.e. Conform to IEC.

Do not use the possibility to set the 0x register to zero on every warm restart if you use a project in Concept that you want to convert to Unity Pro.

NOTE: If you require non-buffered behavior, define the warm restart event with the SYSSTATE function block and explicitly copy the value 0 (zero) to the %Mx register.

Forced Outputs (%M)

WARNING

UNEXPECTED SYSTEM BEHAVIOR

Do not relay on the Memory Protect switch.

The behavior of forced outputs (%M) between Modsoft/Proworx/Concept and Unity Pro has changed.

- With Modsoft/ProWORX/Concept you **cannot** force outputs when the Memory Protect switch of the Quantum CPU is set to the "On" position.
- With Unity Pro you **can** force outputs even when the Memory Protect switch of the Quantum CPU is set to the "On" position.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

⚠ WARNING

UNEXPECTED SYSTEM BEHAVIOR

Reforce the outputs following a cold start.

The behavior of forced outputs (%M) between Modsoft/Proworx/Concept and Unity Pro has changed.

- With Modsoft/ProWORX/Concept, forced outputs **maintain** their values following a cold start.
- With Unity Pro, forced outputs **lose** their values following a cold start.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Quantum Remote I/O Control

In Concept, only LL984 sections can be assigned I/O stations (Drops). This is not possible in Concept projects with IEC conforming sections (FBD, LD, SFC, IL, ST).

Unity Pro offers this option, in which a logic is recreated in accordance with LL984. This logic must be entered manually, however.

Example of a section processing order in Unity Pro:

Section n-2

Section n-1

RIO call (u,v,w)

Section n

Section n+1

RIO call (u+1,w,x)

Section n+2

RIO call (u+2,x,y)

RIO (x,y,z) is the explicit I/O call here:

- Write the outputs to the I/O station x.
- Wait at the inputs of the I/O station y.
- Prepare the inputs of the I/O station z.

NOTE: Take these new settings into consideration when structuring your project.

Setting Variables Cyclically

Unlocated variables **cannot** be set cyclically in Unity Pro. (It is possible in Concept).

If you need to set variables cyclically in your project, you should use located variables.

%Mx/%1x registers (EBOOL) can be forced.

%MWx/%IWx registers can be set cyclically (only numerical values).

EFBs

General

The following options are available for converting Concept EFBs to Unity Pro:

- The EFBs are also supported in Unity Pro; they are mapped on a one to one basis.
- The EFBs are **no longer** supported in Unity Pro.
Instead of EFBs appropriate DFBs are placed in the application. The functionality remains unaffected by this.
- The EFBs are **no longer** supported by Unity Pro.
Instead of EFBs, DFBs with no programmatic content are placed in the application. These DFBs contain all the Concept parameters.
An error message is displayed that says that the programmatic content for these DFBs must still be created.

Generic EFs

There are only a few generic elementary functions in Concept (EFs) e.g. MOVE, SEL, MUX. With many other functions, the elementary data type is added to the name of the function.

In Unity Pro, many of these functions are used without the elementary data type added to the name (as defined in IEC 61131). Therefore, the converter removes the added data type from the name of the function.

In some cases, the use of generic functions in Unity Pro will lead to analytic errors. In these cases, disable the **Generate Generic EFs** check box.

Open the **Conversion Settings** tab via **Tools** → **Options** in Unity Pro to enable/disable the **Generate Generic EFs** check box before converting.

- When this checkbox is enabled, the converter removes the added data type from the name of the function.
- When this checkbox is not enabled, the converter will leave the added data type in the name of the function.

DIAGNO Library

When converting Concept to Unity Pro for all DIAGNO blocks the station parameter is omitted.

The following EFBs from the DIAGNO library in Concept are converted to empty DFBs in Unity Pro.

- ACT_DIA
- XACT_DIA
- ERR2HMI
- ERRMSG

NOTE: These DFBs, created in Unity Pro have all the Concept parameters but no programmatic content. An error message is displayed that says that the programmatic content for these DFBs must still be created.

During the program creation in Unity Pro replace the DFBs ACT_DIA, and XACT_DIA with the DFB XACT.

For all DIAGNO blocks which can be extended in Concept (D_PRE, D_GRP ...), the extensible inputs (IN1 ... INx) are **gathered together in one** input. This is implemented using a nested logic AND link. In the FBD language the AND block is positioned at the same location as the DIAGNO block by the converter. This overlap must be resolved manually by the user.

SYSTEM Library

The SKP_RST_SCT_FALSE and LOOPBACK EFBs cannot be used in Unity Pro.

FUZZY Library

The FUZZY library is not supported with the normal Unity Pro range but can be installed as an optional library.

HANDTABL Library

The HANDTABL library is no longer supported by Unity Pro.

EXPERTS Library

The following Concept EFBs are converted to DFBs in Unity Pro:

- ERT_TIME
- SIMTSX22
- EFBs from the EX family
- EFBs from the MVB family
- EFBs from the ULEX family

NOTE: These DFBs, created in Unity Pro have all the Concept parameters but no programmatic content. An error message is displayed that says that the programmatic content for these DFBs must still be created.

The data structures DPM_TIME and ERT_10_TTAG from the time stamp module 140 ERT 854 10 have been changed. The MS element was broken up into MS_LSB and MS_MSB. For more information about this, see *State RAM Assignment Using Derived Data Types*, [page 27](#).

Outputs which describe data structures must be assigned event variables using the (=>) assignment operator within the parameter brackets in the ST and IL languages. This happens automatically during conversion (from Unity 2.0 onwards). The functionality remains the same but the section of the program looks a little different.

EFBs that Use Time Functions

In Unity Pro, function components using Time functions (Timer, Diagnostic, Control Components) remain in RUN mode, even if the SPS is set to STOP mode.

CAUTION

UNEXPECTED BEHAVIOR OF THE CONTROL

Function components using Time functions behave differently in Unity Pro and Concept.
You must take these different behaviors into consideration during the conversion of Concept applications.

Failure to follow these instructions can result in injury or equipment damage.

Converted EFBs

During conversion, Unity Pro standardizes the EFB offer by grouping redundant EFBs. The respective EFBs are automatically converted and the project adjusted accordingly.

Renamed EFBs

The following diagnostics EFBs are renamed when converting Concept to Unity Pro:

Concept	Unity Pro
XACT	D_ACT
XREA_DIA	D_REA
XLOCK	D_LOCK
XGRP_DIA	D_GRP
XDYN_DIA	D_DYN
XPRES_DIA	D_PRE

The Quantum configuration EFB for the Backplane Expander 140 XBE 100 00 is renamed when converting Concept to Unity Pro:

Concept	Unity Pro
XBP	XBE

Programming Language SFC

General

For some programming languages there are restrictions to observe when converting a project from Concept to Unity Pro.

Parallel/Alternative Sequence

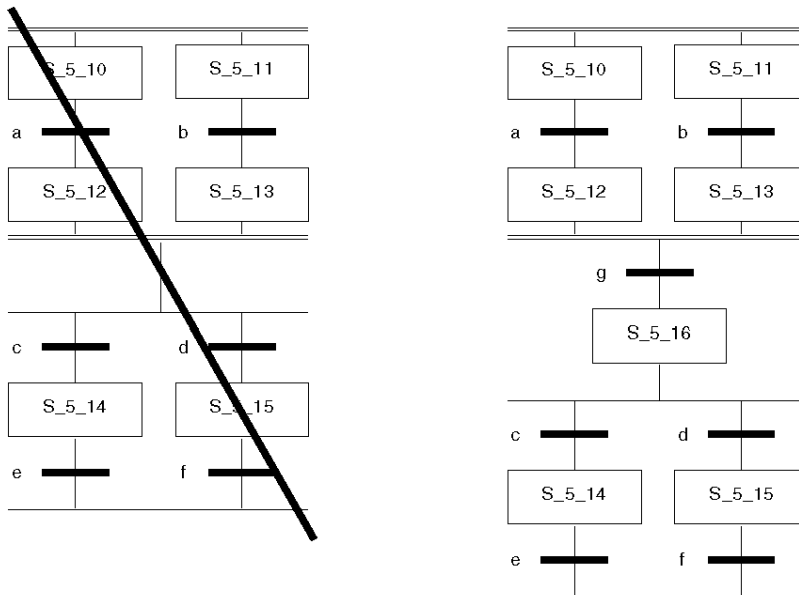
A parallel branch may not be directly followed by an alternative branch.

This type of sequence is not permitted according to IEC 1131.

Unity Pro does **not** support this type of sequence, although it is possible in Concept.

The converter transfers this type of project to Unity Pro, but manual modifications are subsequently required.

This problem can be solved by inserting an dummy step between the branches.



Programming Language LD

General

For some programming languages there are restrictions to observe when converting a project from Concept to Unity Pro.

Conversion of the picture

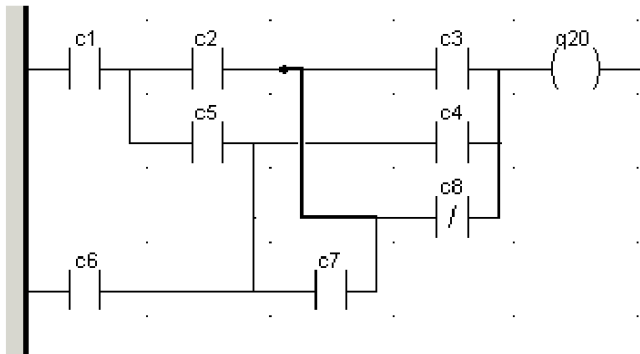
When converting a Concept project to Unity Pro, the ladder diagram LD Picture is also converted, which can lead to a restructuring of the picture.

Crossovers with connections between Boolean objects

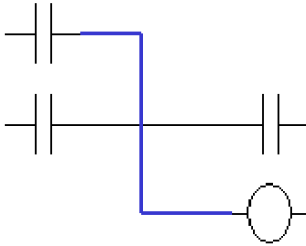
In Concept, FFB connections between Boolean objects may be edited.

This may result in crossovers.

Example of an FFB connection between Boolean objects (coils, contacts, horizontal, and vertical connections) in Concept:



Following the conversion from Concept to Unity, an FFB connection between Boolean objects may look like this:

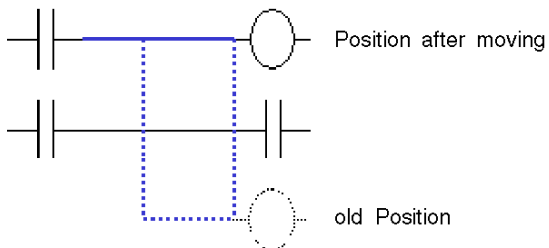


In the Unity LD Editor, such FFB connection may be:

- deleted,
- moved,
- copied and pasted.

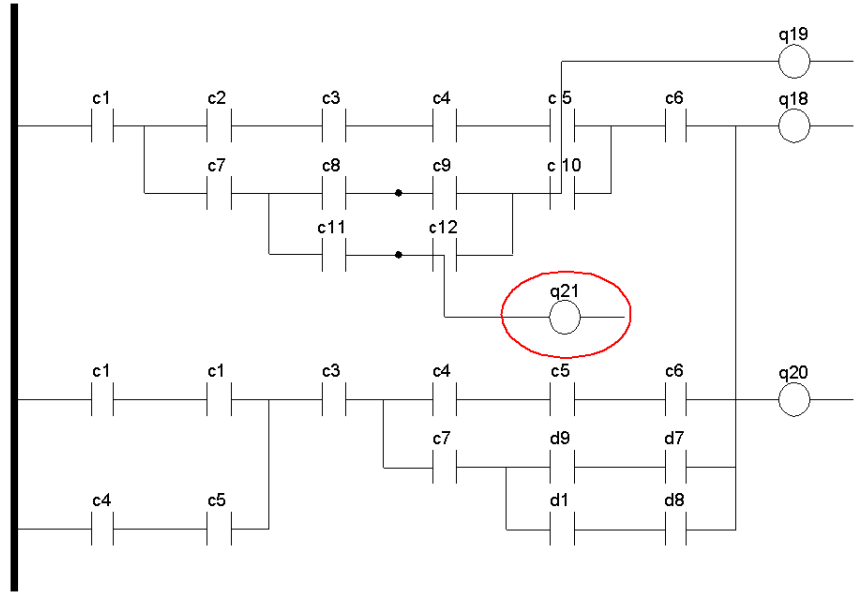
However, such FFB connection **cannot** be created in the Unity LD Editor.

The FFB connection will remain *after* moving in Unity.



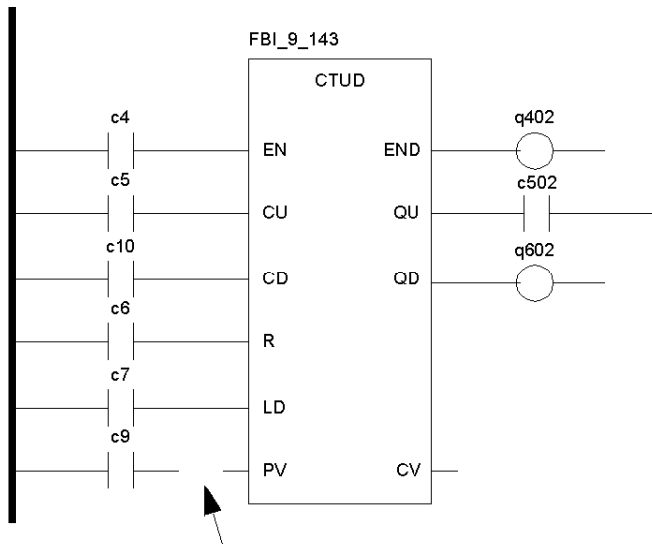
Connection to the right bus bar

A connection to the right bus bar is no longer required.



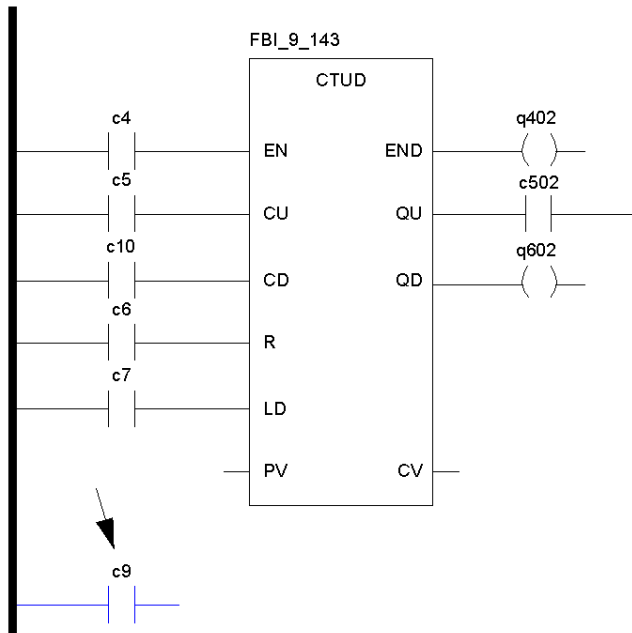
Automatically created connections

In Concept, the contact `c9` is not connected with INPUT PV of the component.



In Unity, the contact `c9` would automatically be connected with INPUT PV because both cells directly border in Concept.

During the conversion from Concept to Unity, the contact c_9 is therefore moved down to avoid the creation of an automatic connection in Unity.



Conversion of the output picture

During the conversion it is desired that the conversion of the picture from Concept to Unity is as exact as possible. To achieve this, the following rules are applied.

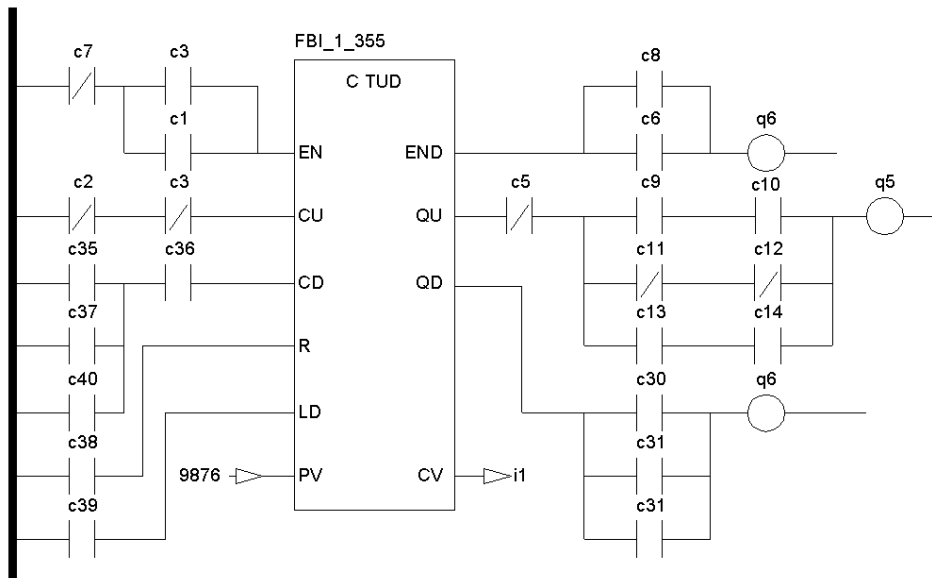
Rules for Object Positioning:

- The distance between two objects must be at least one cell.
- When two FFBs are connected, the minimum distance must equal the number of cells of the first FFB's width.
- The cells in Unity are smaller. If an FFB partially occupies another cell, an additional cell is required for the FFB.
- If an object (contact or coils) has a vertical connection (OR Link), this vertical connection will be located at the end of the cell of the object.
- An additional cell is required if:
 - a vertical connection (OR Link) with an INPUT FFB exists
 - the source FFB has output variables
 - the target FFB has input variables
- A coil may not be directly connected to the left bus bar.

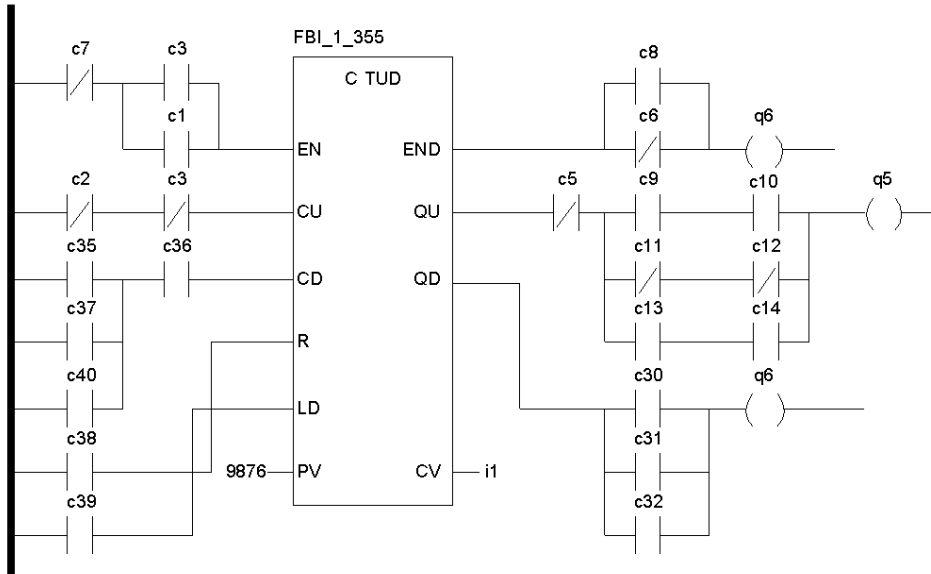
Rules for the conversion of FFB connections:

- FFB connections between variables/constants and FFBs will be ignored. In these cases, Unity will automatically create a connection.
- Purely horizontal FFB connections between objects that are not FFBs will be replaced with horizontal connections with multiple segments.
- When two OR objects are connected, a horizontal connection is first connected to the right side of the source OR object. An FFB connection will then be created between this horizontal connection and the target object. This occurs because the two OR objects would otherwise be combined during the import into Unity.
- Each point of the left bus bar can only be occupied by one connection.

Example of a picture in Concept:



The picture after the conversion to Unity.



The following actions were performed during the conversion according to the rules above:

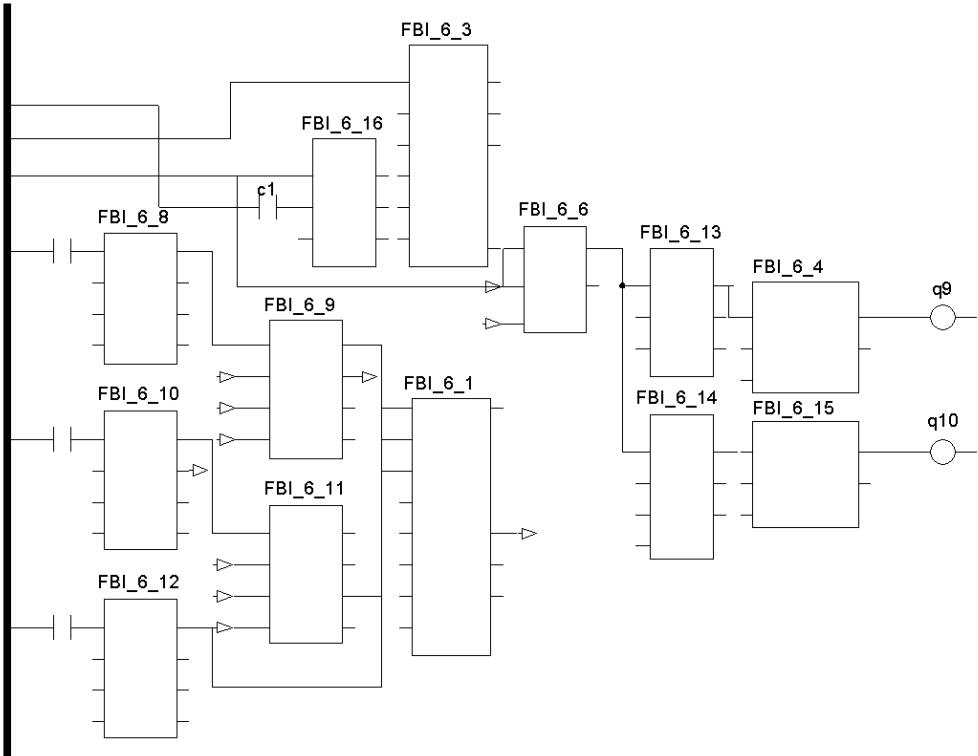
- The space that is occupied by the FFB was expanded to two columns.
- One column each was added at the Input and Output sides of the FFB.
- The connections between coils/contacts and the FFB were realized with FFB connections, not with horizontal connections with multiple segments.

Recognize and disconnect LD Networks

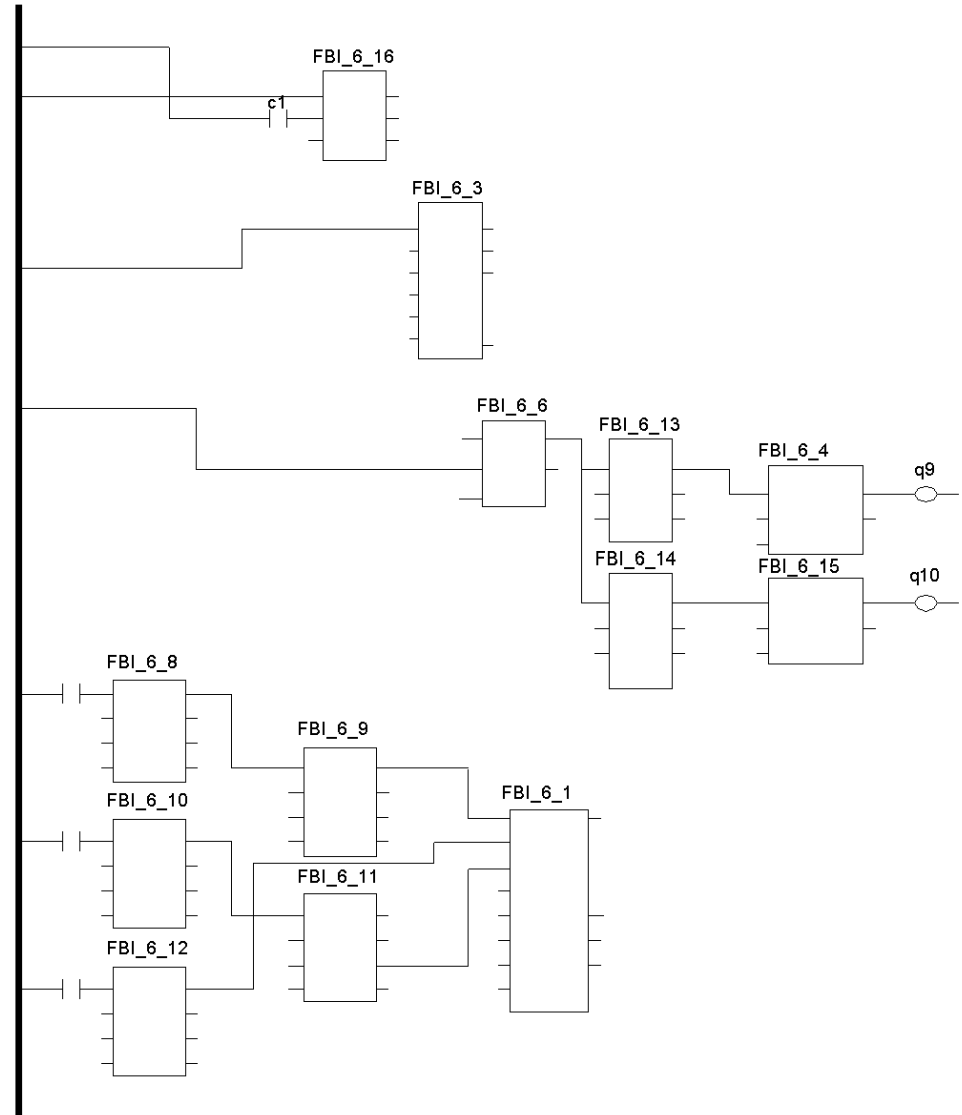
The converter must recognize networks in LD sections during the conversion. To achieve this, the following rules are applied:

- An LD Network is a group of objects that are connected with each other without any other connections to other objects (except the bus bar).
- The minimum distance is always applied to a complete column of a network. This means that if an object of a column requires a certain minimum distance, all other objects are also moved with a higher or equal horizontal position.
- If there are several networks in the same row in Concept, the following network will be moved vertically until it no longer occupies the same rows with the preceding network.
- To avoid undesired automatically created FFB connections, the space that is occupied by an FFB and its connection space will be checked for crossovers. In the event of crossovers the following objects will be moved horizontally.

Schematic diagram of an LD Network in Concept with crossovers



Schematic diagram of an LD Network after the conversion to Unity



Separate LD Networks

IEC LD sections contain many independent graphic areas (networks).

During the conversion of IEC LD sections, additional columns are added to the networks to avoid undesired automatically generated links in Unity Pro.

If the additionally inserted columns were to extend across the entire section, the original graphic would be modified too much. Therefore, the sections are divided in networks during the conversion and additional columns are only inserted for the associated network.

Inserting additional columns may cause a network to exceed its maximum section width and it is then wrapped into the next line.

If this causes networks to vertically overlap, the overlapping of the logic can lead to undesired automatic links in Unity Pro.

Open the **Conversion Settings** tab via **Tools** → **Options** in Unity Pro to enable/disable the **Separate LD Networks** check box before converting.

- When this checkbox is enabled, recognized networks are moved vertically, which prevents overlapping.
- When this checkbox is not enabled, recognized networks are not vertically moved. The original vertical arrangement of the graphics is maintained, but error messages may occur due to overlapping.

LD Column Break

Inserting additional columns may cause a network to exceed its maximum section width and it is then wrapped into the next line.

Open the **Conversion Settings** tab via **Tools** → **Options** in Unity Pro to edit the **LD Column Break** option before converting.

The number entered here determines the column after which a network is wrapped to the next column.

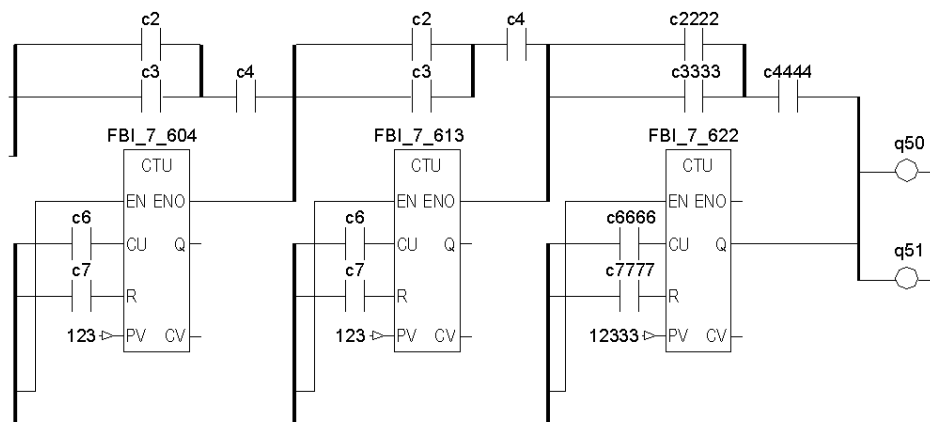
Wrapping networks that are too wide

Since the width of the networks is expanded during the conversion the maximum section width may be exceeded.

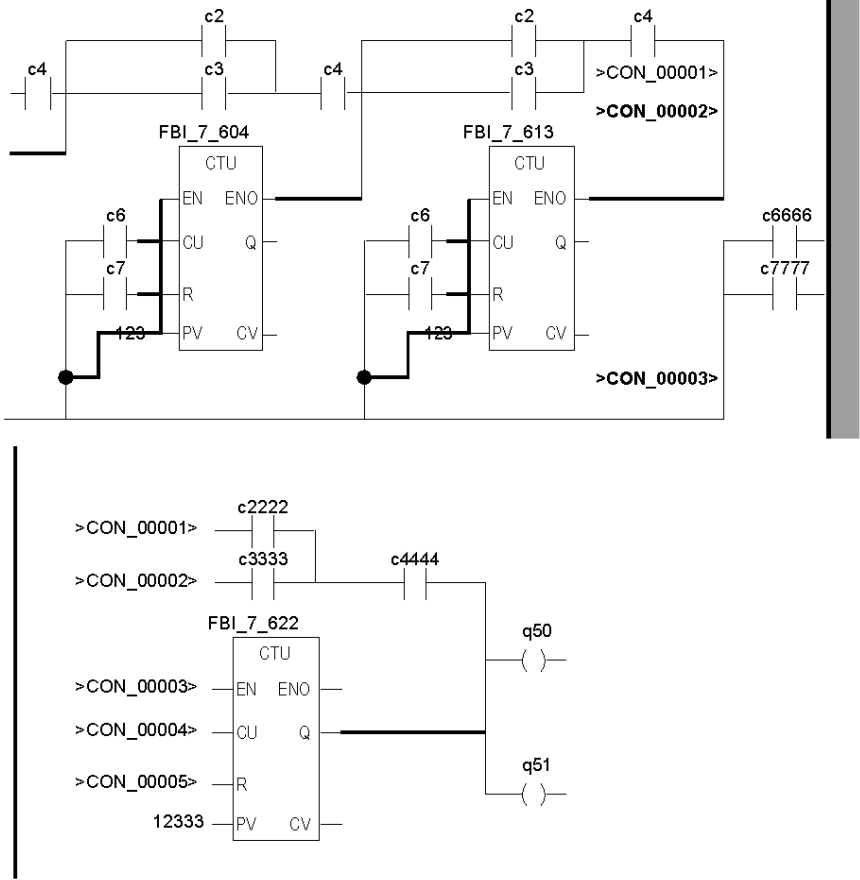
To show the network that is now too wide, the part of the network that reaches beyond the far right edge of the section will be shown in a new row.

The connections are shown as connectors.

Example of an LD Network in Concept.



The wrapped LD Network after the conversion to Unity.



Objects to recognize transitions

The different ways of handling ladder diagram LD objects in Concept (calling an FB) and in Unity Pro (system call) makes the use of State RAM variables (0x/1x register) necessary.

Because of the requirement that several write accesses to the 0x/1x register are possible during a cyclical sweep, there can be differing Online behavior between Concept and Unity Pro.

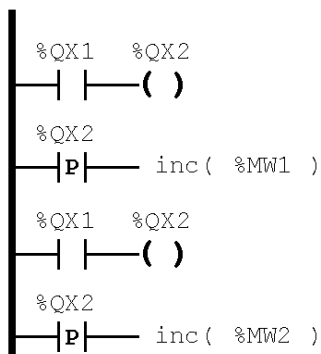
The objects affected are:

- Contact to recognize positive transitions
- Contact to recognize negative transitions

In **Concept** the "Old Value" to recognize a transition will only be updated once per cycle.

In **Unity Pro** the "Old Value" will be updated during every write access.

Example:



Concept: Switch %QX1 from 0 -> 1 and the value of %MW1 **and** %MW2 increase.

Unity Pro: Switch %QX1 from 0 -> 1 and **only** the value of %MW1 increases.

NOTE: Use objects to recognize transitions with a certain variable only once per cycle.

Also see *Located Variable*, [page 29](#) and Unity Pro Reference Manual, Use of set and reset coils leads to edge loss (see *Unity Pro, Program Languages and Structure, Reference Manual*).

Macros

Macros (name begins with @) will be rejected by the converter because macros cannot be implemented in Unity. However, if you try to import an application with macros, the macros will be replaced with Dummy DFBs (indicated with the '~' in the application name).

Error messages regarding these Dummy DFBs will appear during the analysis of the project. To correct these errors, simply remove all DFBs that were created to replace macros.

Programming Language ST/IL

General

For some programming languages there are restrictions to observe when converting a project from Concept to Unity Pro.

Generic EFBs

Only call generic EFBs instances once.

Using Concept 2.2, assign the outputs directly after the EFB call of a variable.

Syntax with Concept 2.5

Only use the new syntax for Concept 2.5 (from Unity V2.0 onwards it is automatically converted).

Syntax with Concept 2.5:

```
GenEFB(in1:=x1, in2:=x2, out1=>x3, out2=>x4;
```

in1, in2, out1 and out2 are type ANY.

Generic EFBs in Concept

List of generic EFBs in Concept:

- COMM library
 - XXMIT
- CONT_CTL library
 - DEADTIME
- EXTENDED library
 - HYST
 - INDLIM
 - LIMD
 - SAH
- LIB984 library
 - FIFO
 - LIFO
 - R2T
 - SRCH
 - T2T
 - GET_3X
 - GET_4X
 - PUT_4X

Declaring EFBs

The declaration of EFBs in Unity Pro is found in the variables editor and no longer in the ST/IL sections as with Concept.

EFBs declared this way are no longer limited to only one section.

Programming Language LL984

General

For some programming languages there are restrictions to observe when converting a project from Concept to Unity Pro.

LL984 is now supported by Unity Pro

NOTE: With Unity Pro 6.0 the LL984 language is supported for Quantum PLCs (but not for Quantum Safety PLCs).

With Unity Pro 6.1 or later the LL984 language is supported for Modicon M340 PLCs with Modicon M340 firmware 2.4 or later.

Programming Language FBD

General

For some programming languages there are restrictions to observe when converting a project from Concept to Unity Pro.

Macros

When converting a Concept project to Unity Pro, sections created using macros are also converted.

These sections can also be manually copied and modified.

Chapter 3

Language Differences

Overview

This chapter contains information about language differences.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Functions Not Present in Unity	56
EFB Replaced by Function	57
FFBs Not Available For All Platforms	58
INOUT Parameters	62
Parameter Type Changed	63
ANY_ARRAY_WORD Parameters	64
Unique Naming required	65
Incomplete LD Generation	66
LD Execution Order Changed	67
Constants	71
Indices in ST and IL	72
Calculate with TIME and REAL	73
WORD Assignments to BOOL Arrays	74
Topological Address Overlapping	75
Substitute %QD by %MF	76
Structure Alignment Changed	77
Undefined Output on Disabled EFs	78
Variables at Empty Pins	80
The set action remains active, even when the associated step becomes inactive	81
SFC Section Retains its State When Performing an Online Modification	82
SFCCNTRL Function Block in Unity Behaves Different to Concept	83
Weekday Numbering	84
System Timer	85
Initial Values	86
Macros	88

Functions Not Present in Unity

DFB Wrapper

Functions from Concept that are not present in Unity get a DFB wrapper if they are called in ST sections (e.g., `WORD_AS_UDINT`). For example:

```
WAUD(* UDINT *) := WORD_AS_UDINT (LOW := WAUL, (* WORD *) HIGH := WAUH(*  
WORD *));
```

... looks like this after conversion:

```
WAUD(* UDINT *) := FBI_ST1_75_33 (LOW := WAUL, (* WORD *) HIGH := WAUH(*  
WORD *));
```

Manual Correction

`FBI_ST1_75_33` is the instance name of the provided DFB wrapper. However, the call is still invalid for the analyzer because the converter cannot yet do multi-object syntax corrections in ST. (Will be present in V2.0).

You must correct this manually to:

```
FBI_ST1_75_33 (LOW := WAUL, (* WORD *) HIGH := WAUH(* WORD *), OUT =>  
WAUD);
```

EFB Replaced by Function

DFB Wrapping

Some standard Concept EFBs are implemented in Unity as functions.

In such cases, a wrapping DFB is provided so that the original interface of the Concept EFB remains valid.

FFBs Not Available For All Platforms

Overview

The FFBs (functions/function blocks) listed below can only be used on Quantum platforms (except `SFC_RESTORE`, see table below).

If Modicon M340 is the target platform, these FFBs appear marked in red and indicated as "type error".

FFBs Not Available

Communication library

Family	FFB	Platform
Extended	CREAD_REG	Quantum only
	CWRITE_REG	
	MBP_MSTR	
	READ_REG	
	WRITE_REG	
	MODBUSP_ADDR	
	SYMAX_IP_ADDR	
	TCP_IP_ADDR	
	XXMIT	

I/O Management library

Family	FFB	Platform
Analog I/O Configuration	I_FILTER	Quantum only
	I_SET	
	O_FILTER	
	O_SET	

Family	FFB	Platform
Analog I/O Scaling	I_NORM	Quantum only
	I_NORM_WARN	
	I_PHYS	
	I_PHYS_WARN	
	I_RAW	
	I_RAWSIM	
	I_SCALE	
	I_SCALE_WARN	
	O_NORM	
	O_NORM_WARN	
	O_PHYS	
	O_PHYS_WARN	
	O_RAW	
	O_SCALE	
	O_SCALE_WARN	
Immediate I/O	IMIO_IN	Quantum only
	IMIO_OUT	

Family	FFB	Platform
Quantum I/O Configuration	ACI030	Quantum only
	ACI040	
	ACO020	
	ACO130	
	AI1330	
	AI133010	
	AIO330	
	AMM090	
	ARI030	
	ATI030	
	AVI030	
	AVO020	
	DROP	
	ERT_854_10	
	NOGSTATUS	
	QUANTUM	
	XBE	
	XDROP	

Motion library

Family	FFB	Platform
MMF Start	CFG_CP_F	Quantum only
	CFG_CP_V	
	CFG_CS	
	CFG_FS	
	CFG_IA	
	CFG_RA	
	CFG_SA	
	DRV_DNLD	
	DRV_UPLD	
	IDN_CHK	
	IDN_XFER	
	MMF_BITS	
	MMF_ESUB	
	MMF_INDX	
	MMF_JOG	
	MMF_MOVE	
	MMF_RST	
	MMF_SUB	
	MMF_USUB	

Obsolete Lib library

Family	FFB	Platform
Extensions/Compatibility	GET_3X	Quantum only
	IEC_BMDI	

System library

Family	FFB	Platform
SFC Management	SFC_RESTORE	Quantum and Premium, not for Modicon M340
Hot StandBy	HSBY_RD	Quantum only
	HSBY_ST	
	HSBY_WR	
	REV_XFER	

INOUT Parameters

Manual Correction

INOUT parameter syntax in ST (and IL) must be corrected manually. Examples are shown:

```
Ascii_FIFO_OUT (Pile := AscFifo_Mess); .....
```

```
AscFifo_Out := Ascii_FIFO_OUT.DataOut;
```

... is manually corrected to:

```
Ascii_FIFO_OUT (Pile := AscFifo_Mess, DataOut => AscFifo_Out);
```

Output Parameters

INOUT parameters in ST sections that were output parameters in Concept (e.g., `DataOut` of FIFO) must be moved manually in ST and IL to the parameters inside parentheses associated with the call.

If INOUT parameters that were outputs only in Concept are connected only to a link at the output side, they must get a manually declared variable at the input side as well. The link must be deleted if it is not connected to another IN/OUT variable. Targets of the deleted link must be assigned to the manually declared variable.

This is done automatically in V2.0.

Change of Variable Type

The converter changes the type of direct variables at INOUT parameters of communication blocks to `ARRAY[0..0] OF WORD`.

This must be corrected manually to correspond to the size of the array.

Parameter Type Changed

Change

The parameter type has been changed from type `WORD` to an array of located words.

Explanation

Unity Comm EFBs no longer accept a single `WORD` address for the communication field because more than one `WORD` is written. So the converter introduces an artificial array (shown in the conversion report) that can be reached from the project tree through the appropriate hyperlink:

```
"For var WORD1 type ARRAY[0..0] OF WORD generated"
```

The array has a single word size because the converter can not determine its size. The user, therefore, needs to manually configure the correct array size.

ANY_ARRAY_WORD Parameters

Error Message

In EF/EFB pins that have the type `WORD` in Concept and have been changed to `ANY_ARRAY_WORD` in Unity, "Cannot import variables" will be the reported type. Such pins usually have a single register address as a formal parameter in Concept, but it is actually used to point to an array of words for which the size has not been explicitly declared.

Change of Parameter Type

In Unity, an array of words has to be declared for this purpose. This is why the converter changes the type to `ARRAY[0..0] OF WORD`.

However, the converter cannot determine the required size because a size declaration is absent in the Concept application. Therefore, the converter defines one data element, `[0..0]`, as a replacement for the original variable.

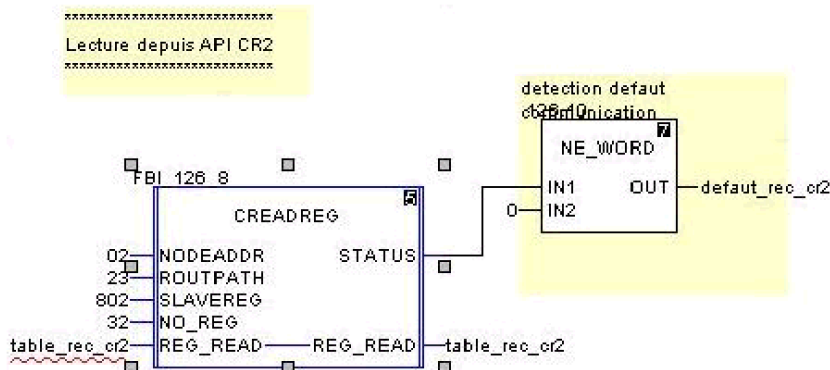
It is up to the user to replace this default range of one element with the number of elements required by the application.

Redefine Back to a One-Dimensional WORD Array

In case the application defined data structures that are mapped to registers that describe the data to be worked with, significant work to redefine this back to a one-dimensional `WORD` array is required. However, this is necessary for Unity V1.0, for example:

```
{Echanges_CR2 : [MAST]} : (r: 42, c: 7) E1092 data types do not match
('CREADREG.REG_READ:ANY_ARRAY_WORD' <-> 'table_rec_cr2:peer_Table')
```

Example:



The Unity converter V2.0 will change these EFB parameter types to `ANY`, avoiding this problem.

Unique Naming required

Unique name

In Concept applications, section names can have the same name as a DDT. That is not the case in Unity.

The converter checks section names to see if they are redundant of DDT names. If so, the converter appends "_Sect" to the section name.

Incomplete LD Generation

LD Generation Not Done Completely

In some cases, LD generation cannot be completed. This can happen when the algorithm allows an object that requires the same position as an existing object. In these cases, the pre-existing object is overwritten.

Messages are issued to make you aware of this:

```
{SAFETY_INTERLOCKS_PL3 : [MAST]} :  
(r: 8, c: 3) E1189 converter error: 'Overwrite happened when generating  
LD network - see report'  
{SAFETY_INTERLOCKS_PL3 : [MAST]} : (r: 8, c: 3) E1002 syntax error
```

Details in Conversion Report

In the conversion report, which may be opened after being imported through the hyperlink in the project tree, some additional detail about the message is given:

```
09:29:05.953 > Error: LD Object PTFDTP1_ENABLED with type coil  
overwritten
```

The user should compare the conversion result to a printout of the original section and correct the converted section accordingly.

Label or Link Views

If in LD networks, some links seem to have disappeared, it's because for a better reading they have been replaced by labels, to see the links, make a right click and select the **Show as link** option.

LD Execution Order Changed

Different Execution Orders

NOTE: Unity's LD execution order can differ from Concept's. In Unity, one LD network can be completed before the next is started.

The converter follows the Concept execution order in graphical positioning, making the original order visible to the user. However, since Unity calculates the order anew (without the possibility of forcing it from the converter), there can be execution order discrepancies.

Generate ConvError Hints

Open the **Conversion Settings** tab via **Tools** → **Options** in Unity Pro to enable/disable the **Generate ConvError Hints** check box before converting.

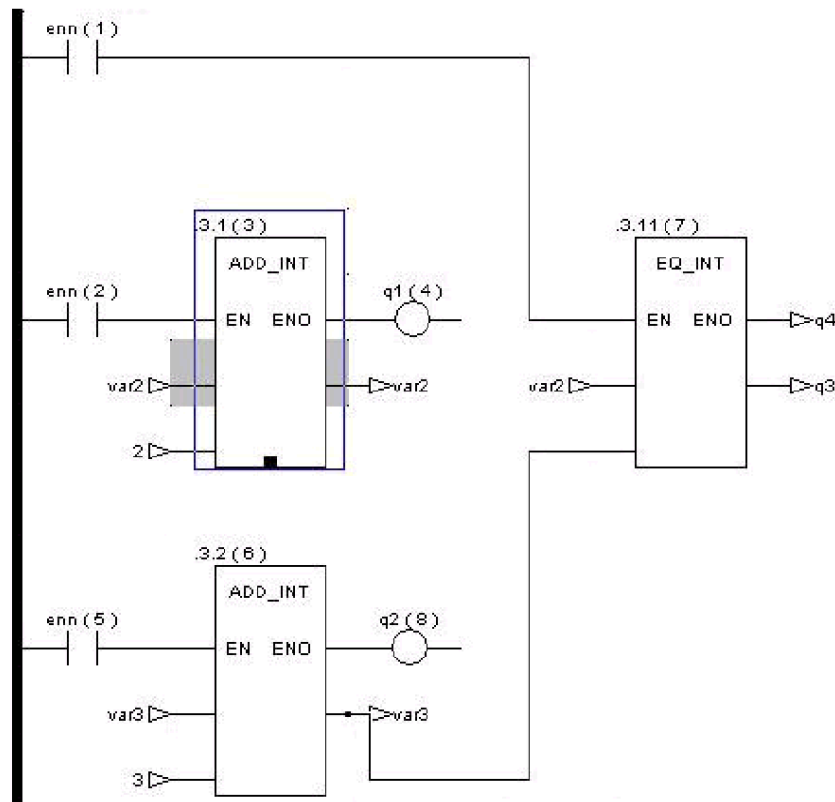
- When this checkbox is enabled, **ConvError** objects are generated in the LD programs during the conversion to draw attention to special issues.
- When this checkbox is not enabled, **no ConvError** objects will be generated during the conversion.

Concept

When analyzing in Concept, the execution order is calculated. The result is shown in parentheses after the instance names in this image.

The selected block is executed in the middle of the other network, even though it has no direct connection to it. Concept calculates the execution order from the block position.

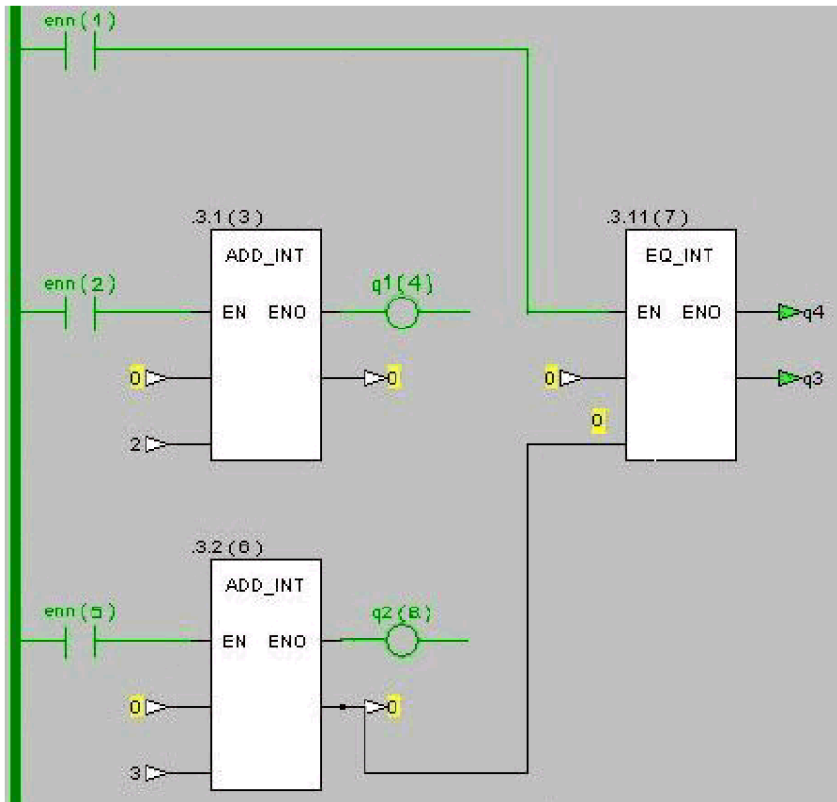
This is the original section as it appears in Concept:



The used variables are initialized in a way that the result of the comparator EQ_INT becomes "true" after execution of the first cycle in Concept:

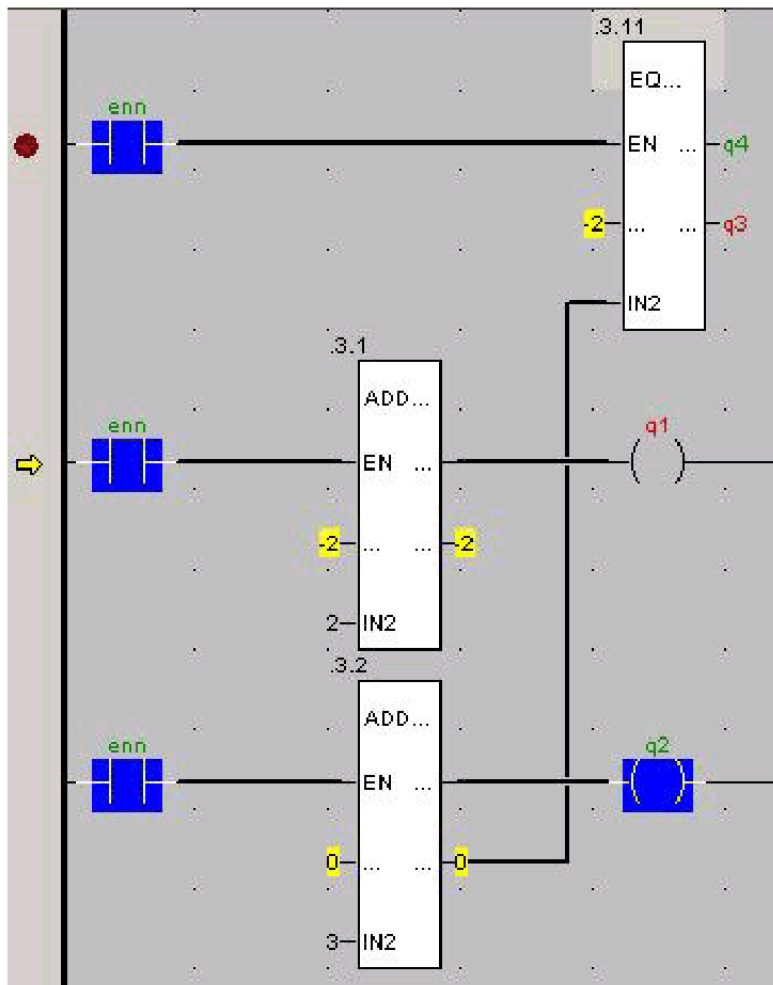
var2	INT	-2
var3	INT	-3

Testing execution in single cycle mode in Concept shows the expected result. The comparator result becomes "true" after the first cycle:



Unity

The converted network reflects the Concept execution order in the graphical position of the blocks:



The image also shows the execution status stopped at a breakpoint in the first cycle. The comparator `EQ_INT` is already executed and will not deliver a "true" result because the first `ADD_INT` integrator block is executed **after** it.

Solution

Replace the connection via a variable by a link to achieve the same result as in Concept.

Constants

Losing the Read-Only Behavior

Constants are not accepted as private DFB variables. Therefore, they are converted to initialized variables in DFBs, in this way losing the read-only behavior.

Indices in ST and IL

High Resolution

In addition to `INT` now `DINT` will be allowed as array index type in all areas of Unity Pro, but with limited value ranges.

For `DINT` the index may only contain `INT` values (`-32768 ... 32767`).

Calculate with **TIME** and **REAL**

Manual Correction

When **TIME** and **REAL** variables are multiplied in ST, **REAL_TO_DINT** must be inserted into the **REAL** variable manually.

WORD Assignments to BOOL Arrays

Manual Correction

Assignments of HEX WORDS to complete Bool arrays sent to Word registers are possible in Concept, but not in Unity. A manual correction must be done, for example:

```
('AR2_BOOL[0]:BOOL'<->'16#0100:DINT')
```

```
('AR2_BYTE[0]:BYTE'<->'16#55AA:DINT')
```

```
('AR2_BYTE[0]:BYTE'<->'16#AA55:DINT')
```

Solution

The ST code must be changed to single-component assignments.

The hex word must be split into single bits:

```
AR2_BOOL[17] := true;
```

Topological Address Overlapping

Same Topological Address

In Unity, you are warned (during application analysis) if the same topological address is assigned to multiple variables.

Substitute %QD by %MF

Introduction

Variables that are directly addressed in Concept with %QD can be initialized floating point constants or dual word constants.

When mainly floating point constants appear, the **Substitute %QD by %MF** checkbox should be enabled.

Conversion Settings

Open the **Conversion Settings** tab via **Tools** → **Options** in Unity Pro to enable/disable the **Substitute %QD by %MF** check box before converting.

- When this checkbox is enabled, %QD variables are converted to %MF variables.
- When this checkbox is not enabled, %QD variables are converted to MW variables.

Structure Alignment Changed

DPM_Time Structure

Unity uses a 2-byte alignment for structures in contrast to Concept (1-Byte) to speed up the access to structure components. This affects system structures mapped to StateRam, because the same structures in Unity can be bigger including some byte gaps.

The concerned structure is `DPM_Time`, which has been redefined for Unity to re-map to the correct hardware addresses.

Concept's `DPM_Time` definition:

```
sync:  BOOL
ms:    WORD
...
```

Unity's `DPM_Time` definition:

```
sync:    BOOL
ms_lsb:  BYTE
ms_msb:  BYTE
...
```

Manual Correction

If an application that includes the `DPM_time` structure is converted, the analyze/build process will fail for the redefined structure components (in the above example, `ms_lsb`, `ms_msb`).

The user has to manually change the usage of these structure components in the application accordingly.

Undefined Output on Disabled EFs

Outputs of EFs Not Kept

In case the EN switches from `TRUE` to `FALSE`, the outputs of EFs from the previous cycle are not kept in Unity. This reduces the memory consumption in the PLC. This is different from EFBs, which keep their value from the previous cycle. Concept uses static links to latch the value from the previous cycle.

Execution Behavior Differs Significantly

If a Concept application relies on the outputs of EFs to keep their old values, the execution behavior in UNITY will differ significantly.

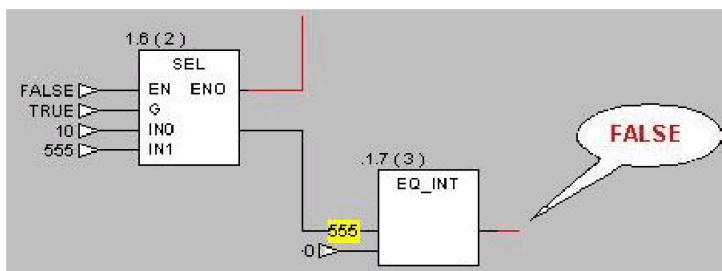
Manual Correction

The application has to be changed manually.

Links from outputs, which are assumed to keep their value, need to be replaced by variables. If the `EN` of an EF is set to false, the EF is not executed and a connected variable is not touched.

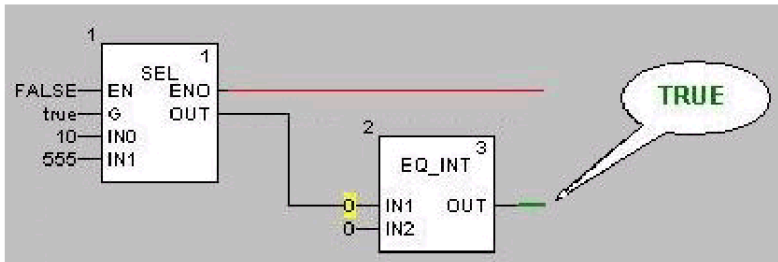
Concept

The output of the disabled `SEL` EF is kept and used as input for the `EQ_INT` function block:



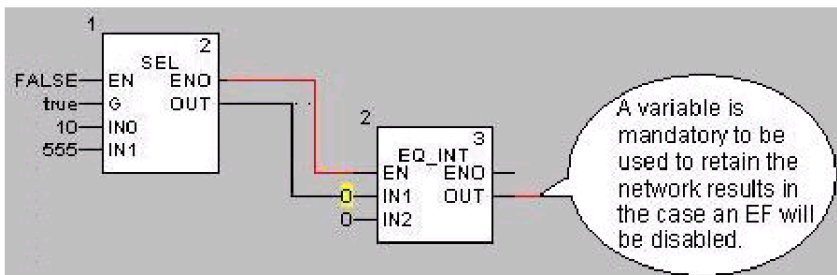
Unity

The output of the disabled SEL EF gets an undefined value, in this case 0. Therefore the output of EQ_INT function block has become true:



Solution

If the EN of the SEL is set to false, the ENO of the EQ_INT is also set to false, but the connected output variable keeps its value from the previous cycle:



NOTE: The use of a variable is mandatory to retain network results in case an EF becomes disabled.

Variables at Empty Pins

Introduction

In Unity Pro it is necessary to fill provided inputs and outputs for derived data types or I/O parameters (this is not necessary in Concept).

If these types are not generic, the converter will fill these initially empty inputs and outputs with variables created by the converter.

Conversion Settings

Open the **Conversion Settings** tab via **Tools** → **Options** in Unity Pro to enable/disable the **Variables at empty pins** check box before converting.

- When this checkbox is enabled, empty link points will be filled with variables created by the converter.
- When this checkbox is not enabled, empty link points will not be filled with variables created by the converter.

The set action remains active, even when the associated step becomes inactive

In Concept

The action could be reset in other sections.

In Unity Pro

The action only becomes inactive, when it is reset in another step of the current SFC section, using the qualifier R.

NOTE: To get identical behaviour, the solution is to use "Section" as Action instead of "Variable". In the section you can program SET(bit_xxx) ; and in the section outside the SFC you can program a RESET (bit_xxx) and it will work.

SFC Section Retains its State When Performing an Online Modification

Online Modifications Without Resetting

In Unity it is possible to do online modifications of an SFC chart without resetting it. The SFC chart retains its state and will continue the execution.

NOTE: In Concept, the online modification of an SFC chart usually results in the resetting of the chart.

SFCCNTRL Function Block in Unity Behaves Different to Concept

RESETSFC VS. INIT

In Concept the RESETSFC input of SFCCNTRL resets all action variables of the related SFC section.

In Unity the INIT input of SFCCNTRL (that has a similar function as RESETSFC input in Concept) only resets the action variables that have been set by the SFC step. Action variables, for example, set by user logic or the Animation Table will not be reset.

Weekday Numbering

Different Numbering

In Unity the numbering of weekdays is different than Concept:

Number	Unity	Concept
1	Monday	Sunday
7	Sunday	Saturday

SET_TOD / GET_TOD

Function blocks: `SET_TOD` and `GET_TOD` will be converted to Unity as DFBs, which work in both directions.

Because `SET_TOD` expects a "Concept" numbered weekday and translates it as a Unity coded value. Also the `GET_TOD` reads Unity value and returns to User the Concept value.

System Word %SW49

NOTE: We do not recommend that you mix `GET_TOD` and `SET_TOD` programming with the use of system words (e.g. `%SW49`) in the same application.

System Timer

Concept

Concept's system timer was located on a user-defined register word (16-bit) and incremented at 10 ms.

Unity

Unity provides an incremental timer with 100 ms updating (%SW18).

A 10 ms timer can be logically created using the `FREERUN` function (sec timer).

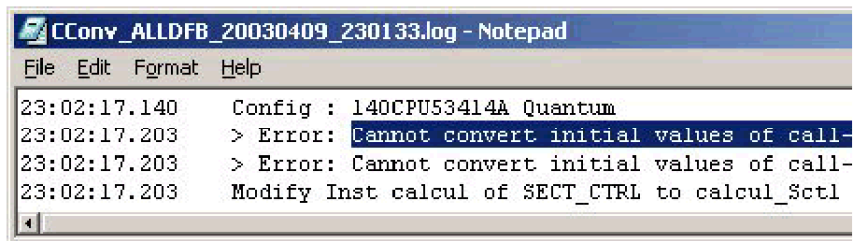
Initial Values

Definition of Initial Values

Concept allows the initial values on DFB pins of a structured array to be defined.

Unity forbids this option for pins of array type. This option is reserved for output pins of structure type.

The converter reflects this with the following error message in the conversion log:



Error: Cannot convert initial values of call-by-reference data (pin Add_PV.in1)

Pins to be Connected

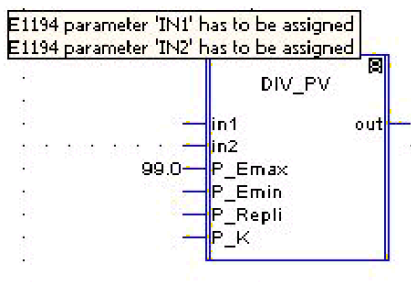
At the same time, Unity enforces pins of array type and input pins of structured type to be connected, which in this case leads to analysis errors:

```
{ALL:[MAST]}: (r:26, c:68) E1194 oarameter 'IN2' has to be assigned
{ALL:[MAST]}: (r:26, c:68) E1194 oarameter 'IN1' has to be assigned
```

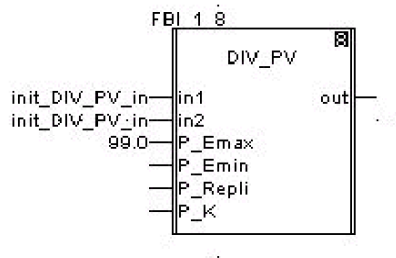
Solution

To solve this problem, create a variable of the pin's type and initialize it with the original values. Connect this constant to the appropriate pin of each DFB instance.

Example



Solution: Add initialized variable.



Macros

Macros Replaced by Dummy DFBs

Macros (name starting with @) are refused by the converter because Unity does not implement macros. However, if you try to import an application containing macros, they will be replaced by dummy DFBs (as indicated by the '~' character in the application name).

While analyzing the project, you will get error messages regarding these dummy DFBs. To correct these errors, simply remove all of the DFBs that were created as a replacements for macros.

AXx, EPARx Parameters

AXx and EPARx parameters in Concept's extensible motion blocks are automatically invoked with the newly required array instead of with Unity's formerly present extensible pins. Constants present at the Concept pins are also placed as initialization values to such arrays. However, variables and links must be attached manually with move blocks to these arrays.

Chapter 4

Possible application behavior change

Overview

This chapter contains information about possible application behavior change, when migrating from Concept to Unity Pro.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
General	90
Concept Behavior	91
IEC Demands	92
Unity Behavior	95
Consequences	97

General

Concept

In Concept and Unity Function Block interfaces are implemented with data structures (instance areas) collecting parameters, according to the standard IEC61131 which both systems refer to.

Function Block invocations refer to those data structures. However, Concept does not include output parameters into those instance areas. All parameters of DFB/EFBs (Elementary Function Blocks) are generally handled by reference, therefore output parameters are directly written to by the Function Block code by Concept. Unity DFB/EFBs buffer output parameters in the instance area, as the standard IEC61131 prescribes.

The Concept behavior was used to enable, for example, easy manual mode implementation of closed-loop-control function blocks.

If the output is written to, only once in a cycle, the behavior is the same in both systems. If output values are not written in all invocation cases, but are assigned by several Function Block instances, different behavior between both systems can result.

If the concerned variable is written to by some other control part prior to a Function Block having the same output parameter, invoked in a case where the Function Block does not write to the output, nothing changes in Concept, but in Unity the variable buffer value in the instance area resulting from a previous invocation is assigned to the output parameter.

To detect such cases, multiple assignments to elementary variables or derived data type components from Function Blocks are detected by the Concept converter, if the appropriate option is checked:

- Detected for Elementary and Derived Function Blocks.
- Works in DFB and Program Sections.
- Reports during conversion in the build tab of the output window with textual identification of the concerned locations.
- The same textual report appears in the Conversion report.
- Reports in FBD and LD sections with 'ConvError' blocks placed above the concerned Function Blocks.
- On Analysis messages appear in the Analyze/Build tab of the output window, which can be opened by double clicking and opening the concerned section and directly showing the concerned Function Block.

Using this report, the user can adapt this code to ensure common operation, e.g. by changing the DFB outputs to InOut parameters, which offer direct writing also in Unity.

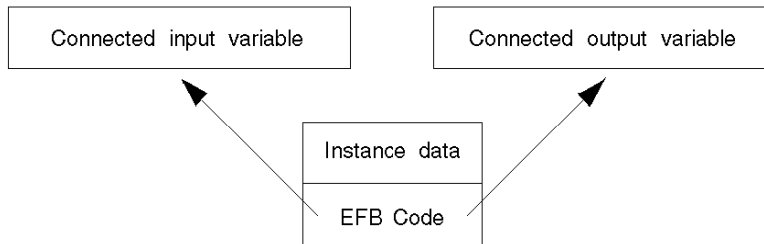
NOTE: If the application uses multi-assignment on EFB outputs, you should carefully read the following chapter to verify that the converted application works in the intended way (EFBs cannot be changed by the user, only new ones can be introduced).

Concept Behavior

Parameters are Handled by Reference

In Concept all function block parameters are handled by reference, means the blocks receives a pointer to the data of every function block pin and works directly on the connected variable.

Connected variables:



Function Block Code

Therefore in Concept it is up to the function block code to decide whether:

- to behave IEC compliant or
- to write input data or
- to read output data or
- not to write output data.

IEC Demands

Function Block

For the purposes of programmable controller programming languages, a function block is a program organization unit which, when executed, yields one or more values.

Multiple, named instances (copies) of a function block can be created.

Each instance shall have an associated identifier (the instance name), and a data structure containing its output and internal variables, and, depending on the implementation, values of or references to its input variables.

All the values of the output variables and the necessary internal variables of this data structure shall persist from one execution of the function block to the next.

Therefore, invocation of a function block with the same arguments (input variables) need not always yield the same output values.

Assignment of a Value

Assignment of a value to an output variable of a function block is not allowed except from within the function block.

The assignment of a value to the input of a function block is permitted only as part of the invocation of the function block.

Unassigned or unconnected inputs of a function block shall keep their initialized values or the values from the latest previous invocation, if any.

Allowable usage of function block inputs and outputs are summarized in table below, using the function block FF75 of type SR.

The examples are shown in the ST language.

Usage	Inside function block	Outside function block
Input read	IF IN1 THEN ...	Not allowed ^{1,2}
Input assignment	Not allowed ¹	FB_INST (IN1:=A, IN2:=B) ;
Output read	OUT := OUT AND NOT IN2;	C := FB_INST.OUT;
Output assignment	OUT := 1;	Not allowed ¹

Usage	Inside function block	Outside function block
In-out read	IF INOUT THEN ...	IF FB1.INOUT THEN...
In-out assignment	INOUT := OUT OR IN1; ³	FB_INST (INOUT:=D) ;
- Those usages listed as "not allowed" in this table could lead to implementation-dependent, unpredictable side effects. - Reading and writing of input, output and internal variables of a function block may be performed by the "communication function", "operator interface function", or the "programming, testing, and monitoring functions" defined in IEC 61131-1. - Modification within the function block of a variable declared in a VAR_IN_OUT block is permitted.		

EN and ENO in Function Blocks

For function blocks also an additional Boolean **EN** (Enable) input or **ENO** (Enable Out) output, or both, can be provided by the manufacturer or user according to the declarations.

When these variables are used, the execution of the operations defined by the function block shall be controlled according to the following rules:

1. If the value of **EN** is **FALSE** (0) when the function block instance is invoked, the assignments of actual values to the function block inputs may or may not be made in an implementation-dependent fashion, the operations defined by the function block body shall not be executed and the value of **ENO** shall be reset to **FALSE** (0) by the programmable controller system.
2. Otherwise, the value of **ENO** shall be set to **TRUE** (1) by the programmable controller system, the assignments of actual values to the function block inputs shall be made and the operations defined by the function block body shall be executed. These operations can include the assignment of a Boolean value to **ENO**.
3. If the **ENO** output is evaluated to **FALSE** (0), the values of the function block outputs (**VAR_OUTPUT**) keep their states from the previous invocation.

Not Connected EN Inputs

When **EN** inputs are left open the concerned blocks are not executed in Concept whereas they would be executed in Unity Pro.

To eliminate this difference the Concept Converter applies a constant boolean value of **FALSE** to not connected **EN** inputs. In this way achieving the same behavior as in Concept.

In-Out Variables

In-out variables are a special kind of variable used with program organization units (POUs), i.e., functions, function blocks and programs.

They do not represent any data directly but reference other data of the appropriate type. They are declared by use of the **VAR_IN_OUT** keyword. In-out variables may be read or written to.

Inside a POU, in-out variables allow access to the original instance of a variable instead of a local copy of the value contained in the variable.

Function Block Invocation

A function block invocation establishes values for the function block's input variables and causes execution of the program code corresponding to the function block body.

These values may be established graphically by connecting variables or the outputs of other functions or function blocks to the corresponding inputs, or textually by listing the value assignments to input variables.

If no value is established for a variable in the function block invocation, a default value is used.

Depending on the implementation, input variables may consist of the actual variable values, addresses at which to locate the actual variable values, or a combination of the two.

These values are always passed to the executing code in the data structure associated with the function block instance.

The results of function block execution are also returned in this data structure.

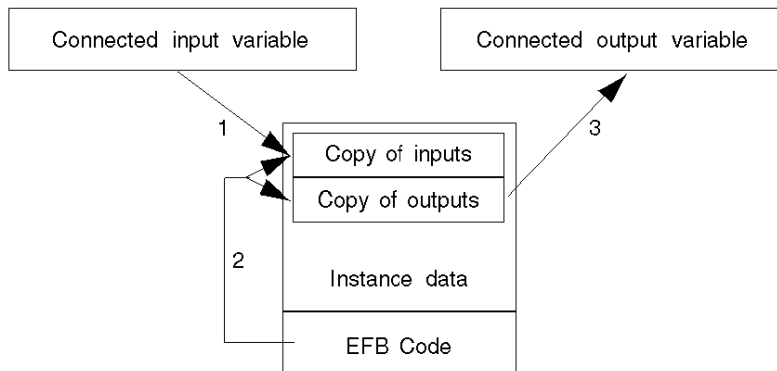
Hence, if the function block invocation is implemented as a procedure call, only a single argument - the address of the instance data structure - need be passed to the procedure for execution.

Unity Behavior

Changed Parameter Handling

To fulfill the IEC demands the normal EDT (Elementary Data Types) parameter handling was changed from Concept to Unity.

The following figure describes the actual implementation in Unity.



The EFBs no longer get pointers to their connected pin variables.

They always get the data by value.

In every scan the application code updates the copy of the input data in the instance data, before the function block is called (1).

The copy of the pin data is located in the instance data of the block and the function block code always works on the instance data (2).

After the function block code execution the application code copies the updated function block output data from the instance data to the connected output variables (3).

This is valid for all EDTs. Derived data types and more complex data types are treated still by reference in some cases.

Addressing Modes

The addressing mode of a Function Block element is directly linked to the type of the element.

The currently known addressing modes are:

- by value (VAL)
- by address (L-ADR)
- by address + Number of elements (L-ADR-LG)

Table with four columns and legend

-	EDT (Except STRING)	STRING	DDT Array	DDT Struct	ANY_ ARRAY	ANY...
Input parameter	VAL	L-ADR-LG	L-ADR-LG	L-ADR	L-ADR-LG	L-ADR-LG
Input_Output parameter	L-ADR ¹	L-ADR-LG	L-ADR-LG	L-ADR	L-ADR-LG	L-ADR-LG
Output parameter	VAL	VAL	L-ADR-LG	VAL	L-ADR-LG	L-ADR-LG
Public Variable	VAL	VAL	-	VAL	-	-
Private Variable	VAL	VAL	-	VAL	-	-
1 Except for BOOL type, the addressing mode is VAL.						

Function Block Invocation

The following rules must be taken into account while invoking a Function Block instance:

- All input_output parameters have to be filled
- All input parameters using the L-ADR or L-ADR-LG addressing modes have to be filled
- All output parameters using the L-ADR or L-ADR-LG addressing modes have to be filled

All other kind of parameters could be omitted while Function Block Instance invocation.

For input parameters, the following rules are applied (in the given order):

- The values of the previous invocation are used.
- If no previous invocation, the initial values are used.

Consequences

Potential Problems

 **WARNING**

UNEXPECTED APPLICATION BEHAVIOR

Take care when an application is migrating from Concept to Unity.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

NOTE:

Because of this architectural change, when an application is migrated from Concept to Unity you have to evaluate the consequences of the migration, and specially in the following cases:

- **Multi assignment of connected output variables:**

In Concept there are function blocks, mainly in the closed-loop-control area, which do not write their output values to the connected variables in special operating modes (manual mode). In these special modes it was possible to write the variables from other locations inside the application.

This will work in Unity only, if the variables are written after the function block call. If they are written before the function block call, the copy process from the instance data to the connected variables will overwrite this value with the old value from the instance data.
- **Controlling output variables by animation table or HMI:**

If a block doesn't write his outputs in special operating modes (like manual mode, see above), it was possible to modify the connected output variables by animation tables or HMI. This will no longer work in Unity, since the copy process from the instance data to the connected variables of the function block will overwrite the modified value with the old value from the instance data.

Changed EFB Layout

To avoid major problems, a lot of function blocks (mainly in the Motion and CLC area) were changed in their layout from Concept to Unity to ensure a correct mode of operation in the intended way for the function blocks.

The concerned pins were changed from type `OUT` to `IN/OUT`.

In nearly all cases the modification meets better the reality, since it is read from the concerned output pins and so they are in fact `IN/OUTs`.

The following tables summarize the EFBs, where at least one pin was changed from `OUT` to `IN/OUT` during migration from Concept to Unity.

Library CONT_CTL:

Family	Function Block	Concerned Pin
Controller	PI_B	OUT
	PIDFF	OUT
Output Processing	MS	OUT
Setpoint Management	SP_SEL	SP

Library Motion:

Family	Function Block	Concerned Pin
MMF Start	CFG_CP_F	MFB, CFG_BLK
	CFG_CP_V	MFB, CFG_BLK
	CFG_CS	MFB, CFG_BLK
	CFG_FS	MFB, CFG_BLK
	CFG_IA	MFB, CFG_BLK
	CFG_RA	MFB, CFG_BLK
	CFG_SA	MFB, CFG_BLK
	DRV_DNLD	MFB
	DRV_UPLD	MFB
	IDN_CHK	MFB
	IDN_XFER	MFB
	MMF_BITS	MFB
	MMF_ESUB	MFB
	MMF_INDX	MFB
	MMF_JOG	MFB
	MMF_MOVE	MFB
	MMF_RST	MFB
	MMF_SUB	MFB
	MMF_USUB	MFB

Library Obsolete Lib:

Family	Function Block	Concerned Pin
CLC_PRO	ALIM	Y
	COMP_PID	Y, YMAN_N, OFF_N, SP_CAS_N
	DERIV	Y
	INTEG	Y
	LAG	Y
	LAG2	Y
	LEAD_LAG	Y
	PD_OR_PI	Y
	PI	Y
	PID	Y
	PID_P	Y
	PIP	Y
	PPI	Y
	VLIM	Y
Extensions/Compatibility	R2T	OFF
	SRCH	INDEX
	T2T	OFF

Concept Converter Behavior

The Concept Converter normally handles the layout change in the following way, when a Concept application is imported into Unity:

- **Case 1: A variable is connected to the output pin in Concept:**
The Concept Converter keeps the variable at the output side of the IN/OUT pin and adds the variable additionally at the input side of the pin.
- **Case 2: A link is connected to the output pin in Concept:**
The Concept Converter removes the link, creates a new variable of the needed type and writes this new variable to the start and end position of the removed link. Additionally the variable is added to the input side of the pin.

Further Potential Problems

The following tables contain blocks, where also some consequences of the architectural change from Concept to Unity may arise in case of multi-assignment, because in Concept:

- The blocks do not write their listed output pin in case of errors inside the block.
- The blocks do not write their listed output pin in **COLD** or **WARM INIT** scan.
- The blocks write their listed output pin conditionally depending from internal mode of operation.

Library CONT_CTL:

Family	Function Block	Concerned Pin
Conditioning	DTIME	OUT
	SCALING	OUT
	TOTALIZER	OUT, INFO
Controller	AUTOTUNE	TRI, INFO
	PI_B	OUT_D, DEV
	PIDFF	OUT_D, INFO
	STEP2	DEV
	STEP3	DEV
Output Processing	MS	OUTD, STATUS
	MS_DB	OUTD, STATUS
	SPLRG	OUT1, OUT2
Setpoint Management	RAMP	SP
	RATIO	KACT, SP
	SP_SEL	LSP_MEM

Library I/O Management:

Family	Function Block	Concerned Pin
Analog I/O Configurationj	I_SET	CHANNEL
	O_SET	CHANNEL
Analog I/O Scaling	I_NORM_WARN	WARN
	I_PHYS_WARN	WARN
	I_SCALE_WARN	WARN
Quantum I/O Configurationj	ACI040	CHANNEL1..16
	ACO130	CHANNEL1..8
	AI1330	CHANNEL1..8, INTERNAL
	AI133010	CHANNEL1..8
	AIO330	CHANNEL1..8
	ARI030	CHANNEL1..8

Library Motion:

Family	Function Block	Concerned Pin
MMF Start	CFG_CP_F	Q, ERROR
	CFG_CP_V	Q, ERROR
	CFG_CS	Q, ERROR
	CFG_FS	Q, ERROR
	CFG_IA	Q, ERROR
	CFG_RA	Q, ERROR
	CFG_SA	Q, ERROR
	DRV_DNLD	Q, ERROR, IDN_CNT
	DRV_UPLD	Q, ERROR, REG_CNT, DATA_B, LK
	IDN_CHK	Q, ERROR, NOT_EQ
	IDN_XFER	Q, ERROR, OUT_RAW, OUTCONV
	MMF_ESUB	Q, ERROR, RET1, RET2, RET§
	MMF_INDX	Q, ERROR
	MMF_JOG	Q, ERROR
	MMF_MOVE	Q, ERROR
	MMF_RST	Q
	MMF_SUB	Q, ERROR, RET1, RET2, RET§
	MMF_USUB	Q, ERROR, RET1, RET2, RET§

Library Obsolete Lib:

Family	Function Block	Concerned Pin
CLC	DELAY	Y
	PI1	ERR
	PID1	ERR
	PIDP1	ERR
	THREE_STEP_CON1	ERR_EFF
	THREEPOINT_CON1	ERR_EFF
	TWOPOINT_CON1	ERR_EFF
CLC_PRO	COMP_PID	STATUS, ERR
	DEADTIME	Y
	FGEN	Y, N
	INTEG	STATUS
	PCON2	ERR_EFF
	PCON3	ERR_EFF
	PD_OR_PI	ERR, STATUS
	PDM	Y_POS, Y_NEG
	PI	ERR, STATUS
	PID	ERR, STATUS
	PID_P	ERR, STATUS
	PIP	ERR, SP2, STATUS
	PPI	ERR, SP2, STATUS
	PWM	Y_POS, Y_NEG
	QPWM	Y_POS, Y_NEG
	SCON3	ERR_FF
	VLIM	STATUS
Extensions/Compatibility	FIFO	EMPTY, FULL
	LIFO	EMPTY, FULL

NOTE: The pins were not changed, because in normal operation mode of the blocks this has no influence.

Chapter 5

The Conversion Process

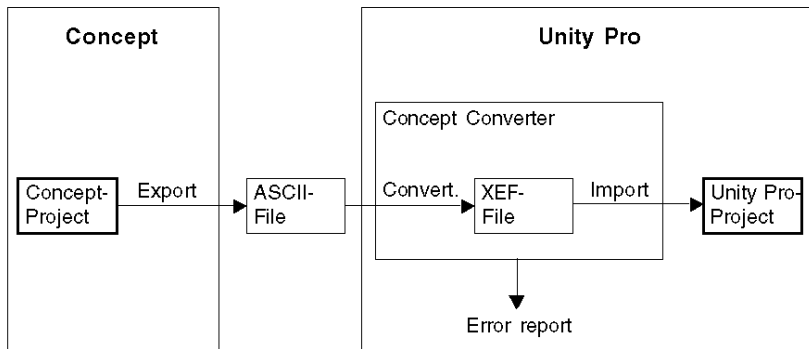
Conversion Process

General

A Concept project is exported from Concept and then converted automatically into a Unity Pro project using the Unity Pro Concept Converter.

Conversion Process

Representation of the conversion process:



Description of the conversion levels:

Level	Description
1	A project is exported from Concept. An ASCII file is created.
2	The Unity Pro Concept Converter is called. The ASCII file is converted into an XEF file.
3	The XEF file is imported into Unity Pro. A Unity Pro project is created.
4	The error report is checked. There must be no errors.
5	The project is now available in Unity Pro and can be generated and then loaded into a PLC or processed in Unity Pro.

Error Report and Analysis

Errors that occur during conversion are logged in an error report and displayed in an output window.

Substitute objects are used in place of objects that cannot be converted. The Unity Pro project can be analyzed using the main menu **Create → Analyze Project**. Subsequently messages are displayed in the output window to find the substitute objects.

The errors displayed in the output window must be corrected manually to ensure the Unity Pro project runs correctly.

Animation tables converting process

In Concept, RDE tables could have different names, the concept converter creates an animation table for each data editor reference file found in the project folder.

For Prowrx32 Data Watch windows, every data Watch window will result in an Animation table.

Chapter 6

Conversion Procedure

Overview

This chapter contains the procedures required to convert a Concept project into a Unity Pro project.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Exporting a Project from Concept	106
Importing a Project into Unity Pro	107
Missing Datatypes at the Beginning of the Import	108
Converting Only Parts of a Concept Application	109
Removing Accidentally Included Concept Macros	110
Initialization Values	111
If the convertedMomentum application contain more than one XMIT block	112

Exporting a Project from Concept

General

A Concept project that should be used in Unity Pro must first be exported from Concept. It is then possible to use the Unity Concept Converter to make the conversion to a Unity Pro project.

Export Project

Perform the following steps to export a project:

Step	Procedure
1	Start the Concept Converter program from the Concept program group.
2	Select File → Export... , to open the menu for selecting the export range.
3	Select the export range: ● Project with DFBs: All project information including the DFBs and data structures (derived data types) used in the project are exported. ● Project without DFBs: All project information including all data structures (derived data types), but not DFBs and macros, is exported. Result: The dialog box for selecting the files to be exported is opened.
4	Select the following file extension: ● Export projects: Select the extension .prj from the format list box.
5	Select the project and confirm using OK . Result: The project is stored in the current directory as an ASCII file (.asc).
6	End the Concept Converter program using File → Exit .

Importing a Project into Unity Pro

General

A Concept project that is to be used in Unity Pro must first be exported from Concept. It is then possible to use the Unity Concept Converter to make the conversion to a Unity Pro project.

Import Project

Carry out the following steps to convert and import a project:

Step	Procedure
1	Launch Unity Pro.
2	Open the project exported from Concept using File → Open . Select the data type CONCEPT PROJECTS (*.ASC). NOTE: Do not use the project with used DFBs (Re-Connect to Equal) option when creating the *.ASC file. Unity Pro cannot import the application if this option is used.
3	Result: The ASCII file is converted to Unity Pro source file format and imported automatically. Import errors and messages about objects that cannot be converted and have substitute objects in their place, are displayed in an output window.
4	Edit the errors and messages in the output window manually to ensure the Unity Pro project runs correctly.
5	To ensure that a project contains no more errors, select the menu command Build → Analyse Project again.

Missing Datatypes at the Beginning of the Import

General

If the dialog at the beginning of the import claims for unknown DDTs, search for local type declarations in this DDT and find out which of these are undefined.

Further, types unused but present in the opened *.asc file are reported as unknown in a dialog during import.

Concept System Data Types

This happens for Concept system data types, which are considered for Concept to be always presented and therefore not included in the export by Concept.

The Converter automatically includes the standard system data types of Concept individually, if they are needed. They are part of the converter command and include file CConv.xml present in the execution directory of Unity.

If the read-only-flag is removed, this file can be extended to include additional data types for user EFB libraries.

Such data type files beyond the global/local ones are placed in the lib subdirectory of Concept to be merged into the Concept application, but these data types do NOT appear in the Concept export file.

Concept *.dty Files

The V1.1 version of the Concept converter will have the capability to add Concept *.dty files, which are stored in the same directory as the *.asc file, to the converted application, as if their content were appearing inside the *.asc file itself.

Converting Only Parts of a Concept Application

General

The Concept converter is prepared to convert complete applications and parts of applications.

If only parts of a Concept application are needed either

- use a reduced application export with Concept (see sections below) or
- use the Conversion Wizard (see *Conversion with the Conversion Wizard*, [page 16](#)).

Single DFB

If a single DFB is needed, make a new application with 1 single section and place a call to the desired DFB into this section.

Export the application using the menu item **Export with used DFBs** in Concept.

Convert the resulting *.asc file in Unity Pro via **File →Open**.

Subset of Sections

To export a subset of sections use the **File →Export →Program: Section(s)** menu in Concept.

Select the source application and the desired sections and follow the user guidance to get a reduced application.

However, if the section contains references to SFC steps, Concept requires to export the referenced SFC section as well.

Convert the resulting *.sec file in Unity Pro with the Conversion Wizard via **Tools →Convert Partially**.

Subset of Variables

To export a subset of variables first open the **Variable Editor** in Concept and select the desired variables.

After that use the **File →Export →Variables: Text delimited** menu.

Convert the resulting *.txt file in Unity Pro with the Conversion Wizard via **Tools →Convert Partially**.

Animation Tables

If animation table files are present in the application export directory, the animation tables will be automatically included in the conversion result.

Removing Accidentally Included Concept Macros

General

If a Concept Macro has been included into the Concept export, this Macro is converted as if it were a DFB and appears in the project browser tree as a DFB.

Delete this DFB because Unity Pro does not support Macros.

Initialization Values

General

Initialization values are contained in Concept export in an array, describing the State RAM.

This array is converted in Unity Pro to clusters, i.e. that it is cut into contiguous sequences of non-zero values with single-zero values tolerated.

Each cluster is converted to an individual array with the names `LL_SRAMxxx`.

If the converted Momentum application contain more than one XMIT block

Rule

The 171 CBU 78090, 171 CBU 98090, or 171 CBU 98091 processor uses a much faster COM port speed than the legacy Momentum PLC.

As a result of this, the XMIT block might not function properly, if the converted Momentum application contain more than one XMIT block.

The fast COM port speed might result in the next XMIT block to execute before the prior XMIT block completes it's operation. If this happens, the block may appear to be functioning incorrectly.

To avoid this simultaneous execution of the XMIT blocks, a time delay or additional logic to test the DONE output of the XMIT block may need to be added to the block start or enable logic.

Part II

Blocks from Concept to Unity Pro

Overview

This part contains a description of the blocks which are not part of Unity Pro as standard.

However, if these blocks were used in Concept they are generated during the project conversion from Concept to Unity Pro in order to map the functionality configured in Concept into Unity Pro on a one to one basis.

What Is in This Part?

This part contains the following chapters:

Chapter	Chapter Name	Page
7	BYTE_TO_BIT_DFB: Type conversion	115
8	CREADREG: Continuous register reading	119
9	CWRITREG: Continuous register writing	127
10	DINT_AS_WORD_DFB: Type conversion	133
11	DIOSSTAT: Module function status (DIO)	135
12	GET_TOD: Reading the hardware clock (Time Of Day)	137
13	LIMIT_IND_DFB: Limit with indicator	141
14	LOOKUP_TABLE1_DFB: Traverse progression with 1st degree interpolation	145
15	PLCSTAT: PLC function status	151
16	READREG: Read register	167
17	RIOSSTAT: Module function status (RIO)	175
18	SET_TOD: Setting the hardware clock (Time Of Day)	179
19	WORD_AS_BYTE_DFB: Type conversion	183
20	WORD_TO_BIT_DFB: Type conversion	185
21	WRITEREG: Write register	189

Chapter 7

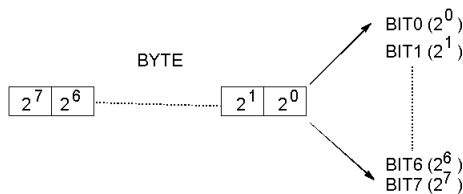
BYTE_TO_BIT_DFB: Type conversion

Description

Function description

This derived function block converts one input word from the `BYTE` data type to 8 output values of the `BOOL` data type.

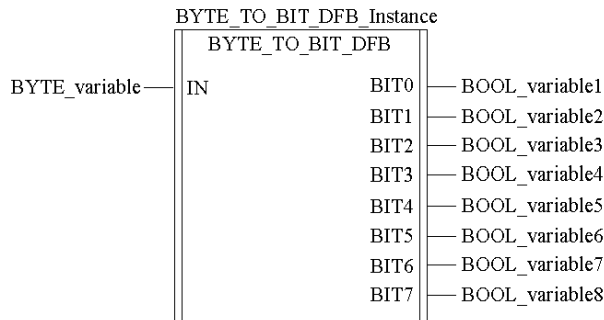
The individual bits of the byte at the input are assigned to the outputs according to the output names.



`EN` and `ENO` can be configured as additional parameters.

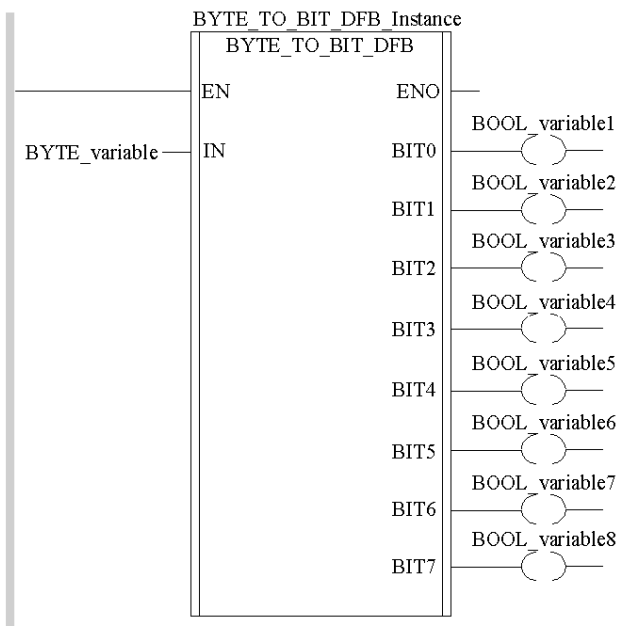
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL BYTE_TO_BIT_DFB_Instance (IN:=BYTE_variable,
    BIT0=>BOOL_variable1, BIT1=>BOOL_variable2,
    BIT2=>BOOL_variable3, BIT3=>BOOL_variable4,
    BIT4=>BOOL_variable5, BIT5=>BOOL_variable6,
    BIT6=>BOOL_variable7, BIT7=>BOOL_variable8)
```

Representation in ST

Representation:

```
BYTE_TO_BIT_DFB_Instance (IN:=BYTE_variable,
    BIT0=>BOOL_variable1, BIT1=>BOOL_variable2,
    BIT2=>BOOL_variable3, BIT3=>BOOL_variable4,
    BIT4=>BOOL_variable5, BIT5=>BOOL_variable6,
    BIT6=>BOOL_variable7, BIT7=>BOOL_variable8) ;
```

Parameter description

Description of the input parameters:

Parameter	Data type	Meaning
IN	BYTE	Input

Description of the output parameters:

Parameter	Data type	Meaning
BIT0	BOOL	Output bit 0
BIT1	BOOL	Output bit 1
:	:	:
BIT7	BOOL	Output bit 7

Chapter 8

CREADREG: Continuous register reading

Introduction

This chapter describes the CREADREG block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Description	120
Mode of Functioning	123
Parameter description	124
Modbus Plus Error Codes	125

Description

Function description

This derived function block reads the register area continuously. It reads data from addressed nodes via Modbus Plus.

EN and ENO can be configured as additional parameters.

NOTE: It is necessary to be familiar with the routing procedures of your network when programming a CREADREG function. Modbus Plus routing path structures are described in detail in the Modbus Plus Network Planning and Installation Guide ([see page 9](#)).

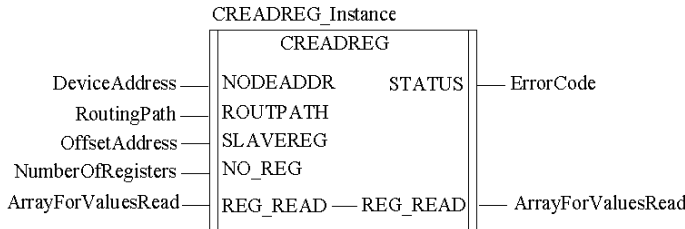
NOTE: This function block only supports the local Modbus Plus interface (no NOM).
If using a NOM work with the block CREAD_REG from the communication block library.

NOTE: This function block does not support TCP/IP- or SY/MAX-Ethernet.
If TCP/IP- or SY/MAX-Ethernet is needed, use the block CREAD_REG of the communication block library.

NOTE: Several copies of this function block can be used in the program. However, multiple instancing of these copies is not possible.

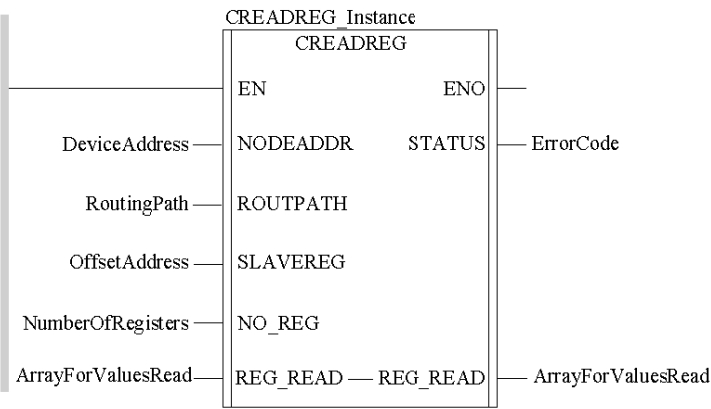
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL CREADREG_Instance (NODEADDR:=DeviceAddress,  
    ROUTPATH:=RoutingPath, SLAVEREG:=OffsetAddress,  
    NO_REG:=NumberOfRegisters,  
    REG_READ:=ArrayForValuesRead,  
    STATUS=>ErrorCode)
```

Representation in ST

Representation:

```
CREADREG_Instance (NODEADDR:=DeviceAddress,  
    ROUTPATH:=RoutingPath, SLAVEREG:=OffsetAddress,  
    NO_REG:=NumberOfRegisters,  
    REG_READ:=ArrayForValuesRead,  
    STATUS=>ErrorCode;
```

Parameter description

Description of the input parameters:

Parameters	Data type	Meaning
NODEADDR	INT	Device address within the target segment
ROUTPATH	DINT	Routing path to target segment
SLAVEREG	DINT	Offset address of the first 4x register in the slave to be read from
NO_REG	INT	Number of registers to be read from slave

Description of input / output parameters:

Parameters	Data type	Meaning
REG_READ	ANY_ARRAY_WORD	Writing data (For the file to be read a data structure must be declared as a located variable.)

Description of the output parameters:

Parameters	Data type	Meaning
STATUS	WORD	Error Code

Mode of Functioning

Function mode of CREADREG blocks

Although a large number of CREADREG function blocks can be programmed, only four read operations may be active at the same time. It makes no difference whether these operations are performed using this function block or others (e.g. MBP_MSTR, READREG). All function blocks use one data transaction path and require multiple cycles to complete a task.

The complete routing information must be separated into two parts:

- in the NODEADDR of the destination node (regardless of whether it is located in the local segment or in another segment) and
- the routing path, in case there is a link via network bridges.

The resulting destination address consists of these two information components.

The routing path is a DINT data type, which is interpreted as a sequence of two-digit information units. It is not necessary to use "00" extensions (e.g. both routing paths 4711 and 47110000 are valid, for NODEADDR 34 the result is destination address 47.11.34.00.00).

NOTE: This function block puts a heavy load on the network. The network load must therefore be carefully monitored. If the network load is too high, the program logic should be reorganized to work with the READREG function block, which is a variant of this function block that does not operate in continuous mode, but is command driven.

Parameter description

NODEADDR

Identifies the node address within the target segment.

The parameter can be entered as an address, located variable, unlocated variable or literal.

ROUTPATH

Identifies the routing path to the target segment. The two-digit information units run from 01 64 (see *Mode of Functioning*, [page 123](#)). If the slave resides in the local network segment, ROUTPATH must be set to "0" or must be left unconnected.

The parameter can be entered as an address, located variable, unlocated variable or literal.

SLAVEREG

Start of the area in the addressed slave from which the source data are read. The source area always resides within the 4x register area. SLAVEREG expects the source reference as offset within the 4x area. The leading "4" must be omitted (e.g. 59 (contents of the variables or value of the literal) = 40059).

The parameter can be entered as an address, located variable, unlocated variable or literal.

NO_REG

Number of registers to be read from slave processor (1 ... 100).

The parameter can be entered as an address, located variable, unlocated variable or literal.

STATUS

Error code, see *Modbus Plus Error Codes*, [page 125](#)

The parameter can be specified as an address, located variable or unlocated variable.

REG_READ

An ANY_ARRAY_WORD that is the same size as the requested transmission must be agreed upon ($\geq$ NO_REG) for this parameter. The name of this array is defined as a parameter. If the array is defined too small, then only the amount of data is transmitted that is present in the array.

The parameter must be defined as a located variable.

Modbus Plus Error Codes

Form of the function error code

The form of the function error code for Modbus Plus is **Mmss**, which includes:

- **M** is the high code
- **m** is the low code
- **ss** is a subcode

Hexadecimal error code

Hexadecimal error code for Modbus Plus:

Hex. Error Code	Meaning
1001	Abort by user
2001	An operation type that is not supported was specified in the control block
2002	One or more control block parameters were modified while the MSTR element was active (this only applies to operations which require several cycles for completion). Control block parameters may only be modified in inactive MSTR components.
2003	Illegal value in the length field of the control block
2004	Illegal value in the offset field of the control block
2005	Illegal value in the length and offset fields of the control block
2006	Unauthorized data field on slave
2007	Unauthorized network field on slave
2008	Unauthorized network routing path on slave
2009	Routing path equivalent to own address
200A	Attempting to retrieve more global data words than available
30ss	Unusual response by Modbus slave (see page 126)
4001	Inconsistent response by Modbus slave
5001	Inconsistent response by network
6mss	Routing path error (see page 126) Subfield m shows where the error occurred (a 0 value means local node, 2 means 2nd device in route, etc) .

ss hexadecimal value in 30ss error code

ss hexadecimal value in 30ss error code:

ss hex. Value	Meaning
01	Slave does not support requested operation
02	Non-existent slave registers were requested
03	An unauthorized data value was requested
05	Slave has accepted a lengthy program command
06	Function cannot currently be carried out: lengthy command running
07	Slave has rejected lengthy program command

ss hexadecimal value in 6mss error code

NOTE: Subfield m in error code 6mss is an `Index` in the routing information that shows where an error has been detected (a 0 value indicates the local node, 2 means the second device in the route, etc.).

The ss subfield in error code 6mss is as follows:

ss hexadecimal value	Meaning
01	No response receipt
02	Access to program denied
03	Node out of service and unable to communicate
04	Unusual response received
05	Router-node data path busy
06	Slave out of order
07	Wrong destination address
08	Unauthorized node type in routing path
10	Slave has rejected the command
20	Slave has lost an activated transaction
40	Unexpected master output path received
80	Unexpected response received
F001	Wrong destination node specified for <code>MSTR</code> operation

Chapter 9

CWRITREG: Continuous register writing

Introduction

This chapter describes the CWRITREG block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Description	128
Mode of Functioning	131
Parameter description	132

Description

Function description

This derived function block writes continuously to the register area. It transfers data from the PLC via Modbus Plus to a specified slave destination processor.

EN and ENO can be configured as additional parameters.

NOTE: When programming a CWRITREG function, you must be familiar with the routing procedures used by your network. Modbus Plus routing path structures are described in detail in the Modbus Plus Network Planning and Installation Guide ([see page 9](#)).

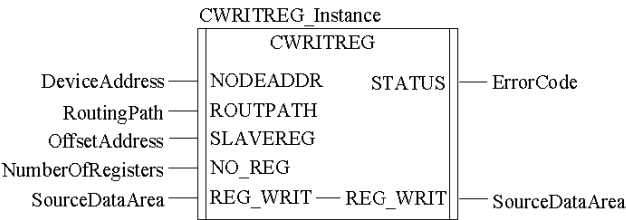
NOTE: This function block only supports the local Modbus Plus interface (no NOM).
If using a NOM work with the CWRITE_REG block from the communication block library.

NOTE: This function block does not support TCP/IP- or SY/MAX-Ethernet.
If TCP/IP- or SY/MAX-Ethernet is needed, use the CWRITE_REG block from the communication block library.

NOTE: Several copies of this function block can be used in the program. However, multiple instancing of these copies is not possible.

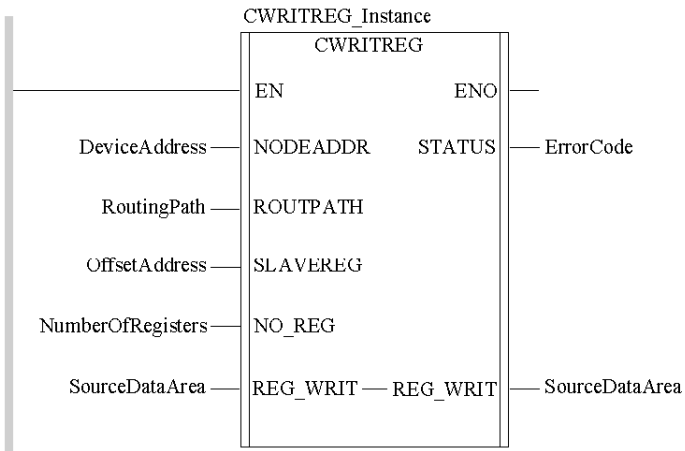
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```

CAL CWRITREG_Instance (NODEADDR:=DeviceAddress,
  ROUTPATH:=RoutingPath, SLAVEREG:=OffsetAddress,
  NO_REG:=NumberOfRegisters,
  REG_WRIT:=SourceDataArea,
  STATUS=>ErrorCode)
  
```

Representation in ST

Representation:

```

CWRITREG_Instance (NODEADDR:=DeviceAddress,
  ROUTPATH:=RoutingPath, SLAVEREG:=OffsetAddress,
  NO_REG:=NumberOfRegisters,
  REG_WRIT:=SourceDataArea,
  STATUS=>ErrorCode) ;
  
```

Parameter description

Description of the input parameters:

Parameters	Data type	Meaning
NODEADDR	INT	Device address within the target segment
ROUTPATH	DINT	Routing path to target segment
SLAVEREG	DINT	Offset address of the first 4x register in the slave to be written to
NO_REG	INT	Number of registers to be written from slave

Description of input / output parameters:

Parameters	Data type	Meaning
REG_WRIT	ANY_ARRAY_WORD	Source data field (A data structure must be declared as a located variable for the source file.)

Description of the output parameters:

Parameters	Data type	Meaning
STATUS	WORD	Error Code

Mode of Functioning

Function mode of CWRITREG blocks

Although an unlimited number of CWRITREG function blocks can be programmed, only four write operations may be active at the same time. It makes no difference whether these operations are performed using this function block or others (e.g. MBP_MSTR, WRITEREG). All function blocks use one data transaction path and require multiple cycles to complete a task.

If several CWRITREG function blocks are used within an application, they must at least differ in the values of their NO_REG or REG_WRIT parameters.

The complete routing information must be separated into two parts:

- in the NODEADDR of the destination node (regardless of whether it is located in the local segment or in another segment) and
- the routing path, in case there is a link via network bridges.

The resulting destination address consists of these two information components.

The routing path is a DINT data type, which is interpreted as a sequence of two-digit information units. It is not necessary to use "00" extensions (e.g. both routing paths 4711 and 47110000 are valid, for NODEADDR 34 the result is destination address 47.11.34.00.00).

NOTE: This function block puts a heavy load on the network. The network load must therefore be carefully monitored. If the network load is too high, the program logic should be reorganized to work with the WRITEREG function block, which is a variant of this function block that does not operate in continuous mode, but is command driven.

Parameter description

NODEADDR

Identifies the node address within the target segment.

The parameter can be specified as an address, located variable, unlocated variable or literal.

ROUTPATH

Identifies the routing path to the target segment. The two-digit information units run from 01 64 (see *Mode of Functioning*, [page 131](#)). If the slave resides in the local network segment, `ROUTPATH` must be set to "0" or must be left unconnected.

The parameter can be specified as an address, located variable, unlocated variable or literal.

SLAVEREG

Start of the destination area in the addressed slave to which the source data are written. The destination area always resides within the 4x register area. `SLAVEREG` expects the destination address as an offset within the 4x area. The leading "4" must be omitted (e.g. 59 (contents of the variables or value of the literal) = 40059).

The parameter can be entered as an address, located variable, unlocated variable or literal.

NO_REG

Number of registers to be written to slave processor (1 ... 100).

The parameter can be specified as an address, located variable, unlocated variable or literal.

REG_WRIT

An `ANY_ARRAY_WORD` that is the same size as the planned transmission must be agreed upon ($\geq$ `NO_REG`) for this parameter. The name of this array is defined as a parameter. If the array is defined too small, then only the amount of data is transmitted that is present in the array.

The parameter must be defined as a located variable.

STATUS

If `MSTR` error code is returned, see *Modbus Plus Error Codes*, [page 125](#)

The parameter can be specified as an address, located variable or unlocated variable.

Chapter 10

DINT_AS_WORD_DFB: Type conversion

Description

Function description

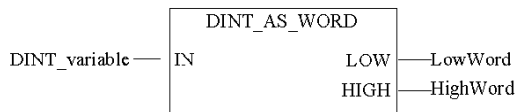
This derived function block converts one input word from the `DINT` data type to 2 output values of the `WORD` data type.

The individual words of the `DINT` input are assigned to the outputs according to the output names.

`EN` and `ENO` can be configured as additional parameters.

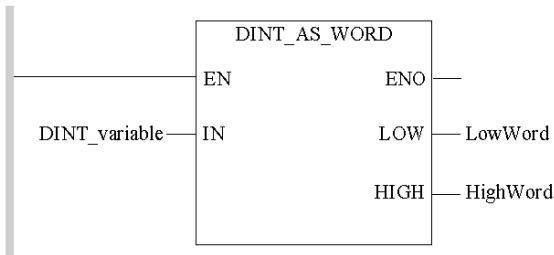
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL DINT_AS_WORD_DFB_Instance (IN:=DINT_variable,  
                                LOW=>LowWord, HIGH=>HighWord)
```

Representation in ST

Representation:

```
DINT_AS_WORD_DFB_Instance (IN:=DINT_variable,  
    LOW=>LowWord, HIGH=>HighWord) ;
```

Parameter description

Description of the input parameters:

Parameters	Data type	Meaning
IN	DINT	Input

Description of the output parameters:

Parameters	Data type	Meaning
LOW	WORD	least significant word
HIGH	WORD	most significant word

Chapter 11

DIOSTAT: Module function status (DIO)

Description

Function description

This function provides the function status for I/O modules of an I/O station (DIO).

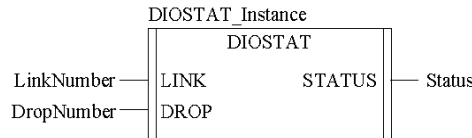
Each module (slot) is displayed as an output "status" bit. The bit on the far left side in "status" corresponds to the slot on the far left side of the I/O station.

NOTE: If a module of the I/O station is configured and works correctly, the corresponding bit is set to "1".

EN and ENO can be configured as additional parameters.

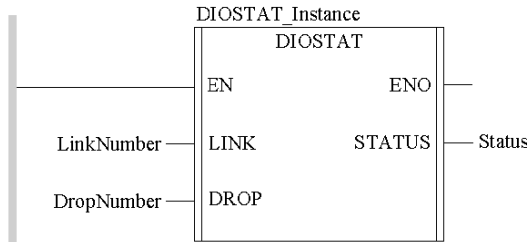
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL DIOSTAT_Instance (LINK:=LinkNumber, DROP:=DropNumber,  
    STATUS=>Status)
```

Representation in ST

Representation:

```
DIOSTAT_Instance (LINK:=LinkNumber, DROP:=DropNumber,  
    STATUS=>Status) ;
```

Parameter description

Description of the input parameters:

Parameter	Data type	Meaning
LINK	UINT	Link No. (0...2)
DROP	UINT	I/O station no: (1...64)

Description of the output parameters:

Parameter	Data type	Meaning
STATUS	WORD	Status bit pattern (see page 135) of an I/O station

Chapter 12

GET_TOD: Reading the hardware clock (Time Of Day)

Description

Function description

This function block searches (together with the other function blocks in the HSBY group) the configuration of the respective PLC for the necessary components. These components always refer to the hardware actually connected.

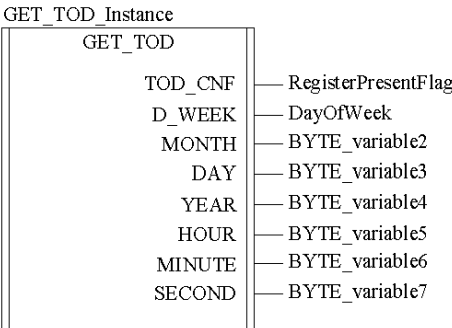
Therefore the correct functioning of this function block on the simulators cannot be guaranteed.

The `GET_TOD` function block reads the hardware clock, if relevant registers are provided with this configuration. If these registers are not present, the `TOD_CNF` output is set to "0".

`EN` and `ENO` can be configured as additional parameters.

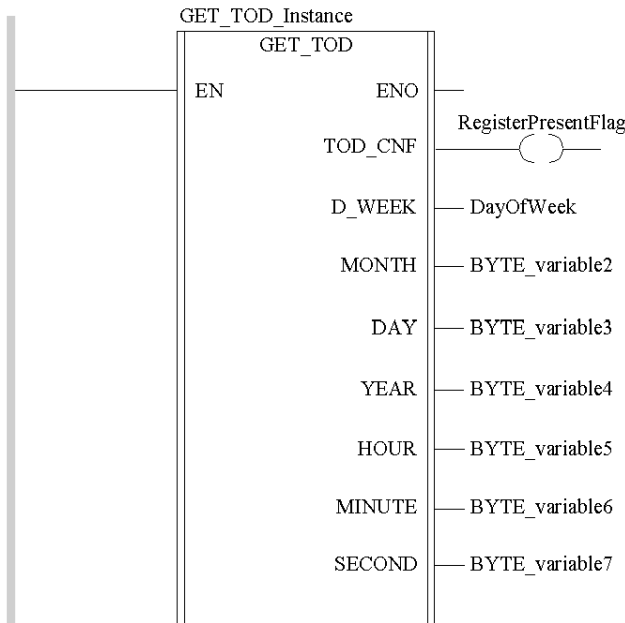
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL GET_TOD_Instance (TOD_CNF=>RegisterPresentFlag,
    D_WEEK=>DayOfWeek, MONTH=>Byte_variable2,
    DAY=>Byte_variable3, YEAR=>Byte_variable4,
    HOUR=>Byte_variable5, MINUTE=>Byte_variable6,
    SECOND=>Byte_variable7)
```

Representation in ST

Representation:

```
GET_TOD_Instance (TOD_CNF=>RegisterPresentFlag,
    D_WEEK=>DayOfWeek, MONTH=>Byte_variable2,
    DAY=>Byte_variable3, YEAR=>Byte_variable4,
    HOUR=>Byte_variable5, MINUTE=>Byte_variable6,
    SECOND=>Byte_variable7) ;
```

Parameter description

Description of the output parameters:

Parameters	Data type	Meaning
TOD_CNF	BOOL	"1" = 4x-register for hardware system clock was found and the clock is operational. "0" = time is set at the moment. In this case the other outputs keep their values.
D_WEEK	BYTE	Weekday, 1 = Sunday .. 7 = Saturday
MONTH	BYTE	Month 1..12
DAY	BYTE	Day 1..31
YEAR	BYTE	Year 0..99
HOUR	BYTE	Hour 0..23
MINUTE	BYTE	Minute 0..59
SECOND	BYTE	Second 0..59

Chapter 13

LIMIT_IND_DFB: Limit with indicator

Description

Function description

This derived function block transfers the unchanged input value (*Input*) to the *Output*, if the input value is not less than the minimum value (*LimitMinimum*) and does not exceed the maximum value (*LimitMaximum*). If the input value (*Input*) is less than the minimum value (*LimitMinimum*), the minimum value will be transferred to the output. If the input value (*Input*) exceeds the maximum value (*LimitMaximum*), the maximum value will be transferred to the output.

Additionally, a indication is given if the minimum or maximum value is violated. If the value at the (*Input*) input is less than the value at the (*LimitMinimum*) input, the (*MinimumViolation*) output becomes "1". If the value at the (*Input*) input is more than the value at the (*LimitMaximum*) input, the (*MaximumViolation*) output becomes "1".

The data types of the (*LimitMinimum*, *Input*, *LimitMaximum*) input values and the (*Output*) output value must be identical.

EN and ENO can be configured as additional parameters.

Formula

Block formula:

$OUT = IN, \text{ if } (IN \leq MX) \ \& \ IN \geq MN$

$OUT = MN, \text{ if } (IN < MN)$

$OUT = MX, \text{ if } (IN > MX)$

$MN_IND = 0, \text{ if } IN \geq MN$

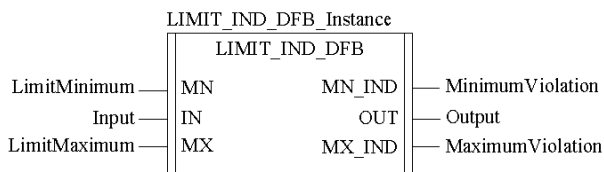
$MN_IND = 1, \text{ if } IN < MN$

$MX_IND = 0, \text{ if } IN \leq MX$

$MX_IND = 1, \text{ if } IN > MX$

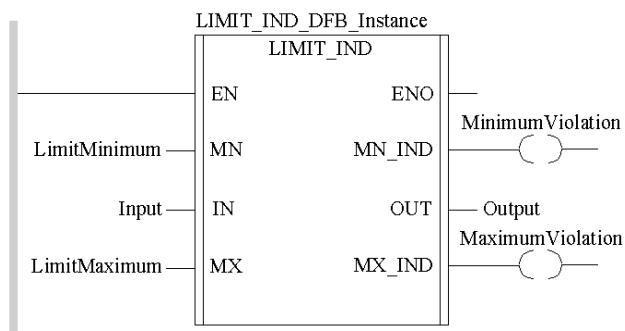
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL LIMIT_IND_DFB (MN:=LimitMinimum, IN:=INPUT,
  MX:=LimitMaximum, MN_IND=>MinimumViolation,
  OUT=>Output, MX_IND=>MaximumViolation)
```

Representation in ST

Representation:

```
LIMIT_IND_DFB (MN:=LimitMinimum, IN:=INPUT,
  MX:=LimitMaximum, MN_IND=>MinimumViolation,
  OUT=>Output, MX_IND=>MaximumViolation) ;
```

Parameter description

Description of the input parameters:

Parameter	Data type	Meaning
LimitMinimum	BOOL, BYTE, WORD, DWORD, INT, DINT, UINT, UDINT, REAL, TIME	Limit of minimum value
Input	BOOL, BYTE, WORD, DWORD, INT, DINT, UINT, UDINT, REAL, TIME	Input
LimitMaximum	BOOL, BYTE, WORD, DWORD, INT, DINT, UINT, UDINT, REAL, TIME	Limit of maximum value

Description of the output parameters:

Parameter	Data type	Meaning
MinimumViolation	BOOL	Display of minimum value violation
Output	BOOL, BYTE, WORD, DWORD, INT, DINT, UINT, UDINT, REAL, TIME	Output
MaximumViolation	BOOL	Display of maximum value violation

Chapter 14

LOOKUP_TABLE1_DFB: Traverse progression with 1st degree interpolation

Introduction

This chapter describes the LOOKUP_TABLE1_DFB block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Description	146
Detailed description	148

Description

Function description

This function block linearizes characteristic curves by means of interpolation. The function block works with variable support point width.

The number of X_iY_i inputs can be increased to 30 by modifying the size of the block frame vertically. This corresponds to a maximum of 15 support point pairs.

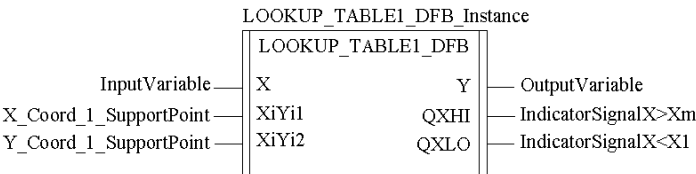
The number of inputs must be even.

The X values must be in ascending order.

EN and ENO can be configured as additional parameters.

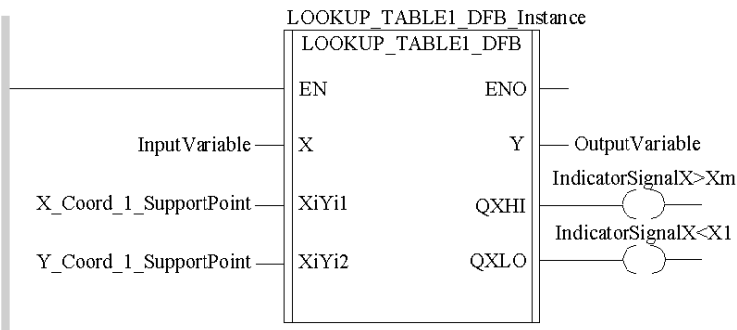
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL LOOKUP_TABLE1_DFB_Instance (X:=InputVariable,
  XiYi1:=X_Coord_1_SupportPoint,
  XiYi2:=Y_Coord_1_SupportPoint, Y=>OutputVariable,
  QXHI=>IndicatorSignalX>Xm, QXLO=>IndicatorSignalX<Xl)
```

Representation in ST

Representation:

```
LOOKUP_TABLE1_DFB_Instance (X:=InputVariable,
  XiYi1:=X_Coord_1_SupportPoint,
  XiYi2:=Y_Coord_1_SupportPoint, Y=>OutputVariable,
  QXHI=>IndicatorSignalX>Xm, QXLO=>IndicatorSignalX<Xl) ;
```

Parameter description

Description of the input parameters:

Parameter	Data type	Meaning
XiYi1	REAL	X coordinate 1. Support point
XiYi2	REAL	Y coordinate 1. Support point
XiYin	REAL	X coordinate m/2. Support point
XiYim	REAL	Y coordinate m/2. Support point
X	REAL	Input variable

Description of the output parameters:

Parameter	Data type	Meaning
Y	REAL	Output variable
QXHI	BOOL	Indicator: $X > X_m$
QXLO	BOOL	Indicate $X < X_l$

Detailed description

Parameter description

Each two sequential inputs ($x_i y_i$) represent a support point pair. The first input $x_i y_i$ corresponds to x_1 , the next one to y_1 , the one after that to x_2 , etc.

For all types of input value in x found between these support points, the corresponding y output value is interpolated, while the traverse progression between the support points is viewed linearly.

For $x < x_1$ is $y = y_1$

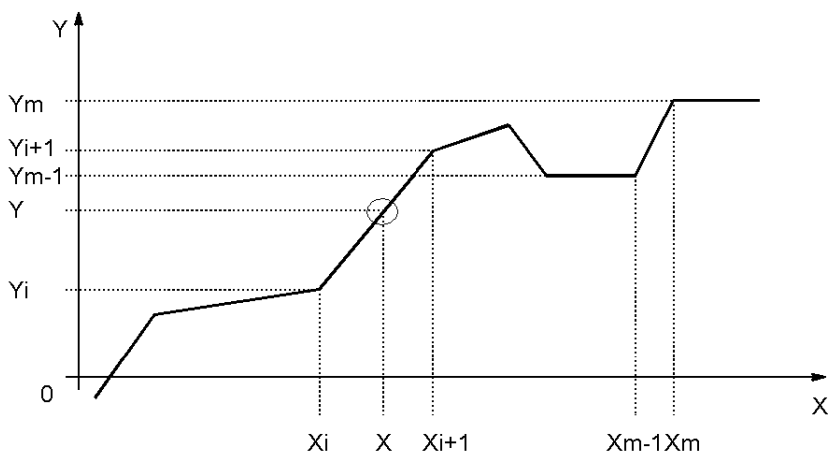
For $x > x_m$ is $y = y_m$

If the value at input x is higher than the value of the last support point x_m , the output Q_{XHI} becomes "1".

If the value at input x is less than the value of the first support point x_1 , the output Q_{XLO} becomes "1".

Principle of interpolation

Traverse progression with 1st degree interpolation)



Interpolation

The following algorithm applies to a point Y :

$$Y = Y_i + \frac{Y_{i+1} - Y_i}{X_{i+1} - X_i} \times (X - X_i)$$

for $X_i \leq X \leq X_{i+1}$ and $i = 1 \dots (m-1)$

Assuming: $X_1 \leq X_2 \leq \dots \leq X_i \leq X_{i+1} \leq \dots \leq X_{m-1} \leq X_m$

The X values must be in ascending order.

Two consecutive X values can be identical. This could cause a discrete curve progression.

In this instance, the special case applies:

$$Y = 0.5 \times (Y_i + Y_{i+1})$$

for

$X_i = X = X_{i+1}$ and $i = 1 \dots (m-1)$

Chapter 15

PLCSTAT: PLC function status

Introduction

This chapter describes the `PLCSTAT` block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Description	152
Derived Data Types	154
PLC status (<code>PLC_STAT</code>)	156
RIO status (<code>RIO_STAT</code>) for Quantum	158
DIO status (<code>DIO_STAT</code>)	160

Description

Function description

This derived function block reads the Quantum PLC internal states and error bits and copies this data to the data structures allocated to the respective outputs.

EN and ENO can be configured as additional parameters.

Only data with the input bit (PLC_READ, RIO_READ, DIO_READ) set to "1" will be read.

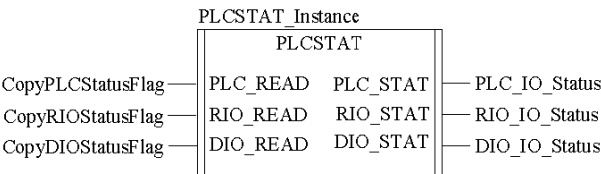
Evaluation

The evaluation of PLC_STAT (PLC status), RIO_STAT (I/O status) and DIO_STAT (I/O communications status) is possible.

NOTE: The name of the output DIO_STAT is confusing. This output only relates to the remote I/O Drop Status Information (S908) and not to the Distributed I/O status. To read the distributed I/O status use the function block DIOSTAT ([see page 135](#)).

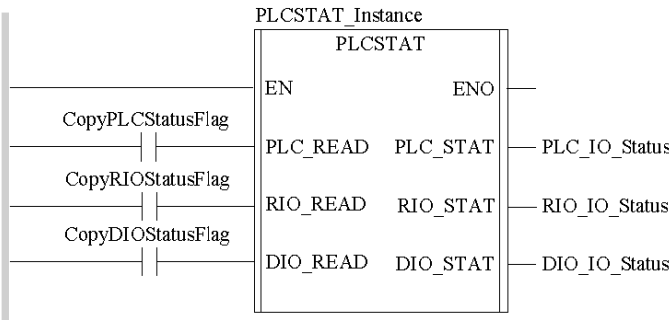
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL PLCSTAT_Instance (PLC_READ:=CopyPLCStatusFlag,
  RIO_READ:=CopyRIOStatusFlag,
  DIO_READ:=CopyDIOStatusFlag,
  PLC_STAT=>PLC_IO_Status, RIO_STAT=>RIO_IO_Status,
  DIO_STAT=>DIO_IO_Status)
```

Representation in ST

Representation:

```
PLCSTAT_Instance (PLC_READ:=CopyPLCStatusFlag,
  RIO_READ:=CopyRIOStatusFlag,
  DIO_READ:=CopyDIOStatusFlag,
  PLC_STAT=>PLC_IO_Status, RIO_STAT=>RIO_IO_Status,
  DIO_STAT=>DIO_IO_Status) ;
```

PLCSTAT parameter description

Description of the input parameters:

Parameters	Data type	Meaning
PLC_READ	BOOL	1 = copies the PLC status from the status table to the output PLC_STAT.
RIO_READ	BOOL	1 = copies the RIO status from the status table to the output RIO_STAT.
DIO_READ	BOOL	1 = copies the DIO status from the status table to the output DIO_STAT.

Description of the output parameters:

Parameters	Data type	Meaning
PLC_STAT	PLCSTATE,	Contains the PLC status.
RIO_STAT	RIOSTATE,	Contains the RIO status (I/O status) for Quantum
DIO_STAT	DIOSTATE,	Contains the DIO status (I/O communication status) Note: The name of this output is confusing. This output only relates to the remote I/O Drop Status Information (S908) and not to the Distributed I/O status. To read the distributed I/O status use the function block DIOSTAT (see page 135).

Derived Data Types

Element description **PLCSTATE**

Description of the **PLCSTATE** element:

Element	Data type	Meaning
word1	WORD	CPU status
word2	WORD	Hot Standby Status
word3	WORD	PLC status
word4	WORD	RIO Status
word5	WORD	Reserve
word6	WORD	Reserve
word7	WORD	Reserve
word8	WORD	Reserve
word9	WORD	Reserve
word10	WORD	Reserve
word11	WORD	Reserve

Element description **RIOSTATE**

Description of the **RIOSTATE** element

Element	Data type	Meaning
word1	WORD	I/O station 1, module rack 1
word2	WORD	I/O station 1, module rack 2
...	...	...
word5	WORD	I/O station 1, module rack 5
word6	WORD	I/O station 2, module rack 1
word7	WORD	I/O station 2, module rack 2
...	...	...
word160	WORD	I/O station 32, module rack 5

Element description DIOSTATE

Description of the DIOSTATE element

Element	Data type	Meaning
word1	WORD	Switch on error numbers:
word2	WORD	Cable A error
word3	WORD	Cable A error
word4	WORD	Cable A error
word5	WORD	Cable B error
word6	WORD	Cable B error
word7	WORD	Cable B error
word8	WORD	Global communication status
word9	WORD	Global cumulative error counter for cable A
word10	WORD	Global cumulative error counter for cable B
word11	WORD	I/O station 1 health status and repetition counter (first word)
word12	WORD	I/O station 1 health status and repetition counter (second word)
word13	WORD	I/O station 1 health status and repetition counter (third word)
word14	WORD	I/O station 2 health status and repetition counter (first word)
...	...	...
word104	WORD	I/O station 32 health status and repetition counter (first word)
word105	WORD	I/O station 32 health status and repetition counter (second word)
word106	WORD	I/O station 32 health status and repetition counter (third word)

PLC status (PLC_STAT)

General information

NOTE: Information corresponds to status table words 1 to 11 in the dialog **PLC status**.
The conditions are true when the bits are set to 1.

PLC status (PLCSTATE: word1)

Bit allocation:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
10	Run light OFF
11	Memory protect OFF
12	Battery failed

Hot Standby status (PLCSTATE: word2)

Bit allocation:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1	CHS 110/S911/R911 present and OK
11	0 = CHS shift switch set to A 1 = CHS shift switch set to B
12	0 = PLCs have equal logic 1 = PLCs have unequal logic
13, 14	Remote system condition Dec binary 1 0 1 = Offline 2 1 0 = Primary 3 1 1 = Standby
15, 16	Local system condition Dec binary 1 0 1 = Offline 2 1 0 = Primary 3 1 1 = Standby

PLC status (PLCSTATE: word3)

Bit allocation:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1	First cycle

RIO status (PLCSTATE: word4)

Bit allocation:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1	IOP defect
2	IOP timeout
3	IOP Loopback
4	IOP memory disturbance
13-16	00 IO has not responded 01 no response 02 Loopback defect

RIO status (RIO_STAT) for Quantum

General information

NOTE: The information corresponds to status table words 12 to 171 in the PLC status dialog.

The words show the I/O module function status.

Five words are reserved for each of the maximum 32 I/O stations. Each word corresponds to one of maximal 2 possible module racks in each I/O station.

Function display for Quantum hardware

Each of the module racks for Quantum hardware can contain up to 15 I/O modules (except for the first rack which contains a maximum 14 I/O modules). Bit 1... 16 in each word show the corresponding I/O module function display in the racks.

I/O module function status

Bit allocation:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1	Slot 1
2	Slot 2
...	...
16	Slot 16

Conditions for a correct function display

Four conditions must be fulfilled if an I/O module can give a correct function display:

- The data traffic of the slot has to be monitored.
- The slot must be valid for the inserted module.
- Valid communication must be established between the module and the RIO interface at RIO stations.
- Valid communication must be established between the I/O processor in the PLC and the RIO interface at the RIO station.

Status words for the MMI user controllers

The status of the 32 element button controllers and PanelMate units in a RIO network can also be monitored with an I/O function status word. The button controllers are located on slot 4 in a I/O rack and can be monitored at bit 4 of the corresponding status word. A PanelMate on RIO is located on slot 1 in module rack 1 of the I/O station and can be monitored at bit 1 of the first status word for the I/O station.

NOTE: The ASCII keyboard communication status can be monitored with the error numbers in the ASCII read/write instructions.

DIO status (DIO_STAT)

General information

NOTE: The information corresponds to status table words 172 to 277 in the PLC status dialog.

The words contain the I/O communication status (DIO status) Words 1 to 10 are global status words. Of the remaining 96 words, three words are allocated to each of the up to 32 I/O stations.

`word1` saves the switch on error numbers. This word is always 0 when the system is running. If an error occurs, the PLC does not start but generates a PLC stop status (`word5` from `PLC_STAT`).

The conditions are true when the bits are set to 1.

Switch on error numbers (DIOSTATE `word1`)

The conditions are true when the bits are set to 1.

Switch on error numbers:

Code	Error	Meaning (location of error)
01	BADTCLEN	Traffic cop length
02	BADLNKNUM	RIO link number
03	BADNUMDPS	I/O station number in traffic cop
04	BADTCSUM	Traffic cop checksum
10	BADDDLEN	I/O station descriptor length
11	BADDRPNUM	I/O station number
12	BADHUPTIM	I/O station stop time
13	BADASCNUM	ASCII port number
14	BADNUMODS	Module number in I/O station
15	PRECONDRP	I/O station is already configured
16	PRECONPRT	Port is already configured
17	TOOMNYOUT	More than 1024 output locations
18	TOOMNYINS	More than 1024 input points
20	BADSLTNUM	Module slot address
21	BADRCKNUM	Rack address
22	BADOUTBC	Number of output bytes
23	BADINBC	Number of input bytes
25	BADRF1MAP	First reference number
26	BADRF2MAP	Second reference number
27	NOBYTES	No input or output bytes
28	BADDISMAP	I/O marker bit not at 16 bit limit

Code	Error	Meaning (location of error)
30	BADODDOUT	Unmated, odd output module
31	BADODDIN	Unmated, odd input module
32	BADODDREF	Unmated odd module reference
33	BAD3X1XRF	1x-reference after 3x-register
34	BADDMYMOD	Dummy module reference already in use
35	NOT3XDMY	3x-module is not a dummy module
36	NOT4XDMY	4x-module is not a dummy module
40	DMYREAL1X	Dummy module, then real 1x-module
41	REALDMY1X	Real, then 1x-dummy module
42	DMYREAL3X	Dummy module, then real 3x-module
43	REALDMY3X	Real, then 3x-dummy module

Status of cable A (DIOSTATE: word2, word3, word4)

Bit allocation for word2:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1 - 8	Counts frame fields
9 - 16	Counts DMA receiver overflows

Bit allocation for word3:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1 - 8	Counts receiver errors
9 - 16	Counts I/O station receiver failures

Bit allocation for word4:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1	1 = frame too short
2	1 = no frame end
13	1 = CRC error
14	1 = alignment error
15	1 = overflow error

Status of cable B (DIOSTATE: word5, word6, word7)

Bit allocation for word5:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1 - 8	Counts frame fields
9 - 16	Counts DMA receiver overflows

Bit allocation for word6:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1 - 8	Counts receiver errors
9 - 16	Counts I/O station receiver failures

Bit allocation for word7:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1	1 = frame too short
2	1 = no frame end
13	1 = CRC error
14	1 = alignment error
15	1 = overflow error

Global communication status (DIOSTATE: word8)

The conditions are true when the bits are set to 1.

Bit allocation for word8:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1	Comm. function display
2	Cable A status
3	Cable B status
5 - 8	Communication counter lost
9 - 16	Cumulative repetition counter

Global cumulative error counter for cable A (DIOSTATE: word9)

The conditions are true when the bits are set to 1.

Bit allocation for word9:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1 - 8	Counts recognized errors
9 - 16	Counts zero responses

Global cumulative error counter for cable B(DIOSTATE: word10)

The conditions are true when the bits are set to 1.

Bit allocation for word10:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1 - 8	Counts recognized errors
9 - 16	Counts zero responses

RIO status (DIOSTATE: word11 to word106)

Words 11 to 106 are used to describe the RIO station status, three status words are planned for each I/O station.

The **first** word in each group of three shows the communication status for the corresponding I/O station:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1	Communication health
2	Cable A status
3	Cable B status
5 - 8	Counter for lost communications
9 - 16	Cumulative repetition counter

The **second** word in each group of three is the cumulative I/O station error counter at cable A for the corresponding I/O station:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1 - 8	Minimum one error in words 2 to 4
9 - 16	Counts zero responses

The **third** word in each group of three is the cumulative I/O station error counter at cable B for the corresponding I/O station:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1 - 8	Minimum one error in words 5 to 7
9 - 16	Counts zero responses

NOTE: For PLCs where the I/O station 1 is reserved for the local I/O, words word11 to word13 are allocated as follows:

word11 shows the global I/O station status:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Bit	Allocation
1	All modules OK
9 - 16	Counts, how often a module is regarded as not OK, counter overflow is at 255

word12 is used as a 16 bit I/O bus error counter.

word13 is used as a 16 bit I/O repetition counter.

Chapter 16

READREG: Read register

Introduction

This chapter describes the READREG block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Description	168
Mode of Functioning	171
Parameter description	172

Description

Function description

With a rising edge at the `REQ` input, this function block reads a register area from an addressed slave via Modbus Plus.

`EN` and `ENO` can be configured as additional parameters.

NOTE: When programming a `READREG` function, you must be familiar with the routing procedures used by your network. Modbus Plus routing path structures are described in detail in the Modbus Plus Network Planning and Installation Guide ([see page 9](#)).

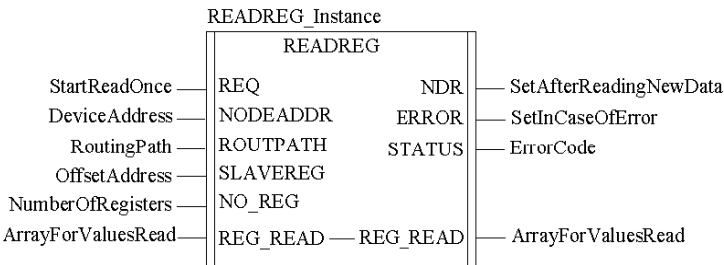
NOTE: This function block only supports the local Modbus Plus interface (no NOM).
If using a NOM, work with the `CREAD_REG` block from the communication block library.

NOTE: This function block does not support TCP/IP- or SY/MAX-Ethernet.
If TCP/IP- or SY/MAX-Ethernet is needed, use the `CREAD_REG` block from the communication block library.

NOTE: Several copies of this function block can be used in the program. However, multiple instancing of these copies is not possible.

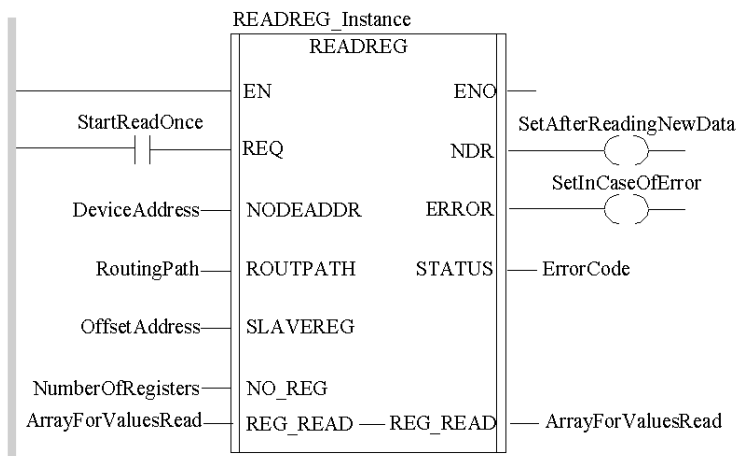
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL READREG_Instance (REQ:=StartReadOnce,
  NODEADDR:=DeviceAddress, ROUTPATH:=RoutingPath,
  SLAVEREG:=OffsetAddress, NO_REG:=NumberOfRegisters,
  REG_READ:=ArrayForValuesRead,
  NDR=>SetAfterReadingNewData, ERROR=>SetInCaseOfError,
  STATUS=>ErrorCode)
```

Representation in ST

Representation:

```
READREG_Instance (REQ:=StartReadOnce,
  NODEADDR:=DeviceAddress, ROUTPATH:=RoutingPath,
  SLAVEREG:=OffsetAddress, NO_REG:=NumberOfRegisters,
  REG_READ:=ArrayForValuesRead,
  NDR=>SetAfterReadingNewData, ERROR=>SetInCaseOfError,
  STATUS=>ErrorCode;
```

Parameter description

Description of the input parameters:

Parameter	Data type	Meaning
REQ	BOOL	With a rising edge at the REQ input, this function block reads a register area from an addressed slave via Modbus Plus.
NODEADDR	INT	Device address within the target segment
ROUTPATH	DINT	Routing path to target segment
SLAVEREG	DINT	Offset address of the first 4x register in the slave to be read from
NO_REG	INT	Number of registers to be read from slave

Description of input / output parameters:

Parameters	Data type	Meaning
REG_READ	ANY_ARRAY_WORD	Writing data (For the file to be read a data structure must be declared as a located variable.)

Description of the output parameters:

Parameters	Data type	Meaning
NDR	BOOL	Set to 1 for one cycle after reading new data
ERROR	BOOL	Set to 1 for one scan in case of error
STATUS	WORD	Error Code

Mode of Functioning

Function mode of READREG_DFB blocks

Although a large number of READREG function blocks can be programmed, only four read operations may be active at the same time. It makes no difference whether these operations are performed using this function block or others (e.g. MBP_MSTR, CREAD_REG). All function blocks use one data transaction path and require multiple cycles to complete a task. The status signals NDR and ERROR report the function block state to the user program.

The complete routing information must be separated into two parts:

- in the NODEADDR of the destination node (regardless of whether it is located in the local segment or in another segment) and
- the routing path, in case there is a link via bridges.

The resulting destination address consists of these two information components.

The routing path is a DINT data type, which is interpreted as a sequence of two-digit information units. It is not necessary to use "00" extensions (e.g. both routing paths 4711 and 47110000 are valid, for NODEADDR 34 the result is destination address 47.11.34.00.00).

Parameter description

REQ

A rising edge triggers the read transaction.

The parameter can be specified as an address, located variable, unlocated variable or literal.

NODEADDR

Identifies the node address within the target segment.

The parameter can be specified as an address, located variable, unlocated variable or literal.

ROUTPATH

Identifies the routing path to the target segment. The two-digit information units run from 01 64 (see *Mode of Functioning*, [page 171](#)). If the slave resides in the local network segment, ROUTPATH must be set to "0" or must be left unconnected.

The parameter can be specified as an address, located variable, unlocated variable or literal.

SLAVEREG

Start of the area in the addressed slave from which the source data is read. The source area always resides within the 4x register area. SLAVEREG expects the source reference as offset within the 4x area. The leading "4" must be omitted (e.g. 59 (contents of the variables or value of the literal) = 40059).

The parameter can be specified as an address, located variable, unlocated variable or literal.

NO_REG

Number of registers to be read from slave processor (1 ... 100).

The parameter can be specified as an address, located variable, unlocated variable or literal.

NDR

Transition to ON state for one program cycle signifies receipt of new data ready to be processed.

The parameter can be specified as an address, located variable or unlocated variable.

ERROR

Transition to ON state for one program cycle signifies detection of a new error.

The parameter can be specified as an address, located variable or unlocated variable.

STATUS

Error code, see *Modbus Plus Error Codes*, [page 125](#)

The parameter can be specified as an address, located variable or unlocated variable.

REG_READ

An `ANY_ARRAY_WORD` that is the same size as the requested transmission must be agreed upon ($\geq \text{NO_REG}$) for this parameter. The name of this array is defined as a parameter. If the array is defined too small, then only the amount of data is transmitted that is present in the array.

The parameter must be defined as a located variable.

Chapter 17

RIOSTAT: Module function status (RIO)

Description

Function description

This function block provides the function status for I/O modules of an I/O station (local/remote I/O). Quantum I/O or 800 I/O can be used.

An output `STATUSx` is allocated to each rack. Each module (slot) of this rack is characterized by a bit of the corresponding `STATUSx` output. The bit on the far left-hand side in `STATUSx` corresponds to the slot on the far left-hand side of the rack x.

Using `STATUS1` to `STATUS5`:

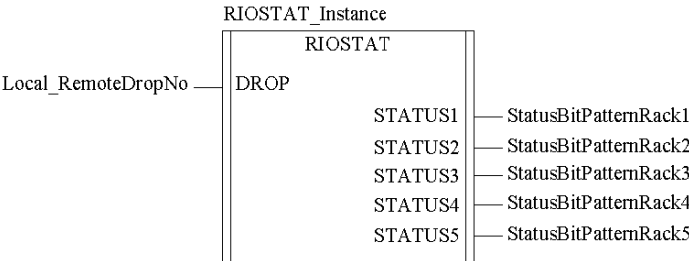
- **Quantum I/O**
There is only one rack for an I/O station, e.g. only `STATUS1` is used.
- **800 I/O**
There can be up to 5 racks for an I/O station, e.g. `STATUS1` corresponds to module rack 1, `STATUS5` corresponds to module rack 5.

NOTE: If a module on the module rack has been configured and works correctly, the corresponding bit is set to "1".

`EN` and `ENO` can be configured as additional parameters.

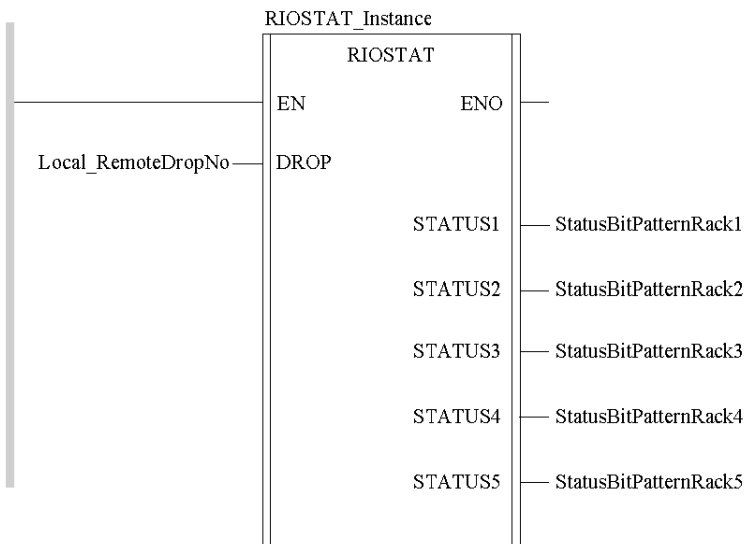
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL RIOSTAT_Instance (DROP:=Local_RemoteDropNo,  
  STATUS1=>StatusBitPatternRack1,  
  STATUS2=>StatusBitPatternRack2,  
  STATUS3=>StatusBitPatternRack3,  
  STATUS4=>StatusBitPatternRack4,  
  STATUS5=>StatusBitPatternRack5)
```

Representation in ST

Representation:

```
RIOSTAT_Instance (DROP:=Local_RemoteDropNo,  
  STATUS1=>StatusBitPatternRack1,  
  STATUS2=>StatusBitPatternRack2,  
  STATUS3=>StatusBitPatternRack3,  
  STATUS4=>StatusBitPatternRack4,  
  STATUS5=>StatusBitPatternRack5) ;
```

Parameter description

Description of the input parameters:

Parameters	Data type	Meaning
DROP	UINT	Local/remote I/O station no. (1...32)

Description of the output parameters:

Parameters	Data type	Meaning
STATUS1	WORD	Module rack 1 status bit pattern
STATUS2	WORD	Module rack 2 status bit pattern (800 I/O only)
...	...	...
STATUS5	WORD	Module rack 5 status bit pattern (800 I/O only)

Chapter 18

SET_TOD: Setting the hardware clock (Time Of Day)

Description

Function description

This function block searches (together with the other function blocks in the HSBY group) the configuration of the respective PLC for the necessary components. These components always refer to the hardware actually connected.

Therefore the correct functioning of this function block on the simulators cannot be guaranteed.

The function block sets the hardware system clock, if the corresponding registers are provided within this configuration. If these registers are not present, the `TOD_CNF` output is set to "0".

The function block reads the input values on the `S_PULSE` input at a rising edge and transfers them to the hardware clock.

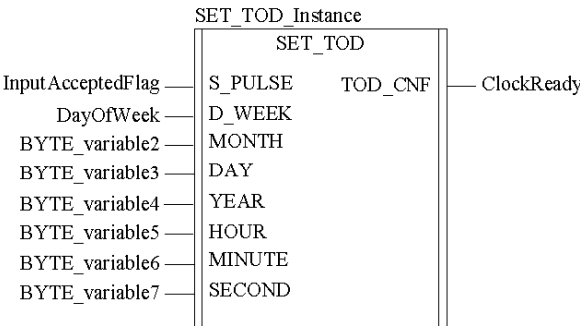
For all input values:

- If the value exceeds the specified maximum value, the maximum is used.
- If the value falls below the specified minimum value, the minimum is used.

`EN` and `ENO` can be configured as additional parameters.

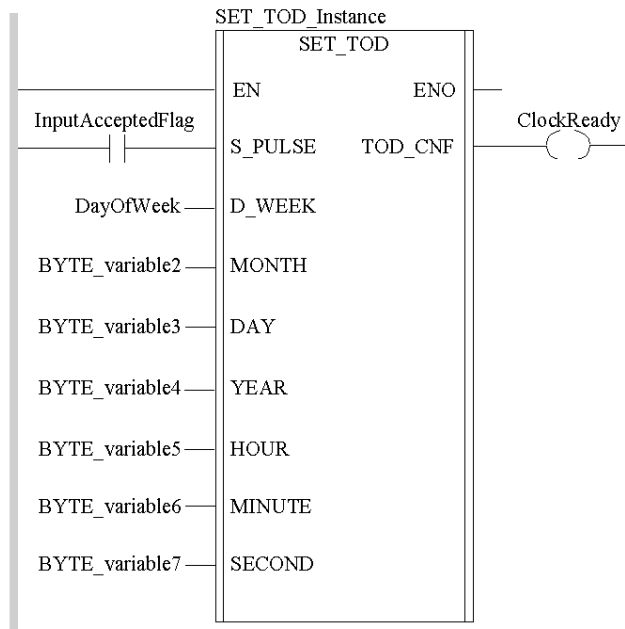
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL SET_TOD_Instance (S_PULSE:=InputAcceptedFlag,  
  D_WEEK:=DayOfWeek, MONTH:=Byte_variable2,  
  DAY:=Byte_variable3, YEAR:=Byte_variable4,  
  HOUR:=Byte_variable5, MINUTE:=Byte_variable6,  
  SECOND:=Byte_variable7, TOD_CNF=>ClockReady)
```

Representation in ST

Representation:

```
SET_TOD_Instance (S_PULSE:=InputAcceptedFlag,  
  D_WEEK:=DayOfWeek, MONTH:=Byte_variable2,  
  DAY:=Byte_variable3, YEAR:=Byte_variable4,  
  HOUR:=Byte_variable5, MINUTE:=Byte_variable6,  
  SECOND:=Byte_variable7, TOD_CNF=>ClockReady) ;
```

Parameter description

Description of the input parameters:

Parameters	Data type	Meaning
S_PULSE	BOOL	"0 -> 1" = the input values are accepted and written into the clock.
D_WEEK	BYTE	Day of week, 1 = Sunday 7 = Saturday
MONTH	BYTE	Month 1..12
DAY	BYTE	Day 1..31
YEAR	BYTE	Year 0..99
HOUR	BYTE	Hour 0..23
MINUTE	BYTE	Minute 0..59
SECOND	BYTE	Second 0..59

Description of the output parameters:

Parameters	Data type	Meaning
TOD_CNF	BOOL	"1" = %MW register (4x) for the hardware system clock was found and the clock is operational. "0" = Time is currently being set or hardware clock was not found.

Chapter 19

WORD_AS_BYTE_DFB: Type conversion

Description

Function description

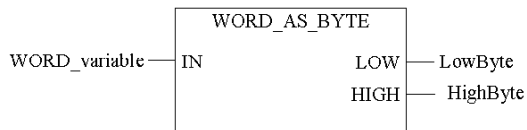
This derived function block converts one input word from the `WORD` data type to 2 output values of the `BYTE` data type.

The individual bytes of the word at the input are assigned to the outputs according to the output names.

`EN` and `ENO` can be configured as additional parameters.

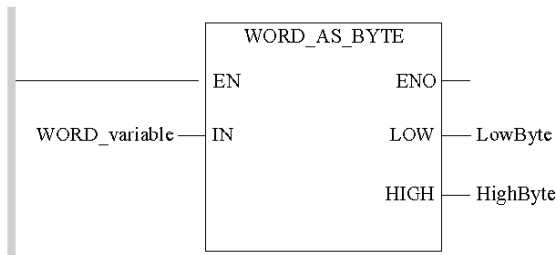
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL WORD_AS_BYTE_DFB_Instance (IN:=WORD_variable,  
    LOW=>LowByte, HIGH=>HighByte)
```

Representation in ST

Representation:

```
WORD_AS_BYTE_DFB_Instance (IN:=WORD_variable,  
    LOW=>LowByte, HIGH=>HighByte) ;
```

Parameter description

Description of the input parameters:

Parameter	Data type	Meaning
IN	WORD	Input

Description of the output parameters:

Parameter	Data type	Meaning
LOW	BYTE	least significant byte
HIGH	BYTE	most significant byte

Chapter 20

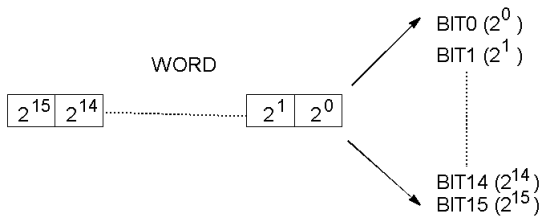
WORD_TO_BIT_DFB: Type conversion

Description

Function description

This derived function block converts one input word from the `WORD` data type to 16 output values of the `BOOL` data type.

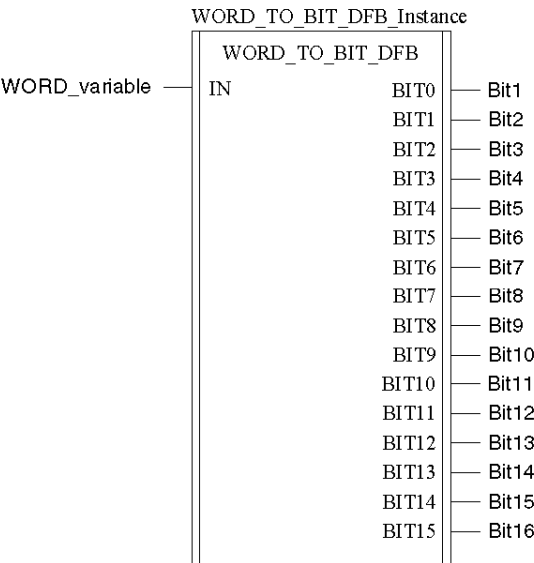
The individual bits of the word at the input are assigned to the outputs according to the output names.



`EN` and `ENO` can be configured as additional parameters.

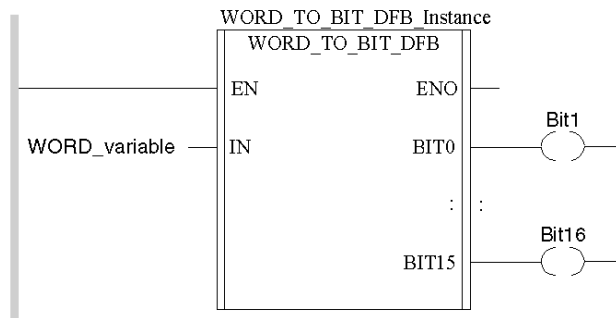
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL WORD_TO_BIT_DFB_Instance (IN:=WORD_variable,
    BIT0=>Bit1, BIT1=>Bit2, BIT2=>Bit3, BIT3=>Bit4,
    BIT4=>Bit5, BIT5=>Bit6, BIT6=>Bit7, BIT7=>Bit8,
    BIT8=>Bit9, BIT9=>Bit10, BIT10=>Bit11, BIT11=>Bit12,
    BIT12=>Bit13, BIT13=>Bit14, BIT14=>Bit15, BIT15=>Bit16)
```

Representation in ST

Representation:

```
WORD_TO_BIT_DFB_Instance (IN:=WORD_variable,
    BIT0=>Bit1, BIT1=>Bit2, BIT2=>Bit3, BIT3=>Bit4,
    BIT4=>Bit5, BIT5=>Bit6, BIT6=>Bit7, BIT7=>Bit8,
    BIT8=>Bit9, BIT9=>Bit10, BIT10=>Bit11, BIT11=>Bit12,
    BIT12=>Bit13, BIT13=>Bit14, BIT14=>Bit15,
    BIT15=>Bit16) ;
```

Parameter description

Description of the input parameters:

Parameter	Data type	Meaning
IN	WORD	Input

Description of the output parameters:

Parameter	Data type	Meaning
BIT0	BOOL	Output BIT0
BIT1	BOOL	Output BIT1
:	:	:
BIT15	BOOL	Output BIT15

Chapter 21

WRITEREG: Write register

Introduction

This chapter describes the WRITEREG block.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
Description	190
Mode of Functioning	193
Parameter description	194

Description

Function description

With a rising edge at the `REQ` input, this function block writes a register area from the PLC to an addressed slave via Modbus Plus.

`EN` and `ENO` can be configured as additional parameters.

NOTE: When programming a `WRITEREG` function, you must be familiar with the routing procedures used by your network. Modbus Plus routing path structures are described in detail in the Modbus Plus Network Planning and Installation Guide ([see page 9](#)).

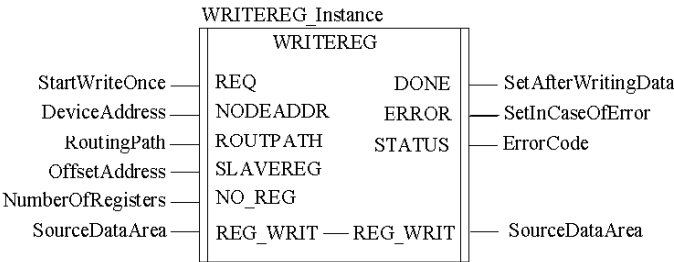
NOTE: This derived function block only supports the local Modbus Plus interface (no NOM). If using a NOM, work with the `WRITE_REG` block from the communication block library.

NOTE: This derived function block also does not support TCP/IP- or SY/MAX-Ethernet. If TCP/IP- or SY/MAX-Ethernet is needed, use the `WRITE_REG` block from the communication block library.

NOTE: Several copies of this function block can be used in the program. However, multiple instancing of these copies is not possible.

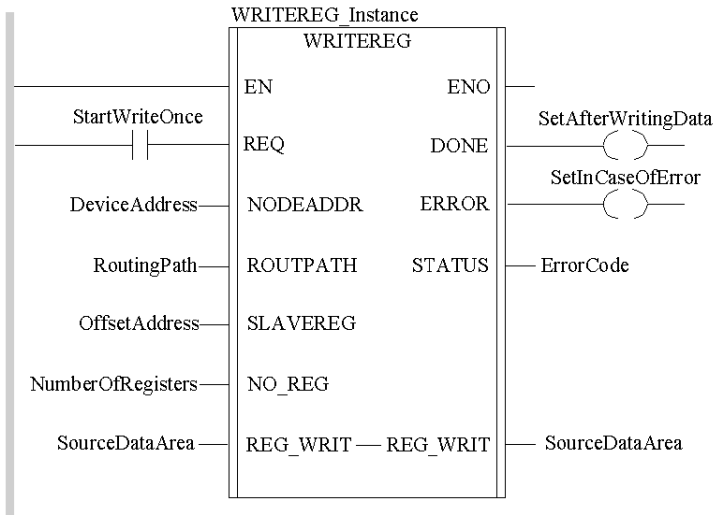
Representation in FBD

Representation:



Representation in LD

Representation:



Representation in IL

Representation:

```
CAL WRITEREG_Instance (REQ:=StartWriteOnce,
  NODEADDR:=DeviceAddress, ROUTPATH:=RoutingPath,
  SLAVEREG:=OffsetAddress, NO_REG:=NumberOfRegisters,
  REG_WRIT:=SourceDataArea,
  DONE=>SetAfterWritingData, ERROR=>SetInCaseOfError,
  STATUS=>ErrorCode)
```

Representation in ST

Representation:

```
WRITEREG_Instance (REQ:=StartWriteOnce,
  NODEADDR:=DeviceAddress, ROUTPATH:=RoutingPath,
  SLAVEREG:=OffsetAddress, NO_REG:=NumberOfRegisters,
  REG_WRIT:=SourceDataArea,
  DONE=>SetAfterWritingData, ERROR=>SetInCaseOfError,
  STATUS=>ErrorCode) ;
```

Parameter description

Description of input parameters:

Parameter	Data type	Meaning
REQ	BOOL	With a rising edge at the REQ input, this function block writes a register area from the PLC to an addressed slave via Modbus Plus.
NODEADDR	INT	Device address within the target segment
ROUTPATH	DINT	Routing path to target segment
SLAVEREG	DINT	Offset address of the first 4x register in the slave to be written to
NO_REG	INT	Number of registers to be written from slave

Description of input / output parameters:

Parameters	Data type	Meaning
REG_WRIT	ANY_ARRAY_WORD	Source data field (A data structure must be declared as a located variable for the source file.)

Description of the output parameters:

Parameters	Data type	Meaning
DONE	BOOL	Set to 1 for one scan after writing data
ERROR	BOOL	Set to 1 for one scan in case of error
STATUS	WORD	Error Code

Mode of Functioning

Function mode of WRITEREG blocks

Although a large number of WRITEREG function blocks can be programmed, only four write operations may be active at the same time. It makes no difference whether these operations are performed using this function block or others (e.g. MBP_MSTR, CWRITE_REG). All function blocks use one data transaction path and require multiple cycles to complete a task.

If several WRITEREG function blocks are used within an application, they must at least differ in the values of their NO_REG or REG_WRIT parameters.

The status signals DONE and ERROR report the function block state to the user program.

The complete routing information must be separated into two parts:

- in the NODEADDR of the destination node (regardless of whether it is located in the local segment or in another segment) and
- the routing path, in case there is a link via network bridges.

The resulting destination address consists of these two information components.

The routing path is a DINT data type, which is interpreted as a sequence of two-digit information units. It is not necessary to use "00" extensions (e.g. both routing paths 4711 and 47110000 are valid, for NODEADDR 34 the result is destination address 47.11.34.00.00).

Parameter description

REQ

A rising edge triggers the write transaction.

The parameter can be entered as an address, located variable, unlocated variable or literal.

NODEADDR

Identifies the node address within the target segment.

The parameter can be entered as an address, located variable, unlocated variable or literal.

ROUTPATH

Identifies the routing path to the target segment. The two-digit information units run from 01 64 (see *Mode of Functioning*, [page 193](#)). If the slave resides in the local network segment, ROUTPATH must be set to "0" or must be left unconnected.

The parameter can be entered as an address, located variable, unlocated variable or literal.

SLAVEREG

Start of the destination area in the addressed slave to which the source data is written. The destination area always resides within the 4x register area. SLAVEREG expects the destination address as an offset within the 4x area. The leading "4" must be omitted (e.g. 59 (contents of the variables or value of the literal) = 40059).

The parameter can be entered as an address, located variable, unlocated variable or literal.

NO_REG

Number of registers to be written to slave processor (1 ... 100).

The parameter can be entered as an address, located variable, unlocated variable or literal.

REG_WRIT

An ANY_ARRAY_WORD that is the same size as the planned transmission must be agreed upon ($\geq$ NO_REG) for this parameter. The name of this array is defined as a parameter. If the array is defined too small, then only the amount of data is transmitted that is present in the array.

The parameter must be defined as a located variable.

DONE

Transition to ON state for one program scan signifies data have been transferred.

The parameter can be entered as an address, located variable or unlocated variable.

ERROR

Transition to ON state for one program cycle signifies detection of a new error.

The parameter can be specified as an address, located variable or unlocated variable.

STATUS

Error code, see *Modbus Plus Error Codes*, [page 125](#)

The parameter can be specified as an address, located variable or unlocated variable.

Appendices



Overview

This section contains the appendices.

What Is in This Appendix?

The appendix contains the following chapters:

Chapter	Chapter Name	Page
A	FAQ Build Errors	199
B	FAQ Conversion Errors	225

Appendix A

FAQ Build Errors

Overview

This chapter includes information on build errors.

What Is in This Chapter?

This chapter contains the following topics:

Topic	Page
General	200
Object Link Creation Error	201
Object Must be Connected to a Successor	202
Link Together with Variable isn't Allowed	204
Data Type 'xxxx' Expected	205
Empty DFB to Replace Obsolete EFB	210
Undefined Symbol 'xxxx'	211
Call of Non-Function Block	213
Parameter 'xxxx' Has to Be Assigned	216
' xxxx' Is Not a Parameter of 'yyyy'	217
DDT Component Is Missing	218
EHC Parameters Out of Range	219
Not a Valid Address	220
140 NOG 111 00 Configuration Not Converted	221
E1163 Use of Unconfigured Direct Address	222
The Instance Is Located on an Address That Is Not Configured	223

General

Overview

After converting a Concept application, the **Rebuild All** Menu should be invoked.

If the application is not built with this command, all error messages in the build output window should be examined by double-clicking on them. This opens the section with the origin of the problem.

The whole section should be compared to the original in Concept and functional differences should be manually corrected in the converted application.

Example

Examples for messages:

- {SCADA_Info : [MAST]} : (r: 172, c: 4) E1218 Object must be connected to a successor, at least the Right-Power-Rail
- {FC124_Visual_call_up_part_3 : [MAST]} : (r: 31, c: 5) E1189 converter error: 'Object Link creation error (Link pin can not be located in original object) : Link to pin (linkSource: row=30, col=4, Object=, Pin=OUT1.) can not be created. Object has not been created during import.'

Potential Messages

Short forms of potential messages are given in the following list, which are linked to explanation details:

- *Object Link Creation Error*, [page 201](#)
- *Object Must be Connected to a Successor*, [page 202](#)
- *Link Together with Variable isn't Allowed*, [page 204](#)
- *Data Type 'xxxx' Expected*, [page 205](#)
- *Empty DFB to Replace Obsolete EFB*, [page 210](#)
- *Undefined Symbol 'xxxx'*, [page 211](#)
- *Call of Non-Function Block*, [page 213](#)
- *Parameter 'xxxx' Has to Be Assigned*, [page 216](#)
- *'xxxx' Is Not a Parameter of 'yyyy'*, [page 217](#)
- *DDT Component Is Missing*, [page 218](#)
- *EHC Parameters Out of Range*, [page 219](#)
- *Not a Valid Address*, [page 220](#)

Object Link Creation Error

Cause

One reason of this message, which occurs during import already and when analyzing, can be that the converter does not have implemented the substitution of the extensible diagnostic blocks with dual FBs.

Explanation

D_GRP and D_PRE both need an AND block attached to their IN input. This additional AND has to be implemented so, that it gets all inputs of the former extensible area. Add the missing block by hand.

Example

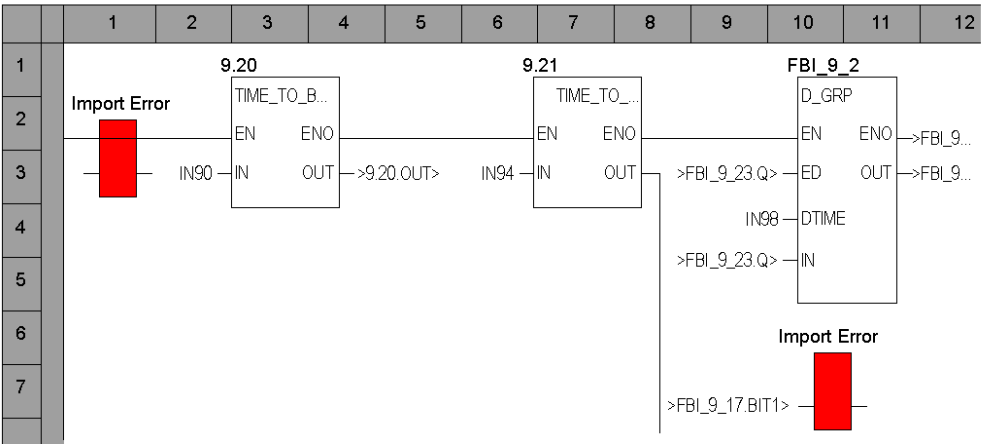
```
{_9_TIME : [MAST]} : (r: 2, c: 1) E1189 converter error: 'Object Link creation error (Link pin can not be located in original object) : Link to pin (linkSource: row=1, col=0, Object=FBI_9_2_DRAW, Pin=OUT.) can not be created. Object has not been created during import.'
```

```
{_9_TIME : [MAST]} : (r: 2, c: 1) E1002 syntax error
```

```
{_9_TIME : [MAST]} : (r: 6, c: 13) E1189 converter error: 'Object Link creation error (Link pin can not be located in original object) : Link to pin (linkDestination: row=5, col=12, Object=FBI_9_2, Pin=.) can not be created. Object has not been created during import.'
```

```
{_9_TIME : [MAST]} : (r: 6, c: 13) E1002 syntax error
```

Figure



Object Must be Connected to a Successor

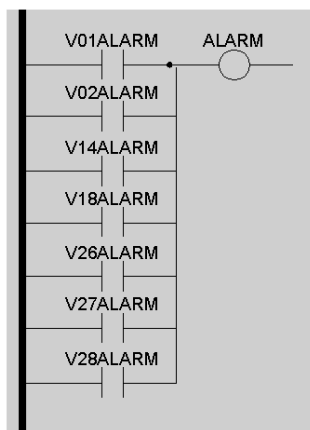
Cause

A message as follows, can have its reason in a Concept 2.1 LD bug:

```
{TANKVLVS <DFB> : [TVALVE]} : (r: 93, c: 3) E1218 Object must be connected to a successor, at least the Right-Power-Rail
```

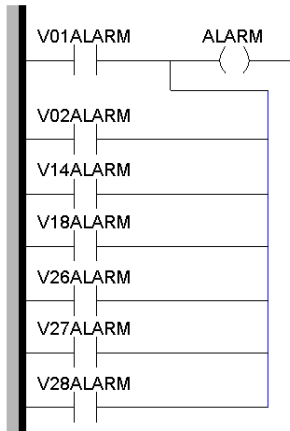
When connecting contacts to an OR (a vertical short), it sometimes happens that the intended first output contact is connected to the input of the OR.

Concept even shows this in its graphics with a small dot at the input of the OR:



In this case, the ALARM coil is connected ONLY to the contact V01ALARM. The OR output is connected to NOTHING.

Consequently, the Unity V1.1 converter translates this to:

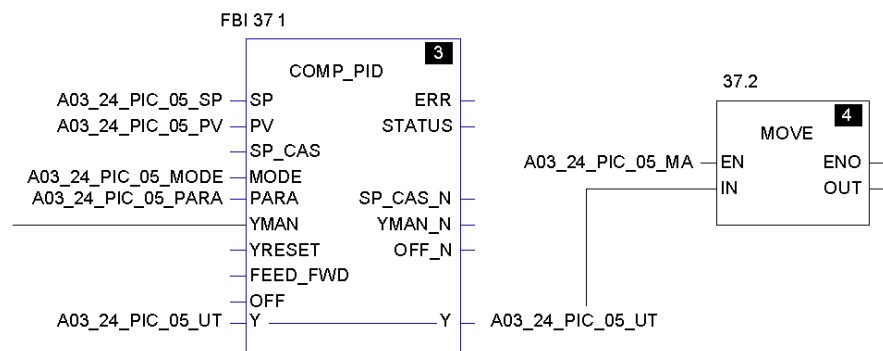


Link Together with Variable isn't Allowed

Overview

This error is reported in connection with **INOUT** pins.

Example

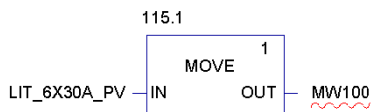


Solution

Delete the link and insert the variable to the destination parameter of the link.

Data Type 'xxxx' Expected

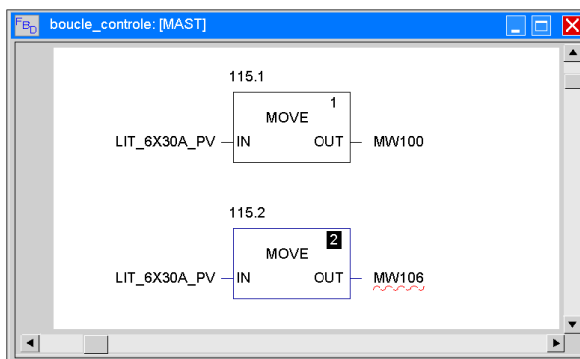
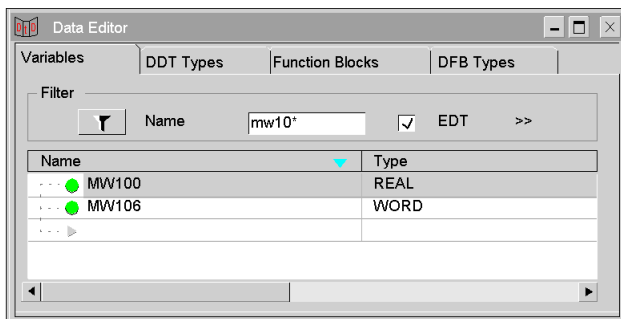
Example



Solution

Replace used data types according to the required type.

The following picture shows the error correction for the 115.1 function block, where the type of output (MW100) has been modified to the type used for the input (REAL).



The Concept converter of V1.0 estimates the type from the address and does not take the actual type into account. This is subject to a later version.

Retyping EFB Parameter

Another reason of this message can be e.g., that the EFB parameter has been retyped to ANY_ARRAY_WORD. See also Parameter type changed.

Combinations of Variables, Variable Instances and Parameters from Concept

Coming from Concept are the following combinations of variables, variable instances and parameters (pins):

Element	Description
Variable Declarations	- The variable declaration has a type of its own and can have a register - Several variable declarations can have individual types and the same register <pre>CP_GV1 "Symbol" 4:100 DPM_Time INIT: FALSE 0 EXP: FALSE RET: FALSE READONLY: FALSE MAS: FALSE TEXT: CP_GV1 "SymbolElem" 4:100 IEC_INT_ID INIT: FALSE 0 EXP: FALSE RET: FALSE READONLY: FALSE MAS: FALSE TEXT:</pre>
Variable Instances	- Variable INSTANCES coming with a SYMBOL have no own type and no register and use the type of the necessary variable declaration. <pre>CP_GVI NAMED_VAR: "SymbolElem" 10 9 FP_IO_OUTPUT</pre> - The variable instance can come with a register, in this case it has a type of its own in the instance declaration and no symbol. <pre>CP_GVI REG_VAR: 4:100 27 16 FP_IO_INPUT DPM_Time EXP: FALSE RET: TRUE MAS: FALSE</pre> - It is not necessary to have a variable declaration for register variable instances: - Textual anonymous declarations (AT %MWxx:DDT;) are equivalent to variable instance declarations with register and declare the type as well. - The type is forced to be the same as an existing variable declaration. If they are conflicting, the declaration is refused in Concept. <pre>CP__ST AT %QW102: REAL;</pre>
Parameters	The pin a variable instance is attached to has a type of its own, which is not necessarily the same as that of the variable instance. It cannot be changed and can be generic. <pre>VS_FRM "IN1" HIDE POSL 2 FP_IO_INPUT FP_INP_NORMAL FP_LOC_OUTSIDE INT TEXT: VS_FRM "IN" HIDE POSL 2 FP_IO_INPUT FP_INP_NORMAL FP_LOC_OUTSIDE ANY TEXT:</pre>

NOTE: So there are three + n different types possible to be declared for a register variable in Concept (1(2=>n),4,6) .

Type Declaration in Unity

Unity accepts one type declared with a symbol associated to a register. If the register is used directly, only its default type is assumed.

To generate code, the type and size of a variable attached to a pin must be determined to one type. Different pins might have different types.

Register Variable Instance

If there is a register variable instance with its type and additionally a variable declaration with a different type and the same register, Concept generates code according to the type supplied with the register or with the symbol individual for each pin.

Default Type

Unity knows only a default type for registers. If this type is to be changed, a variable with a symbol must be declared to carry the type, but two symbols with different types for one register are not accepted.

Unity does not import the second variable, if this application is imported.

Behaviour of Variables, Variable Instances and Parameters in Unity

If...	And There Is...	Then...
symbols are used with a variable instance	-	- the type declared in the declaration with the symbol is to be used - the type of a possibly present register variable instance is not
a register variable with a type different from the default is to be used	already a variable declaration with the same register, but a different type	an error message for this impossibility is issued.
a register variable with a type different from the default is to be used	a variable declaration with the same register with the same type	its symbol is to be used instead of the direct address.
a register variable with a type different from the default is to be used	no variable declaration with the same register	an artificial symbol is to be declared and used instead of the direct address.
a pin in the Unity template has the type <code>ANY_ARRAY_WORD</code>	-	an attached register variable could get the type <code>ARRAY [0..0] OF WORD</code> , if it previously had the type <code>WORD</code> .
the register is used also at pins with the type <code>WORD</code>	-	the register gets the index [0] attached.

Other Type Mismatch Cases

Other type mismatch cases are reported with a build(=analyze) message and left to be resolved by the user.

Word Arrays in Communication Blocks

Communication blocks have Word arrays as parameters, whic are defined in Concept with a reference to the first Word only.

The size of the array is often given by the content of a variable, which is defined during run-time. So the size cannot be determined by the converter.

The user must determine the maximum size and declare the array accordingly by himself.

Example from Concept

Step	Action
1	STMicroScaleConfig: [MAST] <pre>('Micro Scale Clear-Tare (402310 - 402319) ') IF MicroScCTActive = 0 THEN MicroScCTMSTR1 := 1 ; MicroScCTMSTR2 := 0 ; MicroScCTMSTR3 := 3 ; MicroScCTMSTR4 := 47 ; MicroScCTMSTR5 := 1028 ; MicroScCTMSTR6 := 1 ; MicroScCTMSTR7 := 0 ; MicroScCTMSTR8 := 0 ; MicroScCTMSTR9 := 0 ; MicroScCTMSTR10 := 62 ; END_IF ;</pre> All array members appear as single variables. In Unity, they must be combined to an array.
2	This is prepared by the converter by declaring a variable with the range of [0..0].

Step	Action																																												
3	This leads to a set of analyze messages to make the user aware of the correction need. In this case, the correction of the user should look like: ST MicroScaleConfig: [MAST] <pre>(*Micro Scale Clear-Tare (402310 - 402319) *) IF MicroScCTActive = 0 THEN MicroScCTMSTR [1] := 1; MicroScCTMSTR [2] := 0; MicroScCTMSTR [3] := 3; MicroScCTMSTR [4] := 47; MicroScCTMSTR [5] := 1028; MicroScCTMSTR [6] := 1; MicroScCTMSTR [7] := 0; MicroScCTMSTR [8] := 0; MicroScCTMSTR [9] := 0; MicroScCTMSTR10 [1] := 62; END_IF;</pre> Data Editor VariablesDDT TypesFunction BlocksDFB Types Filter Name * ☒ EDT ☒ DDT ☒ IODDT <table><thead><tr><th>Name</th><th>Type</th><th>Address</th><th>Value</th></tr></thead><tbody><tr><td> MicroScCTMSTR10</td><td>ARRAY[1..1] OF WORD</td><td>%MW2319</td><td></td></tr><tr><td> MicroScCTMSTR9</td><td>WORD</td><td>%MW2318</td><td></td></tr><tr><td> MicroScCTMSTR8</td><td>WORD</td><td>%MW2317</td><td></td></tr><tr><td> MicroScCTMSTR7</td><td>WORD</td><td>%MW2316</td><td></td></tr><tr><td> MicroScCTMSTR6</td><td>WORD</td><td>%MW2315</td><td>1</td></tr><tr><td> MicroScCTMSTR5</td><td>WORD</td><td>%MW2314</td><td>1028</td></tr><tr><td> MicroScCTMSTR4</td><td>WORD</td><td>%MW2313</td><td>47</td></tr><tr><td> MicroScCTMSTR3</td><td>WORD</td><td>%MW2312</td><td>3</td></tr><tr><td> MicroScCTMSTR2</td><td>WORD</td><td>%MW2311</td><td></td></tr><tr><td> MicroScCTMSTR</td><td>ARRAY[1..9] OF WORD</td><td>%MW2310</td><td></td></tr></tbody></table> 4 The source code related to this is in this case: <pre>{RESET_CONV <DFB> : [RC_HSK_1]} : (r: 2, c: 2) E1063 call of non-function block {RESET_CONV <DFB> : [RC_HSK_1]} : (r: 4, c: 29) E1067 'Q' is not a member of 'SECT_CTRL'</pre>	Name	Type	Address	Value	MicroScCTMSTR10	ARRAY[1..1] OF WORD	%MW2319		MicroScCTMSTR9	WORD	%MW2318		MicroScCTMSTR8	WORD	%MW2317		MicroScCTMSTR7	WORD	%MW2316		MicroScCTMSTR6	WORD	%MW2315	1	MicroScCTMSTR5	WORD	%MW2314	1028	MicroScCTMSTR4	WORD	%MW2313	47	MicroScCTMSTR3	WORD	%MW2312	3	MicroScCTMSTR2	WORD	%MW2311		MicroScCTMSTR	ARRAY[1..9] OF WORD	%MW2310	
Name	Type	Address	Value																																										
MicroScCTMSTR10	ARRAY[1..1] OF WORD	%MW2319																																											
MicroScCTMSTR9	WORD	%MW2318																																											
MicroScCTMSTR8	WORD	%MW2317																																											
MicroScCTMSTR7	WORD	%MW2316																																											
MicroScCTMSTR6	WORD	%MW2315	1																																										
MicroScCTMSTR5	WORD	%MW2314	1028																																										
MicroScCTMSTR4	WORD	%MW2313	47																																										
MicroScCTMSTR3	WORD	%MW2312	3																																										
MicroScCTMSTR2	WORD	%MW2311																																											
MicroScCTMSTR	ARRAY[1..9] OF WORD	%MW2310																																											

Empty DFB to Replace Obsolete EFB

Cause

A few standard EFB have not been ported from Concept to Unity.

If the Converter finds one of them, it inserts an Empty DFB with the same parameters as the original to allow building of the application and to give the possibility to the user to substitute the original with code of his own.

Solution

Insert the code into the body of the empty DFB, which contains the command to generate a message like the following in it:

```
{S1 : [REAL_W2]} : (r: 1, c: 2) E1189 converter error: 'Empty DFB to  
replace obsolete EFB - fill by user'
```

The command for the message has to be deleted, if valid code has been filled into the DFB body to allow building of the application.

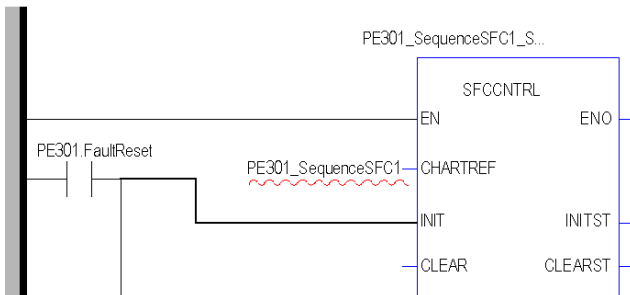
NOTE: Because the ANY type is not allowed on DFBs in Unity, an additional problem occurs if this type has been used in the substituted EFB (e.g. XXMIT EFB).

Customer Defined EFBs

Customer defined EFBs are not converted. If you want to convert an application using Customer defined EFBs, get in contact with Schneider support.

Undefined Symbol 'xxxx'

Wrong SFC Section Name

Explanation	This message is given sometimes in conjunction with the block SFCCNTRL. The message means, that the SFC section name, which has to be attached to the CHARTREF input is not the name of an existing SFC section inside of the current application.
Solution	Create the according section and the message disappears.  The diagram shows a ladder logic network. On the left, a vertical grey bar represents a power rail. A normally open contact labeled 'PE301.FaultReset' is connected to this rail. The output of this contact is connected to the 'EN' input of a block labeled 'SFCCNTRL'. Below the 'EN' input, there is a branch that goes to the 'INIT' input of the same block. This branch contains a red wavy line and is labeled 'PE301_SequenceSFC1_'. The 'CHARTREF' input of the 'SFCCNTRL' block is also connected to this same branch. The 'SFCCNTRL' block has four inputs: 'EN', 'CHARTREF', 'INIT', and 'CLEAR'. It has four corresponding outputs: 'ENO', 'INITST', and 'CLEARST' (the 'ENO' output is not explicitly labeled with a line). The 'CLEAR' input is shown as a blue line.

Wrong Instance Name

Explanation	Another reason of such a message can be, that a Concept function block now in Unity is a function or a procedure. The converter does conversion work in the textual languages ST and IL half-automatic in the case of this incompatibility. The instance name of Concept is removed and substituted by the type name, also for addressing outputs, which is then an illegal syntax: <pre>LOOKUP_TABLE1(X := ODT, XiYi1 := -30.0, XiYi2 := PARA.p1, XiYi3 := -20.0, XiYi4 := PARA.p2, XiYi5 := -10.0, XiYi6 := PARA.p3, XiYi7 := 0.0, XiYi8 := PARA.p4, XiYi9 := 10.0, XiYi10:= PARA.p5, XiYi11:= 20.0, XiYi12:= PARA.p6); OUT := LOOKUP_TABLE1.Y;</pre>
Solution	The last line must be corrected manually. Using the output assign operator, this statement must be changed and moved inside the call parentheses: <pre>LOOKUP_TABLE1(X := ODT, XiYi1 := -30.0, XiYi2 := PARA.p1, XiYi3 := -20.0, XiYi4 := PARA.p2, XiYi5 := -10.0, XiYi6 := PARA.p3, XiYi7 := 0.0, XiYi8 := PARA.p4, XiYi9 := 10.0, XiYi10:= PARA.p5, XiYi11:= 20.0, XiYi12:= PARA.p6, Y=>OUT);</pre>

Call of Non-Function Block

Cause

This message can appear, when a Concept function block now in Unity is a function or a procedure.

The converter removes the instance name of the Concept-block and substitutes it with the type name and moves assignments of outputs inside the invocation parentheses.

For the blocks GET_BIT and SET_BIT, the treatment does not completely apply. During analysis, messages occur:

```
{INPUTS : [MAST]} : (r: 7, c: 4) E1063 call of non-function block
```

The function names remain marked as erroneous after conversion, because the functions are converted with the procedure syntax in ST, not with the needed function syntax, as the corrected version shows. Also, the converter has dropped the indices for the result variable of GET_BIT.

Example

Original Concept Codes	After Conversion	Corrected Version
<pre>VAR INPUT_WORD : GET_BIT; END_VAR; FOR I_BASE := 1 TO 20 DO FOR I_POINT := 1 TO 16 DO INPUT_WORD (IN:=IO_SCAN_IN_WORD[I_BASE] , NO:=I_POINT); INPUT[I_BASE,I_POINT] := INPUT_WORD.RES; END_FOR; END_FOR;</pre>	<pre>FOR I_BASE := 1 TO 20 DO FOR I_POINT := 1 TO 16 DO GET_BIT(IN:=IO_SCAN_IN_WORD [I_BASE], NO:=I_POINT, RES => INPUT); ; END_FOR;</pre>	<pre>FOR I_BASE := 1 TO 20 DO FOR I_POINT := 1 TO 16 DO INPUT[I_BASE,I_POINT]:= GET_BIT(IN:=IO_SCAN_IN_WORD [I_BASE], NO:=I_POINT); END_FOR; END_FOR;</pre>
<pre>VAR OUTPUT_WORD : SET_BITX; END_VAR; FOR O_BASE := 1 TO 20 DO FOR O_POINT := 1 TO 16 DO OUTPUT_WORD (RES := IO_SCAN_OUT_WORD[O_BASE], IN :=OUTPUT[O_BASE,O_POINT], NO := O_POINT); END_FOR; END_FOR;</pre>	<pre>FOR O_BASE := 1 TO 20 DO FOR O_POINT := 1 TO 16 DO SET_BIT(RES := IO_SCAN_OUT_WORD[O_BASE], IN := OUTPUT[O_BASE,O_POINT], NO := O_POINT); END_FOR; END_FOR;</pre>	<pre>FOR O_BASE := 1 TO 20 DO FOR O_POINT := 1 TO 16 DO IO_SCAN_OUT_WORD[O_BASE]:= SET_BIT(IN := OUTPUT[O_BASE,O_POINT], NO := O_POINT); END_FOR; END_FOR;</pre>

Conjunction With Other Messages

This message can appear in conjunction with other messages:

- {RESET_CONV <DFB> : [RC_HSK_1]} : (r: 2, c: 2) E1063 call of non-function block
- {RESET_CONV <DFB> : [RC_HSK_1]} : (r: 4, c: 29) E1067 'Q' is not a member of 'SECT_CTRL'

Related Source Code

The source code related to this is in this case:

- RESET_CONV147(IN := (CTRL.TB.RC_INI AND V_SYNCHRO), PT := t#500ms);
- T_CONVRESET := RESET_CONV147.Q;

Double Use of the Instance Name

Unity Pro associates the instance name to the derived data type SECT_CTRL, even though it is intended to address a timer. This usually happens, if the Concept application used the instance name twice. To find this out, proceed as follows:

Step	Action
1	Open the Concept .asc export file, and search the instance name without the figures at the end with a search command of the text editor. Result: In this case here we find: <pre>STR_RCI: (* RC Eingänge = SPS Ausgänge *) STRUCT AUTO : BOOL ; (* Betriebsart Automatik / Hand *) AXIS_EN : BOOL ; (* Achsen angewählt *) Z_UP : BOOL ; (* Z-Achse auf *) RESET_PROG : BOOL ; (* Programm abbrechen *) RESET_CONV : BOOL ; (* Förderer synchronisieren *)</pre>
2	The line introduction...STR has been omitted. <pre>CP_GVS "RESET_CONV" SECT_CTRL INIT: FALSE 0 EXP: FALSE RET: TRUE READONLY: FALSE MAS: FALSE TEXT: CP_SEC "RESET_CONV" SECTK_F_SECTION LANG_ST SVB: FALSE ID: 27 EXEC: 26 TEXT: CP__ST CP__ST VAR CP__ST RESET_CONV : TP; (* Impuls Reset Conveyor *)</pre> The same name has been used as a structure component name, a section name with its control variable, and for a "TP" timer instance.
3	Change the type of the instance in the Data editor to "TP".

Substitute Procedures in ST/IL

Some EFBs from Concept are implemented as procedures in Unity Pro without instance names.

Open the **Conversion Settings** tab via **Tools** → **Options** in Unity Pro to enable/disable the **Substitute Procedures in ST/IL** check box before converting.

- When this checkbox is enabled, the instance name of the Concept call will be replaced with the type name.
- When this checkbox is not enabled, a DFB will be created, which will then access the procedure.

Parameter 'xxxx' Has to Be Assigned

Cause

For inputs, left open pins at blocks get an automatically generated variable with the appropriate type.
For outputs, this is not done yet.

In the case of generic data types, it cannot be done easily.

Solution

In these cases the user still must declare appropriate variables and attach them to the left-open pins.

' xxxx' Is Not a Parameter of 'yyyy'

Cause

The diagnostic EFBs, which have been extensible in Concept, do not get the right calling syntax in IL.

```
{_9_TIME : [MAST]} : (r: 43, c: 17) E1031 'IN1' is not a parameter of
function block 'GRP_DIA_9'
```

```
{_9_TIME : [MAST]} : (r: 44, c: 17) E1031 'IN2' is not a parameter of
function block 'GRP_DIA_9'
```

Solution

In the case of the in Concept extensible diagnostic EFBs, the extension can be done with a logical AND function the output of which is tied to the single input of the diagnostic function. This is done with the first three lines in the correction.

The used output must be processed by `BOOL_TO_TIME`, which is bypassed in the automatic conversion and which is corrected in the last three lines.

Example

Original Concept Code	After Conversion	Corrected Version
CAL GRP_DIA_9 (ED :=DUMMY_1_91, DTIME :=IN92, IN1 :=DUMMY_1_94, IN2 :=DUMMY_1_96) LD GRP_DIA_9.ERR BOOL_TO_TIME ST OUT90	CAL GRP_DIA_9 (ED :=DUMMY_1_91, DTIME :=IN92, IN1 :=DUMMY_1_94, IN2 :=DUMMY_1_96, ERR => OUT90) BOOL_TO_TIME	LD DUMMY_1_94 AND DUMMY_1_96 ST GRP_DIA_9.IN CAL GRP_DIA_9 (ED :=DUMMY_1_91, DTIME :=IN92) LD GRP_DIA_9.ERR BOOL_TO_TIME ST OUT90

DDT Component Is Missing

Cause

Keywords may not be used as symbols of DDT components or as variable names. Such a case is the name slot.

Solution

If DDT components are missing or import conflicts are imported, proceed as follows:

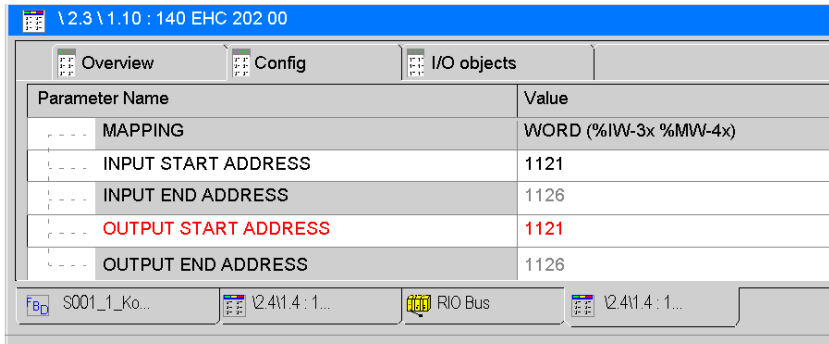
Step	Action
1	Search for the occurrence of the name in the .asc file with another meaning.
2	Change the name for the conflicting meaning.

EHC Parameters Out of Range

Cause

For the High speed counter module, parameter limits are not treated right.

Example



Parameter Name	Value
MAPPING	WORD (%IW-3x %MW-4x)
INPUT START ADDRESS	1121
INPUT END ADDRESS	1126
OUTPUT START ADDRESS	1121
OUTPUT END ADDRESS	1126

The screenshot shows the SIMATIC Manager HW Config window for station \2.3\1.10:140 EHC 202 00. The 'I/O objects' tab is selected, showing a table of parameters. The 'OUTPUT START ADDRESS' is highlighted in red, indicating an error. The bottom status bar shows the project name 'S001_1_Ko...' and the hardware configuration '2 4x1.4:1...' for the 'RIO Bus'.

Parameter <OUTPUT START ADDRESS> out of range (Error with param 17)

Solution

Such Parameters must be manually corrected.

Not a Valid Address

Cause

A message like the following one is generated at analyze time, if a Hot_Stand-By system is incompletely defined.

Analyzing...

```
{Cpu (1.2 ) 140 CPU 671 60} : %MW0 is not a valid address in Quantum
```

Solution

Step	Action
1	Open the local rack of the configuration and the CPU configuration itself and select the Hot Stand-By tab of the CPU configuration.
2	In its lower part appear State Ram and Non Transfer area . Usually both the Start and Length fiels both contain a Zero, directly passed through from the Concept application.
3	To remove the error message, enter 1 into the start field.

140 NOG 111 00 Configuration Not Converted

Concept

The 140 NOG 111 00 is used as a NOM in Concept.

Configuration Not Converted

The conversion creates also a NOM in Unity Pro but the I/O configuration gets lost.

E1163 Use of Unconfigured Direct Address

Description

Address values configured in the legacy application are higher than the maximum allowed **State RAM**.

NOTE: Quantum CPUs maximum value for **%M 0x State RAM**: 65280.

The Instance Is Located on an Address That Is Not Configured

Solution

Reserve more memory words in the CPU **State RAM**.

Appendix B

FAQ Conversion Errors

FAQ Conversion Errors

Overview

This section gives the list of keywords that must not be used to define object names (variables and types of variables).

If a Concept application contains such variable or variable type names, the application cannot be converted to Unity Pro.

The following message to be acknowledged by the user appear: **Error in definition of located variable: Keywords may not be used as variable names.**

Keywords

The following lists gives the keywords in alphabetical order:

A

- Address
- AI_CONSTANT
- AI_SECTION
- AI_VARIABLEINST
- AI_VARIABLEINST_REG
- ANY
- AsciiMsg
- ATTRIBUTE

B

- BCD16
- BCD32
- BCD64
- BCD8
- BEGIN
- binData
- BOOL
- byte

C

- CL_DFB
- CL_FRM
- CL_PLC
- COLUMN_WIDTH
- CONCEPT_

- CONCEPT_VERSION
- configImage
- configTable
- CP__IL
- CP__ST
- CP_ABR
- CP_ACT
- CP_AJN
- CP_APP
- CP_COM
- CP_CON
- CP_DBI
- CP_DFB
- CP_FBI
- CP_GEN
- CP_GV1
- CP_GV2
- CP_GV4
- CP_GVI
- CP_GVS
- CP_GVT
- CP_IFP
- CP_INV
- CP_JMP
- CP_LNK
- CP_OPT
- CP_PBR
- CP_PJN
- CP_PRG
- CP_PRI
- CP_PRO
- CP_PRP
- CP_SEC
- CP_STP
- CP_STR
- CP_TRN
- CP_VRS
- CR_END

D

- DATE
- dint
- DisplayFormat
- dpMasterData
- dpSlaveData
- DRAW
- DropHead
- dropNumber
- DT
- DWORD

E

- END
- END_ENTRY
- END_RDE_TEMPLATE
- ENTRY
- EVN
- extType

F

- FALSE
- FBIPH_INPUT
- FBIPH_OUTPUT
- FP_IO_INOUTPUT
- FP_IO_INOUTPUT_COMP
- FP_IO_INPUT
- FP_IO_INPUT_COMP
- FP_IO_OUTPUT
- FP_IO_OUTPUT_COMP

G

- global

H

- headIndex
- HEIGHT
- HIDE
- HX_n

I

- ID_{Digit}+
- IEC_BCD16_ID
- IEC_BCD32_ID
- IEC_BCD64_ID
- IEC_BCD8_ID
- IEC_BOOL_ID
- IEC_BYTE_ID

- IEC_DATE_ID
- IEC_DINT_ID
- IEC_DT_ID
- IEC_DWORD_ID
- IEC_INT_ID
- IEC_LINT_ID
- IEC_LREAL_ID
- IEC_LWORD_ID
- IEC_REAL_ID
- IEC_SINT_ID
- IEC_STRING_ID
- IEC_TIME_ID
- IEC_TOD_ID
- IEC_UDINT_ID
- IEC_UINT_ID
- IEC_ULINT_ID
- IEC_UNKNOWN_ID
- IEC_USINT_ID
- IEC_WORD_ID
- INIT
- inputBytes
- inputReference
- int
- INV
- IODrop
- IOModule

L

- LANG_FBD
- LANG_IL
- LANG_LD
- LANG_LL
- LANG_SFC
- LANG_ST
- LINT
- LL_INS
- LL_NET
- LL_NOD
- LL_REG
- LL_SON
- LL_SRD
- LL_SRM
- LL_VAR
- local
- locInc

- LREAL
- LWORD

M

- macAddr
- MAS
- maxConstant
- modData

N

- NAMED_VAR
- nodeParam

O

- outputBytes
- outputReference

P

- pcHealthTimeout
- pcHoldLastValue
- PlcCnfDb
- PLCConfig
- plcName
- POSR

R

- rack
- RDE_TEMPLATE
- RDE_TEMPLATE_VERSION
- READONLY
- Real
- REG_VAR
- RET
- rs232Params

S

- scratchPad
- SECTK_F_SECTION
- SetValue
- SFC_STEP_INIT
- SFC_STEP_NORMAL
- SINT
- SLOT
- STRING
- svcFile

T

- Tagname
- TEXT
- time
- TIMN
- TOD

U

- uint
- uint
- ULINT
- UNKNOWN
- USINT

V

- VAL
- Value
- VS_FFB
- VS_FRM

W

- WIDTH
- WINDOW_LOCATION
- WITHOUT_ATT
- WORD



0-9

140 NOG 111 00 configuration
not converted, 221

A

address
not configured, 223
analyzing
projects, 55, 103
application behavior
changes, 89

B

build errors, 199
BYTE_TO_BIT_DFB, 115

C

Concept
conversion wizard, 16
Concept Converter - instructions
BYTE_TO_BIT_DFB, 115
CREADREG, 119
CWRITREG, 127
DINT_AS_WORD_DFB, 133
DIOSTAT, 135
GET_TOD, 137
LIMIT_IND_DFB, 141
LOOKUP_TABLE1_DFB, 145
PLCSTAT, 151
READREG, 167
RIOSTAT, 175
SET_TOD, 179
WORD_AS_BYTE_DFB, 183
WORD_TO_BIT_DFB, 185
WRITEREG, 189
configuration
differences, 22

conversion errors, 225
conversion wizard for Concept), 16
converter, 11
conversion
procedure, 105
process, 103
CREADREG, 119
CWRITREG, 127

D

DINT_AS_WORD_DFB, 133
DIOSTAT, 135
direct address
E1163, 222

E

E1163
direct address, 222
EFBs
differences, 32
EN
not connected, 93
error messages, 55, 89, 103, 108
exporting
DFBs, 106
macros, 106
projects, 13, 106
sections, 106

F

Function Block Diagram
differences, 53

G

GET_TOD, 137

H

hardware
 correspondences, 21
hardware platforms
 supported, 21

I

importing
 DDTs, 108
 macros, 55, 110
 projects, 13, 107
initialization values
 array, 111
 cluster, 111
 LL_SRAMxxx, 111
Instruction List
 differences, 50
instructions
 differences, 32

L

Ladder Diagram
 differences, 36
Ladder Logic
 differences, 52
language objects, 55
 differences, 23
LIMIT_IND_DFB, 141
LL_SRAMxxx
 array, 111
 initialization values, 111
LOOKUP_TABLE1_DFB, 145

N

not configured
 address, 223

O

object types
 differences, 55

P

PLCSTAT, 151
preconditions, 19
program execution
 differences, 23

R

READREG, 167
requirements, 19
RIOSTAT, 175

S

Sequential Function Chart
 differences, 35
SET_TOD, 179
Structured Text
 differences, 50
system objects
 differences, 23

W

WORD_AS_BYTE_DFB, 183
WORD_TO_BIT_DFB, 185
WRITEREG, 189