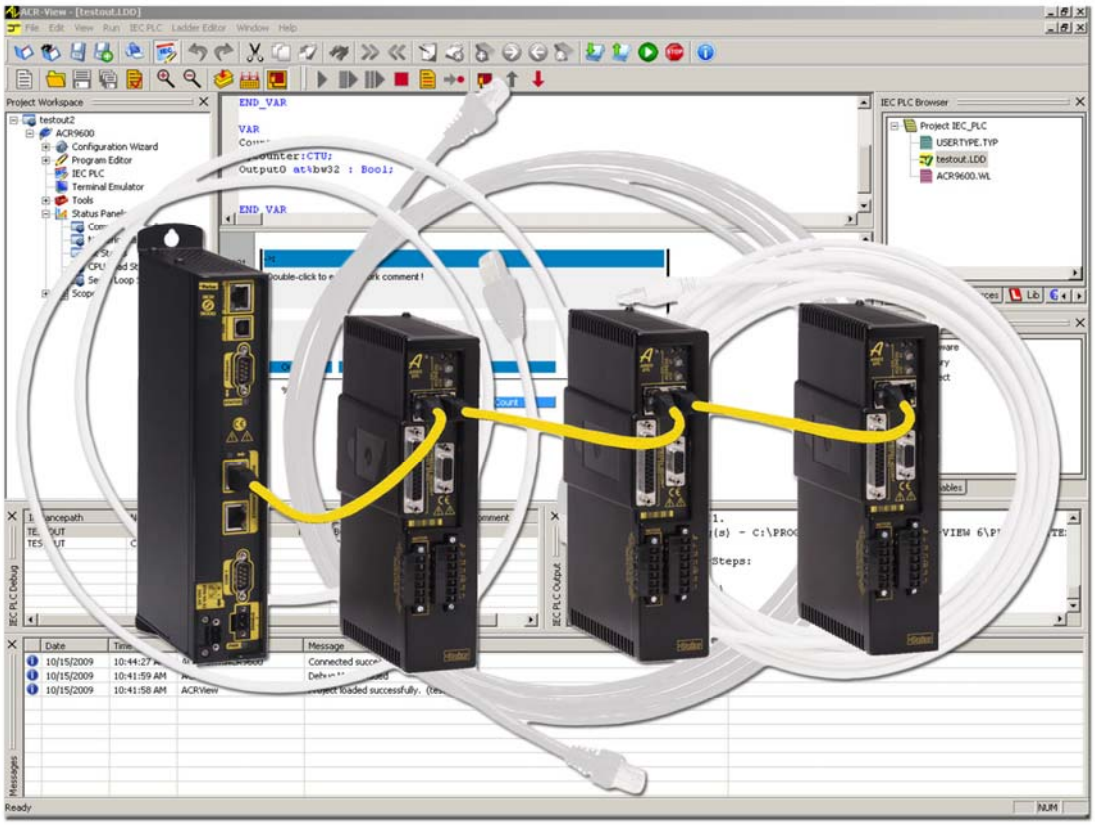
ACR Motion Controllers

88-030081-01A

IEC61131 Reference

Effective: October 2010



ENGINEERING YOUR SUCCESS.

Important User Information

It is important that motion control equipment is installed and operated in such a way that all applicable safety requirements are met. It is your responsibility as an installer to ensure that you identify the relevant safety standards and comply with them; failure to do so may result in damage to equipment and personal injury. In particular, you should study the contents of this user guide carefully before installing or operating the equipment.

The installation, setup, test, and maintenance procedures given in this guide should only be carried out by competent personnel trained in the installation of electronic equipment. Such personnel should be aware of the potential electrical and mechanical hazards associated with mains-powered motion control equipment—please see the safety warnings below. The individual or group having overall responsibility for this equipment must ensure that operators are adequately trained.

Under no circumstances will the suppliers of the equipment be liable for any incidental, consequential or special damages of any kind whatsoever, including but not limited to lost profits arising from or in any way connected with the use of the equipment or this guide.

Warning — High-performance motion control equipment is capable of producing rapid movement and very high forces. Unexpected motion may occur especially during the development of controller programs. KEEP WELL CLEAR of any machinery driven by stepper or servo motors. Never touch any part of the equipment while it is in operation.

This product is sold as a motion control component to be installed in a complete system using good engineering practice. Care must be taken to ensure that the product is installed and used in a safe manner according to local safety laws and regulations. In particular, the product must be positioned such that no part is accessible while power may be applied.

This and other information from Parker Hannifin Corporation, its subsidiaries, and authorized distributors provides product or system options for further investigation by users having technical expertise. Before you select or use any product or system, it is important that you analyze all aspects of your application and review the information concerning the product in the current product catalog. The user, through its own analysis and testing, is solely responsible for making the final selection of the system and components and assuring that all performance, safety, and warning requirements of the application are met.

If the equipment is used in any manner that does not conform to the instructions given in this user guide, then the protection provided by the equipment may be impaired.

Since Parker Hannifin constantly strives to improve all of its products, we reserve the right to modify equipment and user guides without prior notice. No part of this user guide may be reproduced in any form without the prior consent of Parker Hannifin. Since Parker Hannifin constantly strives to improve all of its products, we reserve the

right to modify equipment and user guides without prior notice. No part of this user guide may be reproduced in any form without the prior consent of Parker Hannifin.

Warning — ACR Series products are used to control electrical and mechanical components of motion control systems. You should test your motion system for safety under all potential conditions. Failure to do so can result in damage to equipment and/or serious injury to personnel.

ACR series products and the information in this guide are the proprietary property of Parker Hannifin Corporation or its licensers, and may not be copied, disclosed, or used for any purpose not expressly authorized by the owner thereof.

Since Parker Hannifin constantly strives to improve all of its products, we reserve the right to change this guide, and software and hardware mentioned therein, at any time without notice.

In no event will the provider of the equipment be liable for any incidental, consequential, or special damages of any kind or nature whatsoever, including but not limited to lost profits arising from or in any way connected with the use of the equipment or this guide.

Technical Assistance

Contact your local automation technology center (ATC) or distributor.

North America and Asia
Parker Hannifin
5500 Business Park Drive
Rohnert Park, CA 94928
Telephone: (800) 358-9070 or (707) 584-7558
Fax: (707) 584-3793
Email: emn_support@parker.com
Internet: http://www.parkermotion.com

Germany, Austria, Switzerland
Parker Hannifin
Postfach: 77607-1720
Robert-Bosch-Str. 22
D-77656 Offenburg
Telephone: +49 (0) 781 509-0
Fax: +49 (0) 781 509-176
Email: sales.hauser@parker.com
Internet: http://www.parker-emd.com

Europe (non-German speaking)
Parker Hannifin plc
Electromechanical Automation, Europe
Arena Business Centre
Holy Rood Close
Poole
Dorset, UK
BH17 7BA
Telephone: +44 (0) 1202 606300
Fax: +44 (0) 1202 606301
Email: support.digiplan@parker.com
Internet: http://www.parker-emd.com

Italy
Parker Hannifin
20092 Cinisello Balsamo
Milan, Italy via Gounod, 1
Telephone: +39 02 6601 2478
Fax: +39 02 6601 2808
Email: sales.sbc@parker.com
Internet: http://www.parker-emd.com

Automation

emn_support@parker.com

Table of Contents

Introduction

The ACR9600, ACR9630, and ACR9640 Programmable Automation Controllers (PAC) combine the proven, powerful motion control feature set of the ACR90x0 series with the industry standard PLC programming languages of IEC61131-3. This user guide details the features and commands available for programming the ACR96x0 controller using IEC61131-3.

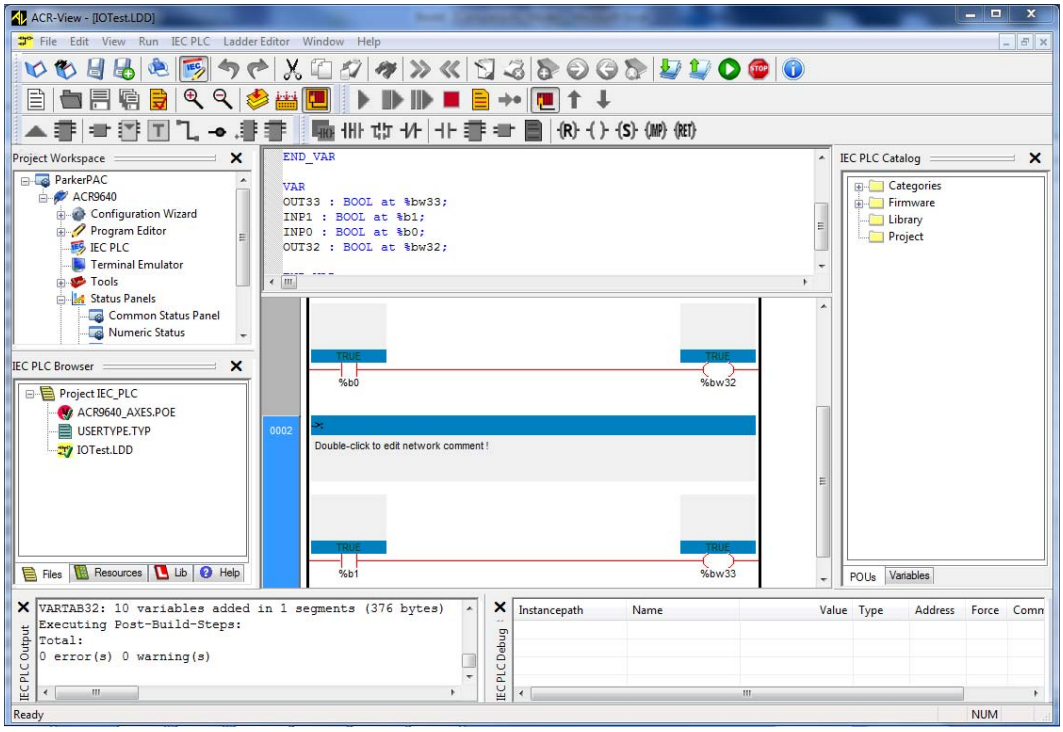
NOTE: This manual uses the nomenclature ACR96x0 to indicate the group of controllers which includes the ACR9600, ACR9630, and ACR9640.

ACR-View IEC PLC Tools

ACR-View

The project is shown in the Project-Browser on the left side. The editor-pane is located in the center. Most editors will use split screen technology to edit declarations in the upper pane and instructions in the lower pane. While declarations look the same for all programming languages, instructions vary widely. ACR-View can host many files at the same time.

Diagnostic messages will be shown in the output window at the bottom.



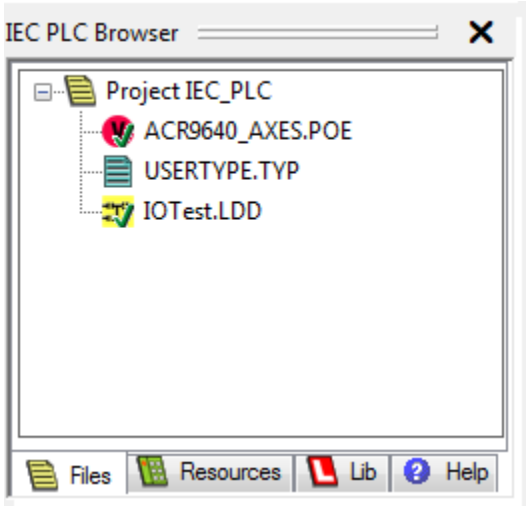
Output Window

The output window is located at the bottom of ACR-View and is used to display diagnostic messages.

Browser

Browser Introduction

The Project-Browser is the PLC File Manager of ACR-View. Using the Browser, you will organize your work into files and programs. From the Browser, you will create and edit files, compile, download and monitor your application:

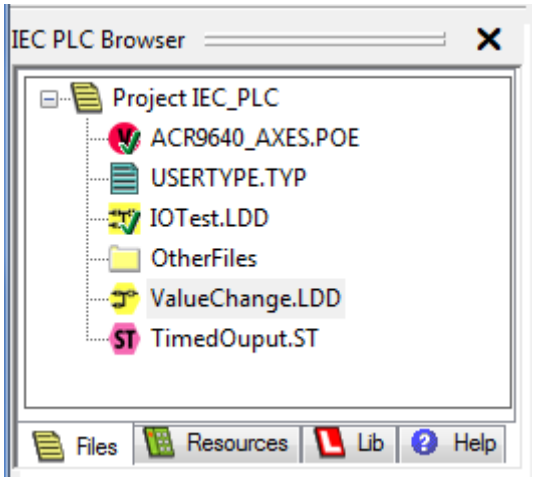


The Browser user interface consists of four different windows (panes):

1. The File-Pane
2. The Resource-Pane
3. The Library-Pane
4. The Help-Pane

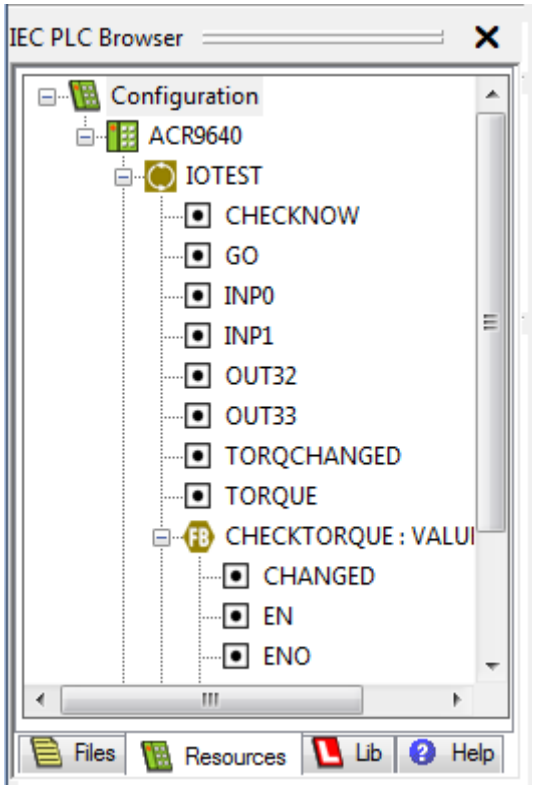
Browser Overview

The File-Pane



The File-Pane contains a directory-tree with all your source files, collected under the current project. These are the files that you write yourself, with one of the editors of ACR-View, or with different applications. All directories and files under the current project-path are shown.

Resource-Pane



The Resource-Pane shows your controllers, the tasks running in these controllers, the instances of functions and function blocks available within these, and all variables defined in these.

In the instance tree, there are only "links" to files and objects defined in the File-Pane: Tasks are referencing POUs of type PROGRAM, global variables are referencing global declaration files etc.

The Library-Pane

The Library-Pane (Lib) contains a tree with all installed libraries of the project. You can install new libraries with **IEC PLC** > **Library** > **Install New...**

You can use a library in a project by selecting it, right-clicking and choosing **Use in current project**. The libraries that are currently used in the project are shown with a red symbol.

The Help-Pane

The Help-Pane contains help-topics.

Files

Creating New Files

Create new files within ACR-View by selecting ***File IEC PLC*** > ***New*** to see the options:

POU for programs, function blocks and functions; the basic code blocks defined by IEC61131-3.

Declarations for creating resource global, direct global, and type declaration files.

Other for folders and watchlists.

File Operations

With the **File IEC PLC** menu you are able to:

- Move a file to another directory
- Copy a file
- Rename a file
- Import a file from another project/location
- Export a file to another project/location

Note: The action belongs to the file selected in the browser.

Resources and Tasks

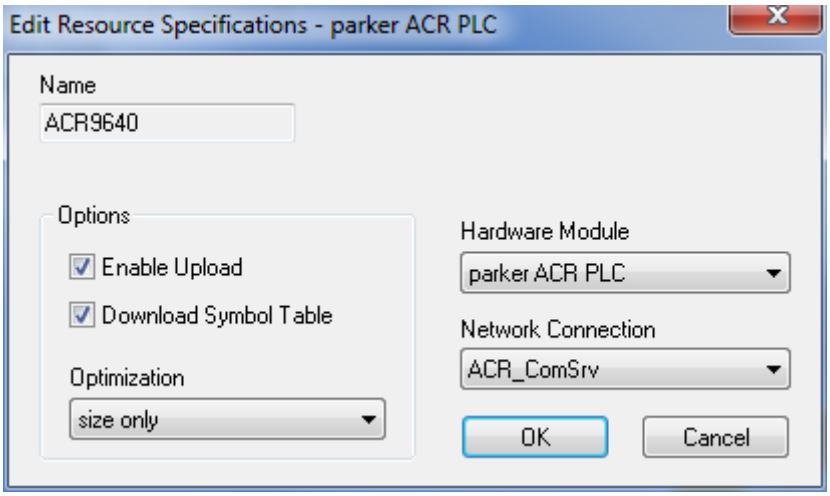
Resources Introduction

In general, a resource is equivalent to a PLC or a micro controller. A resource definition consists of a **name** for identification, the **hardware description**, i.e. Information about the properties of your PLC which will be used by ACR-View, and a **connection name**, i.e. Information about the kind of communication between ACR-View and the control system.

A resource maintains a list of tasks which are to be run on the control system.

Edit Resource

To edit the controller/resource, right-click on it from the Resource pane and choose "Properties" in the context menu. A dialog box opens in which you can change the hardware module and network connection properties, and enable or disable certain options:



Check "Enable Upload" to pack the sources of your application onto the target. This is helpful if at the end of debugging you want to save the project on the controller for future use.

Add Task

In general, a task is equivalent to a program plus the information about how the program can be executed. The definition of a task consists of the name, the Information about the execution of the task and a POU of type PROGRAM which should be executed in this task.

To add a task, mark the program you want to create the task of, and choose ***IEC PLC*** > ***Link to resource***.

After adding of the task, you can double-click it in the Resource-Pane to change the task specifications.

Note that the task name depends on the program name, and can't be changed. To complete the task definition, you must specify the information, how the task can be executed: Cyclic , Timer controlled , Interrupt controlled. Task type, priority and time control the execution of this task and in co-operation with other tasks. To do this, right-click on the task and choose "Properties."

Compiler

Build Active Resource

Build only those parts of your resource that have changed since last build due to modifications. Invoked by ***IEC PLC*** > ***Build active resource***.

ACR-View will automatically build anything as necessary when going online, but it is good practice to recompile from time to time when programming to detect errors as early as possible.

Rebuild Active Resource

To rebuild all tasks of your active resource choose ***IEC PLC > Rebuild active resource*** from the menu. This will completely recompile all parts of the active resource.

Rebuild All Resources

Like "Rebuild active resource" but will rebuild all—active and inactive—resources.

Online

Going Online

To get into online mode, choose ***IEC PLC*** > ***Online*** or press the "go-online" button in the toolbar to go online with the active resource.

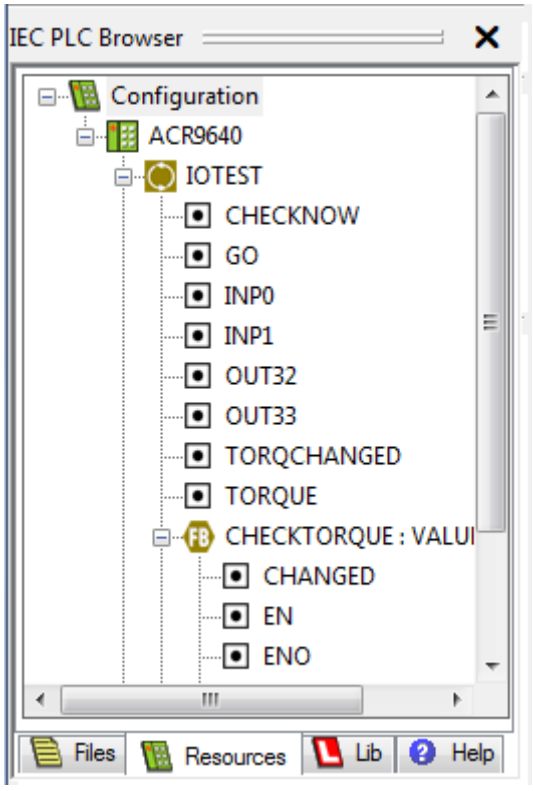
Repeat this to go offline again.

Download

ACR-View will automatically prompt whenever a download seems necessary.

Watching Variables

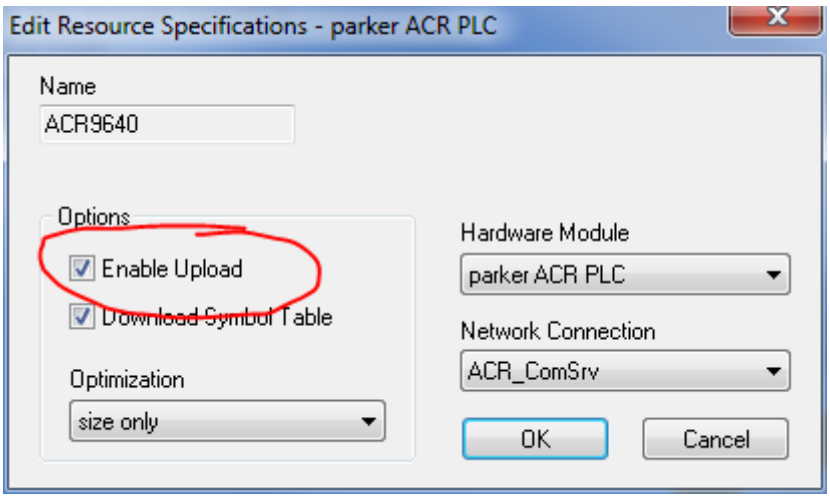
To add variables to the watch list of IEC PLC Debug window, open the resource tree of your application and double-click any of the variables:



Upload

ACR-View supports uploading of projects from the controller to a PC. Therefore, it is not necessary to have the source code of the project when updating the PLC, because the project can be uploaded.

To enable this feature, the "enable upload" box has to be checked in the resource properties before compiling and downloading a resource to the PLC as shown in the figure below:



For uploading the project, make sure that the resource properties are set as described above. Then go to ***IEC PLC*** > ***Upload IEC Project***.

Erase

This is only available in online mode. To remove the entire program from the PLC, select ***IEC PLC*** > ***Erase*** from the menu.

Other Browser Features

Resource Global Variables

In ACR-View, there are two kinds of global resource variables:

Global variables: these are variables without hardware-addresses, for example, for intermediate results.

Direct global variables: these are variables with direct hardware-addresses together with the IO-declarations. These represent the interface to the hardware.

To create a new file with resource global variables, select ***File*** > ***IEC PLC*** > ***New*** > ***Declarations*** > ***Global*** or ***File*** > ***IEC PLC*** > ***New*** > ***Declarations*** > ***Direct Global*** .

Type Definitions

By default, there is a file to hold user defined data types (usertype.typ) with each ACR-View project. To have your own data types, edit this file or create respective files of your own.

Add Files

ACR-View allows the addition of any kind of file to ACR-View projects. Use ***File*** > ***File*** > ***Import...*** and select the file of your choice. Beside files you have written with the editors of ACR-View (LD, ST, CFC), it is possible to import type definition and type declaration files. Furthermore, it is possible to register files in one project, even if they were created by other programs, for example by: Microsoft Word, Microsoft Excel, Microsoft Project, AutoCAD.

Select the desired file type in the popup menu and open the corresponding directory. Select the file you want to copy. This file will be copied to the current directory of the browser and can be edited by a double-click.

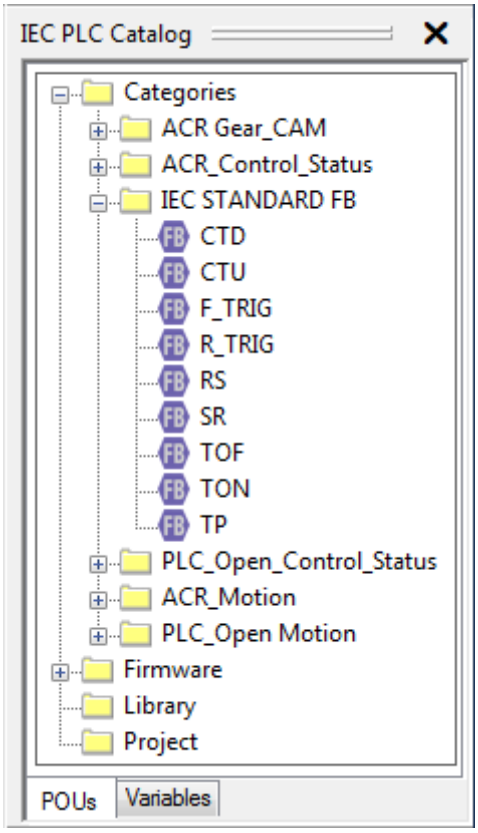
Catalog

Catalog

The Catalog is a tool to insert function blocks to your programs. The Catalog is visible below the project browser. If it is not there, go to ***View > IEC PLC > Catalog***.

With the catalog, you can insert function blocks to your programs by using drag'n drop.

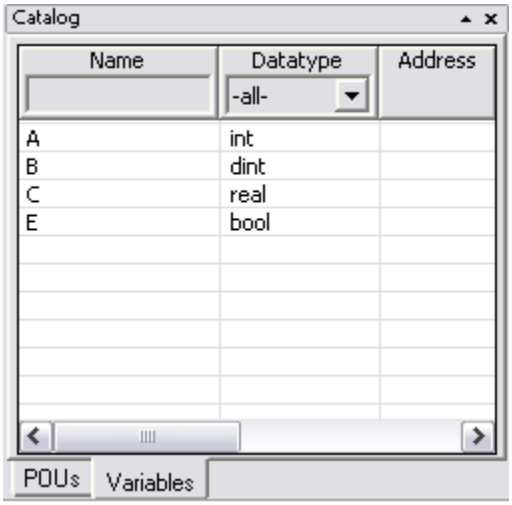
A double-click on an entry within the table opens the help on the function block.



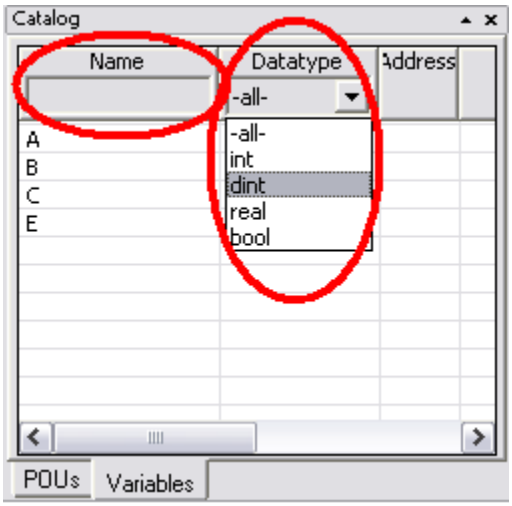
Using the Catalog, you do not have to write the names or go through the menus to insert a function block

Variable Catalog

The Variable Catalog is part of the Catalog. All global variables are shown in the Variable Catalog. You can see their names, datatypes, addresses, comments (if available) and their scopes. At the moment the used flag is only supported by the CFC-Editor.



The Variable Catalog enables you to insert global variables to your program by drag'n drop and also to filter global variables. You can filter names, datatypes and also scopes, to see which variables are available.



Just insert the name and you will see all variables that fit to your input. You can also use asterisks (for example, write "*A*" to the name field and you will get all variables which have an "A" in their names) and also use a combined filtering: First enter a name and then change the datatype.

When you create new global variables, they will not automatically be shown after saving the global variables file. Use a right-click into the variable grid and select refresh to update the Variable_Catalog.

Declaration Editor

Declaration Editor Introduction

IEC61131-3 requires all data objects to be declared as variables. A set of different declaration sections is available to define variables on different scopes. IEC61131-3 comes with a set of predefined data-types, called elementary data types. And, there are some means to define user-defined, so called derived data types, using structures, arrays and enumerations.

With most variables, storage is assigned by the compiler, without any programmer activity. For inputs, outputs, markers and potentially more types of variables, the programmer may specify a memory location, using directly represented variables.

Declarations are entered in text-form just as defined by IEC61131-3.

Declaration Sections

Variables are declared in different sections called declaration blocks. A declaration block starts with a keyword and ends with END_VAR (for example, VAR_GLOBAL ... END_VAR).

VAR_INPUT: If a variable block should only be read inside a POU, you must declare this variable as input-variable. It thereby is not allowed to modify this variable in this POU. An input-variable can be used for the parameter transfer in a function or function block.

VAR_IN_OUT: An input-/ output-variable is accessed under the same name by a function block. The variable gets a reference (pointer) to the transferred variable and its memory location during the parameter transfer by the block-call. Because a write-operation has a direct effect to the content of an In_Out-variable, it isn't allowed to use a write-protected type for the transferred variable as INPUT-variables or variables with attribute CONSTANT.

VAR_OUTPUT: The Output-variables are declared in the function block that uses them for the return of values. The calling POU can access them.

VAR_GLOBAL: A variable should be declared as global variable in the POU ´program´ if this variable should be valid in this POU and in the function blocks called by this POU. This variable must be declared as external variable (VAR_EXTERNAL) in all function blocks which intend to use this variable.

VAR_EXTERNAL: If a declared global variable will be used inside a function block, this variable must be declared as external variable inside this function block.

VAR: A local variable is only valid inside the POU in which it was declared. The declaration of local variables can be supplemented by the attributes ´RETAIN´ or ´CONSTANT´, or by an address.

TYPE: The keyword ´TYPE´ is used for declaration of user defined (derived) data types with local scope in the POU-types ´program´ and ´function block´, or with global scope in the type definitions.

According to the POU-type only certain variable-sections can be used:

- A POU of type Program may use Type, Local, Global and External
- A POU of type Function block may contain Type, Input, Output, In_Out, Local and External
- A POU of type Function may use Type, Input and Local.

CONSTANT may be used as a modifier to the keyword (for example, VAR_GLOBAL CONSTANT) to declare all variables declared in this section as not to be modified by the application. The compiler will issue a warning if such a variable is used in a context where it will or could be modified.

RETAIN may be used as a modifier for the keyword (for example, VAR RETAIN) to declare all variables in this section as retentive; i.e., these variables will not be re-initialized on hot- or warm-start. The system's retentive memory keeps variable values during power failures.

Structure of a Declaration Line

A declaration line has the following form, where optional parts are set in [square] brackets, and expressions are set between <sharp> brackets:

<variable name> [AT <Address>]: <Type> [:= <Initial value>]; [(* <Comment> *)]

First the variable name is given, followed by a colon. Behind the colon is the type, and eventually the hardware address introduced by the attribute ´AT´. Should the variable have a definite value on start, this value will be given after a ´:=´. A line ends always with a semicolon (;). The line can be commented, and comments are set between (* and *).

Example

```
Expvariable1 AT %B0: BOOL;  (* variable of type BOOL at the address %BIT0 *)

Expvariable2 : BOOL := TRUE;  (* variable of type BOOL with the start value TRUE *)
```

Variable with no initial value: `InterMedSum : INT;`

Variable with initial value: `Pieces : INT := 5;`

Directly represented variable with name and with no initial value: `Valve AT %BW32 : BOOL;`

Example function block: `Counter1 : CTU;`

Note: 1) Initial Values can only be given as literals. It is not possible to use other variables to initialize variables during declaration.

2) The significant length of a variable name is 64.

Elementary Data Types

Keyword	Name	Range	Size in Bits
BOOL	Boolean	0 (FALSE), 1 (TRUE)	1 or 8
SINT	Short Integer	-128 to +127	8
USINT	Unsigned Short Integer	0 to 255	8
INT	Integer	-32 768 to +32 767	16
DINT	Double Integer	-2.147.483.648 to +2.147.483.647	32
UINT	Unsigned Integer	0 to 65 535	16
UDINT	Unsigned Double Integer	0 to 4.294.967.295	32
REAL	Real number	+/-3,4E+/-38	32
TIME	Time duration	00:00:00:000 to 23:59:59.999	32
STRING	Character String		length of string plus 2 bytes

Keyword	Name	Range	Size in Bits
WSTRNG	2-byte-character String		length of wstring plus 2 bytes
BYTE	Sequence of 8 bits		8
WORD	Sequence of 16 bits		16
DWORD	Sequence of 32 bits		32

Directly Represented Variables

Directly represented variables are those variables that are mapped to a certain input, output or memory address specified by the programmer. The keyword AT is used to declare this, and the address is specified in a string starting with a percent sign (%).

Direct variables supported in ACR controllers:

- AT%Bnnn A read-only flag, where nnn is the bit number
- AT%BWnnn A read/write flag, where nnn is the bit number
- AT%Pnnn A read-only parameter, where nnn is the parameter number
- AT%PWnnn A read/write parameter, where nnn is the parameter number

Direct variables must be designated BW or PW (read/write) in order for them to be written back out to the controller at the end of a PLC scan.

> Note: Directly represented variables may only be defined in POUs of type "program."

ACR-View does not support the mapping to a physical PLC address (using AT%) for variables of types ARRAY, STRUCT and STRING.

Derived Datatypes

Derived data types are defined by the manufacturer of your controller, or by yourself. These new data types are defined using keywords TYPE ... END_TYPE based on the elementary data types. After definition, they may be used just like predefined or elementary data types.

Example: Derived Data Types
In the following sample code, a new data type is defined to represent a "Pressure" value

```
TYPE
    Pressure : INT;
END_TYPE

VAR
    PreValvePressure: Pressure;
END_VAR
```

It is possible to combine different datatypes in a derived datatype. Arrays and structs can be integrated as well. The following example defines a struct Athe struct itself consists of another struct called B and an integer array of size 5. Three new datatypes are derived within B: Stationname as straing and Value1, Value2 as reals.

```
TYPE
  A :
    STRUCT
      B :
        STRUCT
          Stationname : STRING
          Value1 : REAL
          Value2 : REAL
        END STRUCT
      Arr_5_INT:ARRAY [1..5] OF INT;
    END_STRUCT
END_TYPE
VAR
    Data1: A;
END_VAR
```

Declaration of Array Datatypes

Arrays contain multiple elements of the same data type. The keyword ARRAY is used to define an array. Each element of an array can be an elementary variable.

Example: Array Data Type

Type Arr1 will hold five elements of type INT

```
PROGRAM feld
TYPE
 Arr_5_INT:ARRAY [1..5] OF INT;
END_TYPE
VAR
 Arr1 : Arr_5_INT;
END_VAR
 .
END_PROGRAM
```

Declaration of Structured Datatypes

A structure holds multiple elements of same or different data types, elementary. Key word STRUCT is used to define a structure. The individual elements of a structure are called members of that structure, and are accessed by writing the structure, followed by a dot and the name of the member.

Example: Structured Data Type

```
PROGRAM struktur
TYPE
  RobotArm :
    STRUCT
       Angle_1 : REAL;
       Angle_2 : REAL;
       Grip: BOOL;
       Length: INT;
END_STRUCT;
END_TYPE
VAR
  Robot1 : RobotArm;
  Robot2: RobotArm;
```

```
 END_VAR
  LD Robot1.Grip
.
  .
END_PROGRAM
```

Declaration of Enumeration Datatypes

A variable of an enumerated data type can take any one of a fixed list of values. The list of legal values is listed in the declaration of the enumeration data type, separated by commas. An initial value may be given after the closing ")"; if no initial value is given, the first value will be the de fault.

Example: Enumeration Data Type

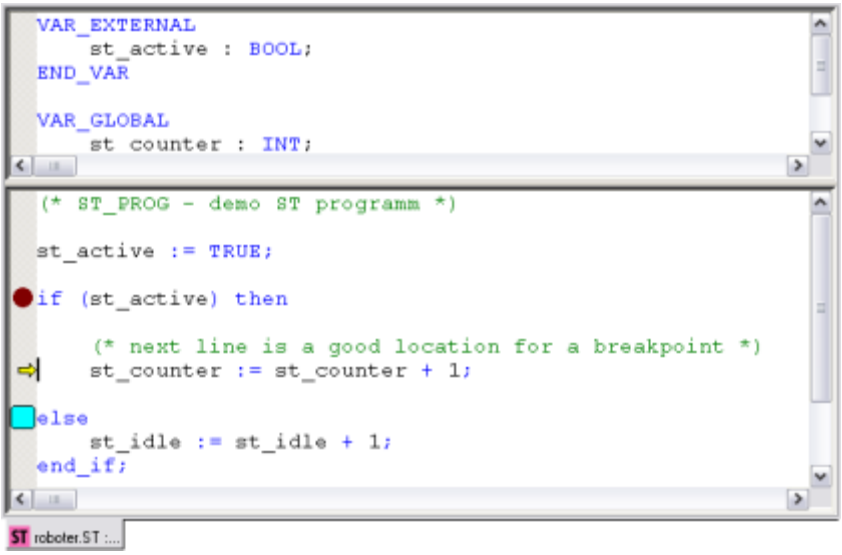
Data type TrafficLight can be "red", "yellow" or "green." "Yellow" shall be the default.

```
TYPE TrafficLight:
    (red,
    yellow,
    green):= yellow;
END_TYPE
VAR
    MainRoad : TrafficLight;
    CrossRoad : TrafficLight;
    StopCar: BOOL;
END_VAR
```

ST Editor

ST Editor Introduction

The ST-Editor is hosted in ACR-View. In the upper part of the ST-Editor, enter the declarations of the POU. In the lower pane, enter ST instructions:



The ST Editor supports bookmarks (for marking lines of interest while editing a file) and Breakpoints.

Instructions in ST

Code written in ST is a sequence of ST-instructions. ST-instructions are terminated with a semi colon.

Linefeeds are not significant, i.e. more than one instruction can be on one line, and one instruction can use one or more line.

For a list of all instructions supported in ST, please see the reference section, Structured Text Keywords.

Expressions in ST

Operands known in ST are:

- Literal variables, for example: 14, "abc", t#3d_5h
- Variables, for example: Var1, Var[2,3]
- Function Call, for example: Max(a,b)

While operators are parts of ST-language, expressions are constructions which must be constructed by aid of ST-elements. Operators need operands to build expressions.

Element	Symbol
Parentheses	( )
function call	
Exponentiation	**
Negation	-
Complement	NOT
Multiplication	*
Division	/
Modulo	MOD
Addition	+
Subtraction	-
Comparison	<, >, <=, >=
Equality	=
Inequality	<>
boolean AND	&, AND
boolean exclusive OR	XOR
boolean OR	OR

Comments in ST

Like all modern programming languages, ST supports comments. A comment is any text included between ´(*´ and ´*)´, for example:

```
(* Comments are helpful *)
```

The compiler will ignore comments when generating executable code, so your program will not accelerate in any way if you omit comments. Comments may span multiple lines, for example:

```
(* This comment
   is long and
   needs more than one
   line
 *)
```

ST Editor Online

To debug and monitor code written in ST, use the ST Editor in monitor mode.

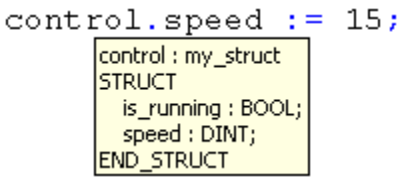
There are three main ways to debug and monitor ST code:

- Use Breakpoints to stop execution, single-step through your code. Use this to understand, follow and find problems in the logic flow of the application.
- Move the mouse cursor over a variable and see a tiny "toolbox" appear, displaying the variable's name, type and value. The value is permanently updated. Use this to quickly examine the current value of different variables within a region of your code, with or without stopping execution, at a breakpoint or while single-stepping.
- Use the watch list in the IEC PLC Debug window to monitor a set of variables, which may be from any part of your applications. Use this to keep an eye on a set of variables while examining different parts of your application's code.

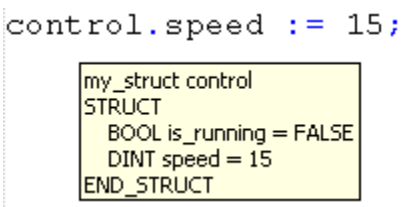
ACR-View supports online edit. For further information see the section Online Edit.

Tooltips for Structs and Elements of Structs

It is possible to watch the whole structure information in any depth in the ST Editor tooltips.



If the Editor is in Edit mode, the struct and its first level members will be shown with datatype information. In Online mode, the values will be shown behind the resolvable members.



Ladder Diagram Editor

Ladder Logic Introduction

The basic principle of Ladder Logic is currency flow through networks. Generally, Ladder Logic is restricted to processing boolean signals (1=True, 0=False).

A Network is restricted by so called margin connectors to the left and to the right within the Ladder Editor. The left margin connector has the logical value 1 (current). There are connections that conduct currency to elements (variables) that conduct currency to the right hand side or isolate depending on their logical state. The result of the procedure depends on the arrangement of elements and the way they are connected (AND = serial; OR = parallel).

Networks consist of the following graphical objects:

- Connections (horizontal or vertical lines, and soldered points).
- Contacts, Coils, Control Relays
- Function blocks and Functions
- Jumps (Graphical elements for control flow).

Network

The instruction section of the Ladder Diagram Editor is subdivided into so called networks, which help structuring the graphic.

A network consists of: Network label, Network comment and Network graphic.

Network label: Each network that may be a jump target from within another network will automatically be assigned a preceding alphanumerical identifier or an unsigned decimal integer. By default, networks will be numbered. This numbering of all networks will be automatically updated whenever a new network is inserted. The numbering simplifies finding a certain network an corresponds to line numbers of textual programming languages.
Network comment: The Network Comment is represented as a square area in the ladder diagram. To enter a commentary text, double click on this square. The comment is always placed below the network label. Note that the first network additionally contains a ladder diagram comment above the network label and the network comment.
Network graphic: The network graphic consist of graphical objects, which may be graphical symbols or connections. Connections transport data between graphical symbols, which process the data at their inputs and transfer the processed data to their outputs. Note that the connections may also cross.

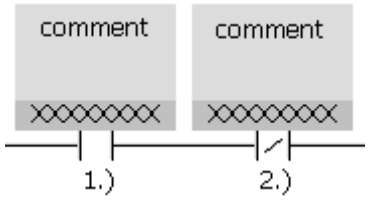
Operators

Within a ladder diagram, the term operator designates the graphical objects contact, coil and jump.

- Contacts: A contact associates the value of an incoming connection with the value of an assigned variable. The kind of association depends on the type of contact. The result value will be transferred to the connection on the right hand side. There are triggers and interruptors (The boolean value of the variable will not be changed).
- Coils: Coils serve to assign values to output variables of networks. A coil copies the state of the connector on its left hand side to its connector on its right hand side without any changes. Furthermore, the coil saves a function of the state or the transition of the left connector into a boolean variable.
- Jumps: Jumps manipulate the control flow of programs. They make it possible to directly invoke certain networks in a defined order. When encountering a jump operator, control flow continues at a different network. Thus, jumps are an exception from the basic principle that networks are always processed in a top down fashion.

Contacts

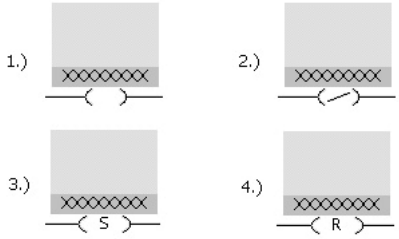
There are two contact symbols for boolean input variables:



- Left is the contact symbol for a variable that must have the value "1" to make the corresponding boolean connection true. If the variable is associated with a physical address, the state "1" corresponds to a released interruptor or a pressed trigger.
- Right is the contact symbol for a variable that must have the value "0" to make the corresponding boolean connection true. If the variable is associated with a physical address, the state "0" corresponds to a pressed interruptor or a released trigger.

Coils

The output variable is always situated to the right hand side of the network and is connected to the right currency rail.



- The result of the logical connection will directly be assigned to the output variable.
- The output variable will be assigned the negation of the result of the logical connection.
- The result of the logical connection will "permanently set" the output variable: If the result of the logical connection is "1", the output variable will be set to "1". If, however, the result of the logical connection is "0", this will have no implications.
- The result of the logical connection will "permanently reset" the output variable: If the result of the logical connection is "1", the output variable will be set to "0". If, however, the result of the logical connection is "0", this will have no implications.

Jumps

- Jump operations manipulate control flow. With jumps, networks may be executed only if certain conditions hold. Jumps may be conditioned by a binary combination result, or not conditioned," i.e., obligatory. The jump target must always be the beginning of a network, designated by its network label.
- Return jumps stop program execution within the current POU, and continue at the point where the POU was invoked from. Return jumps may be conditioned by a binary connection result, or unconditioned.

Control Relay

Control relays are contacts that are inserted in front of coils. Control relays may be used as breakpoints in manual execution, for example. There can always be one control relay before each coil only.

Insert-> Control Relay: Use this command to insert a control relay additional to the logical symbol.

Functionblocks and Functions

To insert Function Blocks or Functions to a network, click on a connection and use **Insert** > **Functionblock...** or **Insert** > **Function...** to insert it at this position. You can then choose the desired block or function from a list of available blocks/functions. Only predefined functions can be chosen.

A function block can only be added to a network if it satisfies the following criteria:

- The first input-parameter of the block has to be of type BOOL and has to have the name "EN". If this parameter is set to FALSE in a network,

the corresponding block won't be started or even get parameters passed.

- The first output-parameter of the block has to be of type BOOL and has to have the name "ENO". This parameter has to be set to TRUE if the block has worked correctly and without errors.

Ladder Editor Online

When you have the Ladder Editor in monitor mode, it will automatically start displaying live values of contacts, coils, function and function block inputs and outputs as far as possible.

If the online editor can't get a value of a variable from the runtime system, it will display "-!-".

Displaying values in the online editor of variable types, that use more than 4 bytes (strings, arrays, structs), is not supported by the current version of the Ladder Editor. To view them use the IEC PLC Debug.

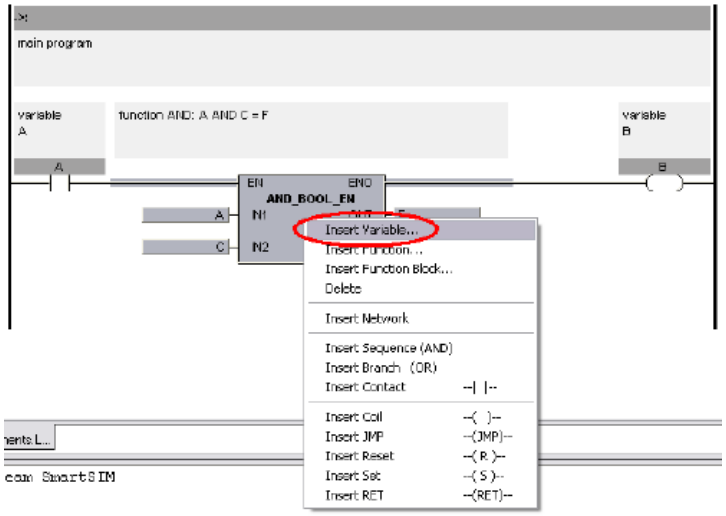
ACR-View supports online edit. For further information see the section Online Edit.

Check over Variable

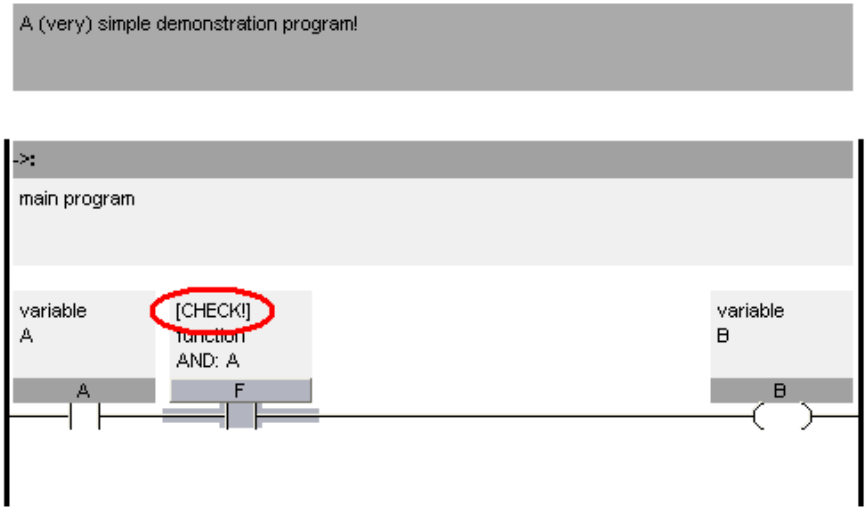
The Ladder Editor contains a comment check method, that marks comments if the semantic of a program has changed. To mark comments that might be wrong, ACR-View pre-writes „[CHECK!] to such comments. Then it's up to you to check if these comments are still correct.

The reason is that when using the ladder editor, it is possible to replace a function (block) by a contact with a variable or vice versa. This changes the semantic of the program and so the comments above the function (block) or variable might be wrong.

To illustrate this, look at the following figures. Choose a function that you want to be replaced by a contact with a variable. Select it with the right mouse button and choose Insert Variable from the context menu.



After replacing this function by a contact, the comment above the function is changed. Now, there is pre-written [CHECK!].



The main reason, therefore, is that the semantic of the program has changed, but the comment is still the same. This is a hint, to verify if this comment is still correct.

CFC Editor

CFC Editor Introduction

The Continuous Function Chart Editor is an engineering tool used to create automation programs graphically.

The main elements of a CFC chart are Blocks (firmware blocks, user defined blocks, compound blocks), that can be freely arranged on the chart, Margin Bars (left and right), which provide links to IEC61131 variables and virtual links within the chart, and connections, to connect one output (block or margin bar) to one or more inputs (block or margin bar).

Working with Blocks

To add blocks to your CFC chart, use Insert > Block for firmware or user-defined blocks, Insert > Textblock for text blocks, or Insert > CompoundBlock for compound blocks.

The mouse cursor will change, click the chart where you want to insert the new block.

To re-arrange blocks, select the blocks and drag-and-drop them to their new location.

When adding new blocks or moving existing blocks, the CFC Editor will make room by moving aside existing blocks as appropriate.

To remove blocks from your chart, select them and press DEL.

Click twice on a block give it an alias name.

Connections

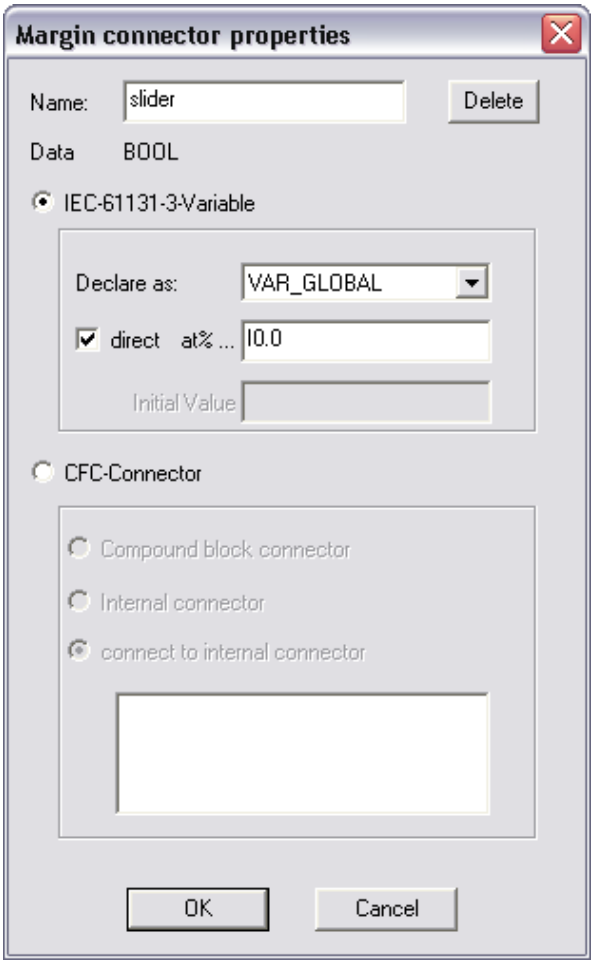
To connect two objects, first select the output object (output of a block, or item on the left margin bar), then select the input (input of a function block, or item on the right margin bar), then press **Insert** > **Connection.**

ACR-View also supports Multiple Connections

Margin Bars

Margin Bars connect the logic contained in the CFC chart to other parts of the same CFC chart, or to other parts of the application or the process to be controlled.

To configure any element of the margin bar, right-click it and select "Properties" from the context menu:



In Name, enter the name of the object. This should be a valid IEC61131-3 variable name.

If you want the CFC-Editor to declare a variable for this margin bar object, select IEC61131-Variable. Otherwise, if you select "CFC-Connector", the object is used only virtually, and all information is immediately propagated

to the connected outputs. This may be more economic in runtime and memory consumption, but it prevents online monitoring.

For IEC61131-3 variables, select the declaration section from the combo-box. The selection offered here depends on the type of block and the type of margin bar. For some kinds of variables, you may choose to select a physical address or an initial value.

For CFC-connectors, you can choose "compound block connector," i.e., a connection from within a compound block to the outside, "(connect to) internal connector", i.e., virtually connecting one entry on the right margin bar back to one on the left margin bar. "Internal connector" and "connect to internal connector" are similar, but the first is only available on a right margin bar (where internal connectors are defined), whereas the latter is available only at a left margin bar, where internal connectors may be used.

CFC Editor Online

When you have the CFC Editor in monitor mode, it will automatically start displaying live values of blocks, connections and margin bar entries as far as possible.

If the online editor can't get a value of a variable from the runtime system, it will display "-!-".

ACR-View supports "online edit. For further information see the section Online Edit.

Advanced CFC topics

Text Block

Use Insert > Textblock to insert a text block into your chart. A text block is only for documentation purposes and does not add anything to the code being executed.

Printing CFC Charts

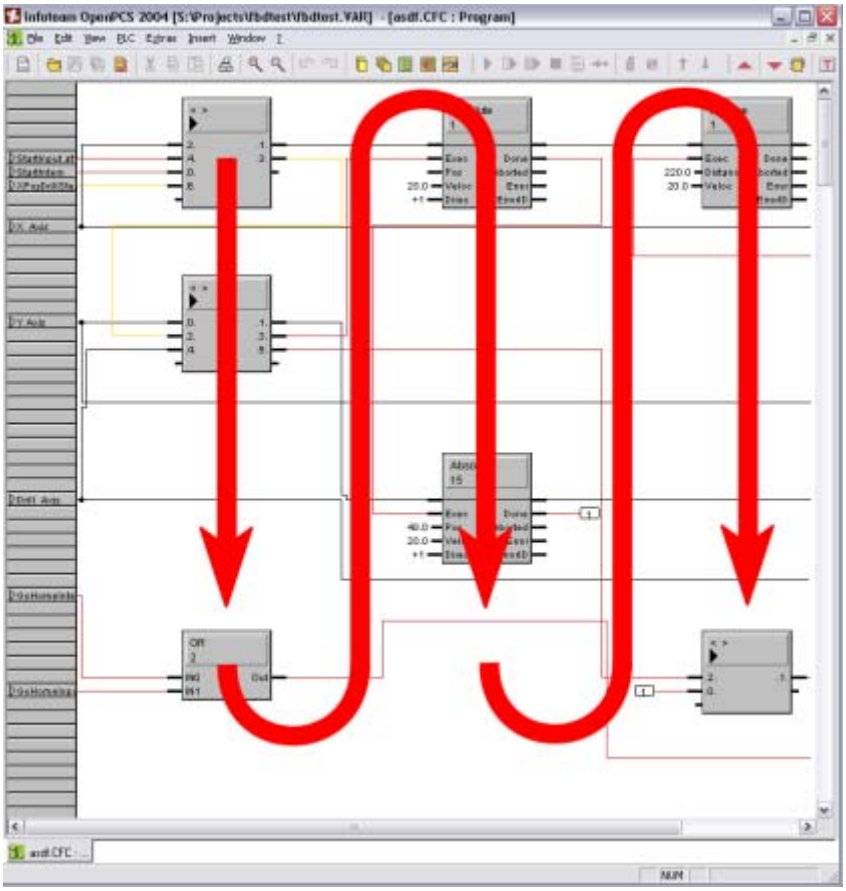
The CFC-Editor offers you several possibilities for printing. Use File > Print to print the current level of a chart, and File > Print All to print all levels of the loaded CFC chart.

Using Constants as Inputs

To use a constant value as the input to a block, select the input (or margin bar entry), right click it with the mouse, select "properties" and enter the constant value in the edit field "value" on sheet "default value."

Execution Order

The arrangement of the blocks on a chart is directly related to the sequence of execution: Blocks are executed first column first from top to bottom, then second column top to bottom, and so on. To modify execution sequence, rearrange the blocks as required.



Compound blocks will be executed as a whole at that moment in the execution order where the compound block is located. The contents of the compound block will be executed in itself following the same rules. This is very similar to subroutines in modern programming languages.

Multiple Connections

The CFC editor supports connections between one output and multiple inputs To create a multiple connection first create a connection between the desired output and one input. Now, mark the next input and click in the output. The connection, created in the first step and the output are now marked. Choose **Insert** > **Connection** to create the multiple connection between the output and the two inputs. You can now add more inputs the same way.

To remove an input from a multiple connection, mark the input and hit the delete-key. Only the connection between this input and the output will be removed.

Replacement of Blocks

The CFC editor supports the replacement of a firmware or user-defined block by a block of another type by selecting the block(s) and choosing Edit > Replace Block from the menu.

A dialog box analogue to the Insert > Block dialog will appear, allowing the user to select the desired new block type from a list of known firmware and user-defined blocks.

Additionally the user may check the option automatically replace all instances of the block type in current plan", which causes the replacement of all instances (even the non-marked ones) of the currently marked block's block type inside the entire CFC-plan.

After selection of a new block type, another dialog box is shown, allowing the user to map the connectors of the old and new block type for reconnection after replacement. The left column of the displayed table lists the connectors of the old block type together with the type and kind (VAR_INPUT/VAR_OUTPUT) of the connector (*1). The right-hand column displays a list of adequate connectors of the new block type.

The user can assign a corresponding connector for each connector of the old block type. Note that each connector of the new block may only assigned once.

If a connector shall or can not be reconnected, do not reconnect automatically" can be chosen.

After clicking OK the CFC editor replaces the block(s) by (a) block(s) of the new block type and rewires the connectors as specified in the assignment dialog.

(*1): VAR_IN_OUT connectors will show twice in the list of connectors: Once as VAR_INPUT& and once as VAR_OUTPUT&. The &" marker signals, that the connector actually represents an VAR_IN_OUT parameter.

Finding Errors in CFC

Double-click the error message in the output window to locate an error.

Block Specific Help

It is possible to get a block specific help. Right-click on the block, you want help for, and select the menu-item Show documentation." If ACR-View finds no reference, you will be prompted. If one reference is found, it will be displayed and if more than one reference you will be prompted to choose which one to display.

Extensible Inputs

The following CFC (and FBD) functions are extensible. This means we can add one ore more inputs as a copy of the first input:
AND, ANDN, OR, ORN, XOR, XORN, MUL, ADD, MUX, MIN, MAX, CONCAT
Appending an input is done by selecting one of those functions and calling (context) menu entry "Append Input." If you want to delete again an added input, select input and call (context) menu entry "Delete Input."

Functions with Negatable Inputs

For all of the following logical CFC (FBD) functions you can negate each boolean input:
AND, ANDN, OR, ORN, XOR, XORN, NOT

Negating an input is done via selecting the input and calling (context) menu entry "Negate Input." A negation circle is drawn at the connector.

The next call of (context) menu entry Negate Input" removes the negation.

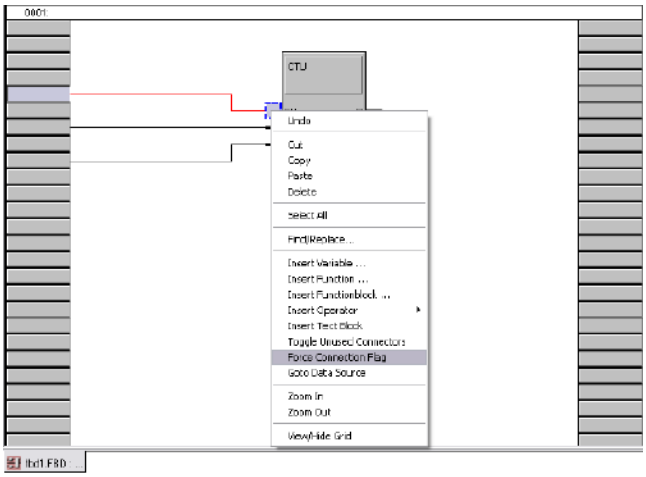
Syntax Check at CFC Connections

After inline editing values or IEC identifiers on all CFC connectors the user input is checked for correct syntax: If a constant value is entered that does not fit the data type of the connector a message like the following is shown, and the value is accepted in spite of the syntax error.

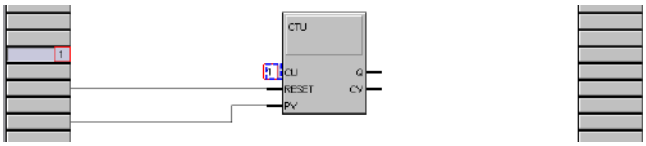
```
Syntax error: Invalid constant for data type xxx."
```

Connection Flag

To reduce the number of connection lines we can suppress single connections and force so called connection flags via (context) menu entry Toggle force connection flag":



Use connection flags for this single connection.



The suppression of connection lines is saved with plan and restored after reloading.

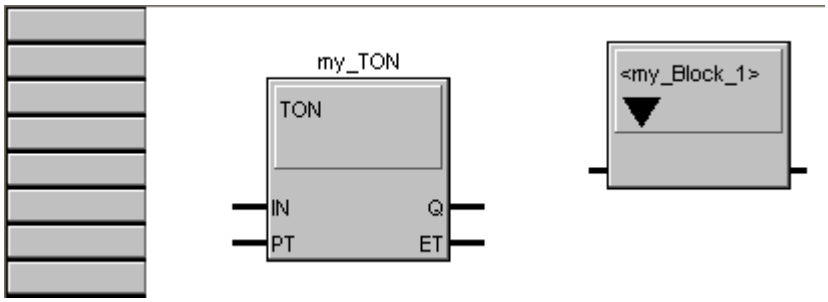
Copying Blocks with Inputs

If at least one block is selected, there is a new (context) menu entry active: Duplicate blocks. Calling it copies the selected block(s) into the internal plan clipboard and sets editor into duplicate mode - mouse cursor and caret style behave and look like they do in paste mode: Everywhere you click or press the space bar, the duplicate(s) of the block(s) is(are) inserted and all input connections are duplicated. Until you right-click the mouse,

press ESCAPE, or click into a "no-paste-allowed" area, the editor stays in duplicate mode so you can insert more duplicates.

Alias Names

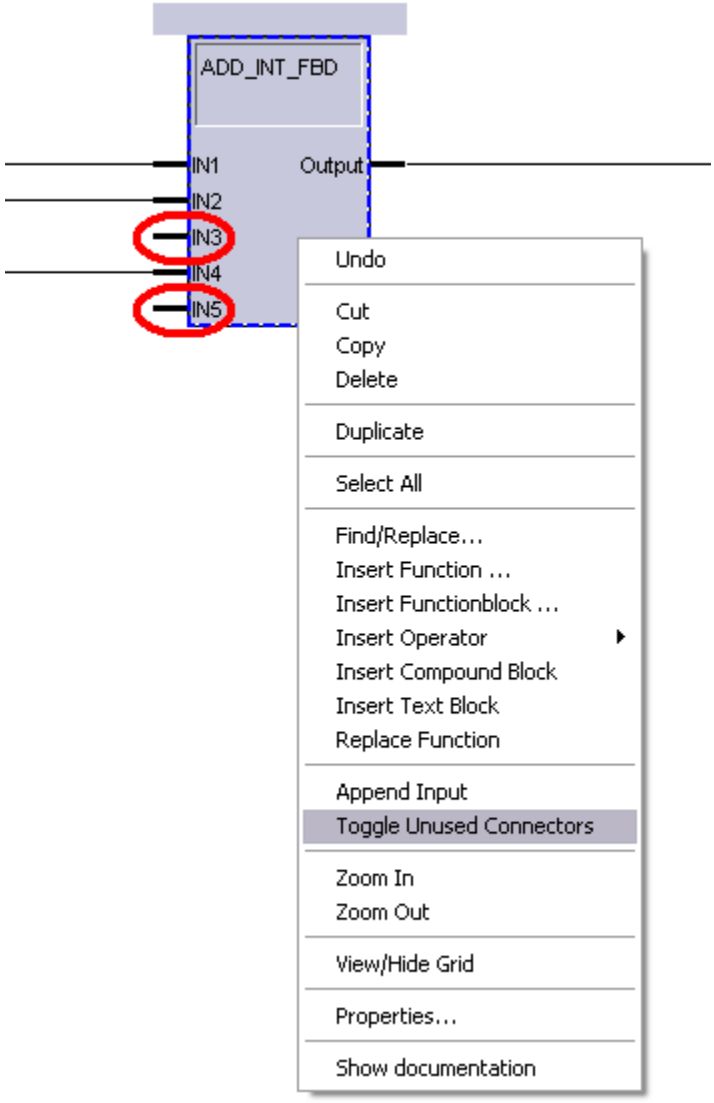
The user can enter alias names for blocks to mark and quick find special blocks. Alias names for functions and function blocks are drawn and inline editable above the block body. Alias names for compound blocks are drawn and inline editable within the block body.



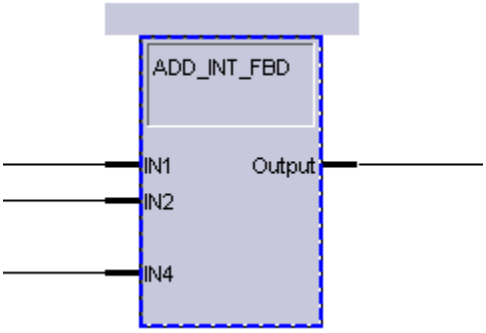
Exception: The Operators SET and RESET cannot have alias names because the boolean variable that is set/reset is located above the block body.

Masking of Unused Connectors

For more clarity there is a new (contex) menu entry "Toggle Unused Connectors." Calling it hide/shows **all** unused block connectors. Unused connectors are connectors without any connections and values.



Unused connectors are not shown.



If unused connectors are hidden, the following conditions result:

- Connectors cannot be found by searching.
- Mouse and keyboard cannot be used for navigation.
- They can be found by double clicking on a compiler/syntax error/warning.

Keyboard Handling for CFC and FBD Editor

Fundamentals for Keyboard Usage

For keyboard navigation, a small caret is displayed which shows the current input focus for the user.

The CFC/FBD editor can be used with mouse and keyboard simultaneouly. **The cursor will not follow the caret.** The form of the cursor will not automatically change due to the state of the caret. The state of the cursor will of course follow the position of the cursor and not the position of the caret

Caret and Selection

The current selection follows the caret. Exceptions or special cases are:

- If the caret is navigated to an empty grid cell, the selection is canceled (nothing is selected).
- To detach the caret position from the current selection for generating a connection, the caret must be navigated while <shift>-key is pressed. As the <shift>-key is released the selection is enlarged by the element at the current caret position (aquivalent to a left-click on the element in the caret). The current implementation takes care that only permitted states of selections can be made.
- Multiple selections with other elements can be made using <ctrl> while navigating. (Multiple selections consisting of isolated blocks are not allowed.)

Representation of the Caret

The caret is always visible, even if the element on which the caret is located is selected.

- In special cases the caret is represented in a different way.
- The caret is always visible even if the selection is done by mouse.
- The caret can not be switched off.
- The caret will not be printed.

Positioning of the Caret

The caret is positioned at the marked point by left or right mouse click. It follows in general the selection by mouse.

Caret Position by Selected Moves

It must be grantueed that (even in co-use of mouse and keyboard) there is always a valid caret position. The caret position is defined for the following actions which remove the element at a valid caret position:

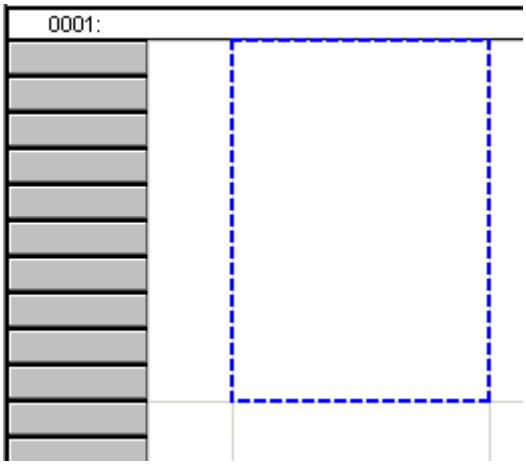
- *Selection by mouse:* The caret follows in general the selection by mouse and automatic functions
- *Removing/cutting a block:* Thereafter the caret will expand to the whole grid cell which was occupied by the removed/cutted block.
- *Removing/cutting a set of blocks:* Thereafter the caret will select the left upper grid cell which was occupied by the set of blocks.
- *Removing/Cutting the input of a block:* The caret will jump to the input that is above the removed/cutted input. If there isn't any, the caret will expand to the whole block.
- *Removing/cutting a network:* The caret will jump to the network above the removed/cutted network. If there isn't any, the caret will jump to next possible network below.
- *Removing/cutting a set of networks:* The caret will jump to the network that is above the uppest network. If there isn't any, it will jump to the first network below.
- *Decreasing the number of rows in a network:* The caret will jump to the grid row above, the grid column will be the same. The caret refers at first to the grid cell even if there is a block contained in it.
- *Caret position after „select all:* After the call of „select all, the caret jumps to left uppest grid cell in the map. The map is scrolled upwards for uncovering the caret. Internally the same method is called as by using the shortcut <ctrl>+<pos1>.

Automatic Positioning of the Caret

- After a file is loaded, the caret is placed at the upper left grid cell. The position of the caret is not saved with the map.
- After the entering of a compound block, the caret will be placed at the upper left grid cell.
- By using undo/redo, the caret follows the position which is provided by the operation. For this purpose, the caret position is saved before undo/redo and will be restored according to network number and position (row, column). If the network or the concerning cell doesn't exist anymore, the caret will jump to the next network/cell above.

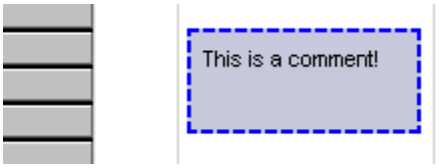
Below, the defaults for the positioning of the caret are listed, depending on the driven CFC/FBD element. How the navigate between these positions is described in a future chapter (Caret navigation).

Caret IN Empty Grid Cells



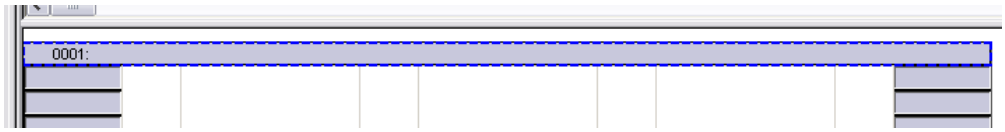
In empty grid cells, the caret takes the size and position of the whole cell.

Caret and Comments



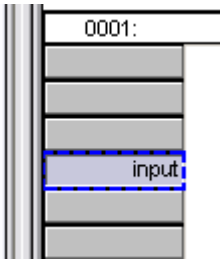
At grid cells with comments, the caret takes the position and size according to the selected comment.

Caret at the (FBD) Network Label



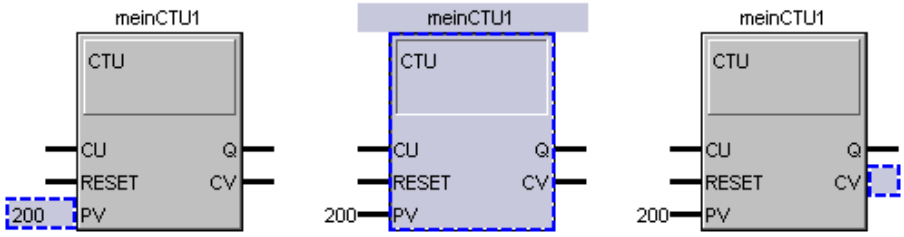
At the network label, the caret takes the position and the size according to the network title line (according to the measures of the selected network label).

Caret at a Margin Connector



At a margin connector, the caret takes the position and size according to the measures of the selected margin connector.

Caret in Grid Cells with Blocks



The caret surrounds either the block field or a connector. The size of the caret at a connector/block corresponds to the selection of a connector/block. The name of an entity will not be surrounded by the caret.

Caret Navigation

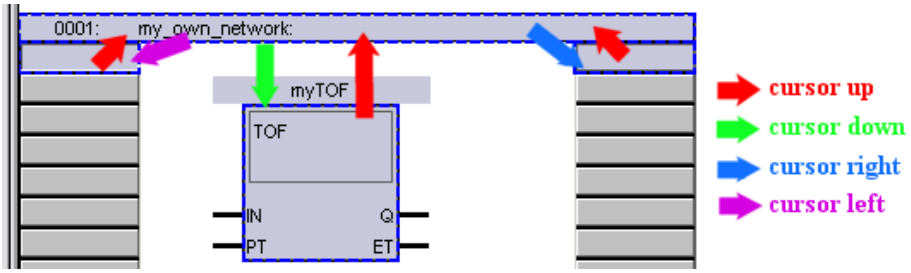
In the following is described how to navigate with the caret inside a CFC/FBD map.

Navigating at Margin

At margin, you can jump to the underlying margin element or the element above by using <UP> or <DOWN> arrow keys.

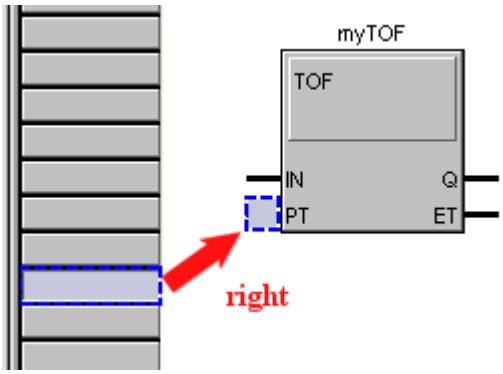
Navigating between (FBD) Networks and Network Labels

- If the caret is on the upper or lower margin connector, you can jump to the network label of the underlying network or network above by using <UP> or <DOWN> arrow keys (see picture below).
- If the caret is on a grid cell or element in the upper row of a network you can jump to the network label of the network above by using <UP>
- If the caret is on a grid cell or element in the lower row of a network, you can jump to the network label of the underlying network by using <DOWN>
- If the caret is on a network label, you can jump to the left lower grid cell (resp. grid element or connector) of the network above by using <UP>
- If the caret is on a network label, you can jump to the left upper gird cell (resp. grid element or connector) of the network belonging to the network label by using <DOWN>. With <RIGHT> or <LEFT> the caret jumps to the upper connecter of the left or right margin.

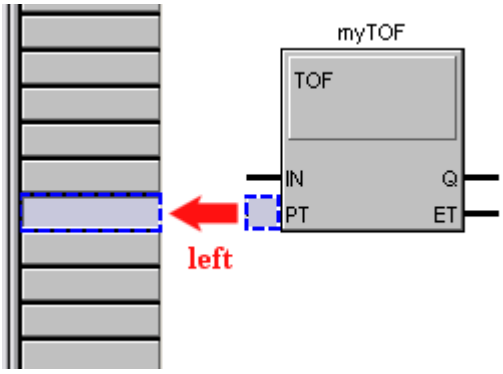


Changeover Margin to Block

By using <RIGHT> or <LEFT> when the caret is located at left or right margin, the caret jumps to the grid cell resp. element of the grid cell which is opposite to the margin connector. A margin connector at the level of a connection channel is always assigned to the grid cell above the connection channel. If the grid cell contains a block, the caret jumps to the closest connector in consideration of the starting position (margin connector).



If the caret is positioned on a grid cell or on a block connector besides the margin, it jumps to the closest margin connector.



Up and Down at Inputs and Outputs

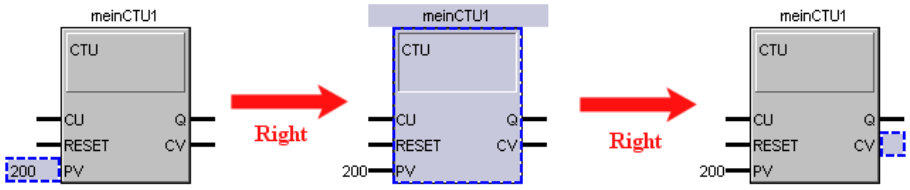
<UP> or <DOWN> navigates the caret to the input or output of a block. If the caret is located on the lowest input/output, you jump to the underlying grid cell or the label of the next network by using <DOWN>.

Left and Right at Inputs and Outputs

<LEFT> or <RIGHT> navigates the caret between input/output and the block field itself.

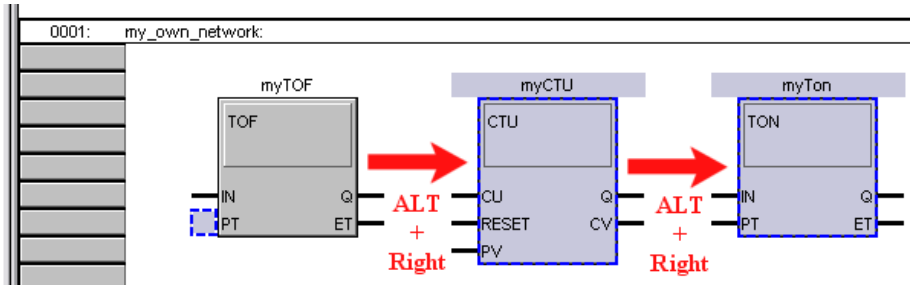
Observe the behavior of the caret by navigating from the inputs/outputs of a block to the outputs/inputs of the same block.

For this purpose, the last caret connector row/column is buffered. Thus, a behavior as in the following picture is possible.



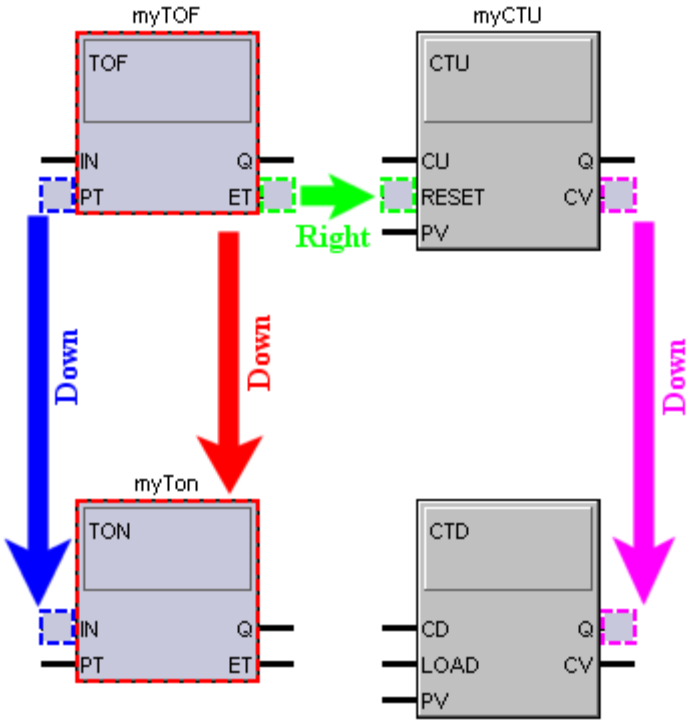
By navigating onto the block field, the caret connector row is not changed and will be evaluated by the next usage of <RIGHT>. The same behavior happens for the caret connector column how we will see in one of the following chapters.

For navigating faster between grid cells with blocks, you can jump directly to the block field by using <ALT> + <UP/DOWN/LEFT/RIGHT>.



Navigating between Grid Cells

Observe the behavior by navigating between grid cells with blocks. By navigating on an empty cell or a cell with a comment, the caret is placed on the comment or the whole grid element with no respect to the starting position. For navigating between grid cells with blocks, the principle of buffering the caret connector row/column as described above is essential.



If there is no connector which fits to the current connector row or column (for example, JMPC), the caret will jump to the block field.

Navigating along Connections

The caret can jump to all connected inputs starting at an output connector. With the methods defined in the chapter „Methods for navigating the caret, you can jump from every input connector to all connected output connectors and vice versa.

Attention: The next output connector is always that one which was connected to the input connector with respect to time.

For these actions, there are entries in the (context) menu:

- **Goto Data Source** : jump to data source
- **Goto Next Data Destination** : jump to next data sink
- **Goto Previous Data Destination** :jump to previous data sink

Fast Navigation with the Caret

Pos1 and End

Pos1 and End **refer only to the grid itself** (the margin is excluded) and locate the caret on the grid in the current row far left or far right.

Ctrl+Pos1 and Ctrl+End

Ctrl+Pos1 and Ctrl+End **refer only to the grid itself** (the margin is excluded) and locate the caret at the upper left or lower right corner of the grid. I.e. Ctrl+Pos1 in FBD jumps to the upper left corner of the first network and Ctrl+End to the lower right corner of the last network.

Page Up and Down

By using Page Up/Down, the visible clip is always aligned to the top edge of a grid cell. It is scrolled only by the number of visible grid cells.

Automatic Post Scrolling

While navigating, the visible clip shall always be scrolled in that way, that the caret (plus a certain amount of tolerance) is visible.

Revoking the Selection

The usage of the <ESC> key revokes the current selection but doesn't change the position of the caret.

Selecting Multiple Elements

By using <CTRL>+<LEFT/RIGHT/UP/DOWN>, multiple elements can be selected. Still, only consistent and valid selections are permitted. (for example,: blocks and border line connectors cannot be selected at the same time)

Attention: While working with the caret, there is no rectangle selection (rubber band selection) possible!

Inline Edit at the Caret Position

If the caret is located on an element, which is inline editable, the element will be selected and opened in the inline edit modus as soon as the user starts to write an alphanumeric sign.

However, if another inline editable element is already selected, that element, which is currently covered by the caret, is set to the inline edit modus.

Insertion of Blocks by Keyboard Usage

The insertion of blocks by keyboard works according to the following procedure:

Call the choosing block dialog by shortcut.

Chose the block type to be inserted.

Close the choosing block dialog and the insert modus is automatically activated.

For finally inserting the block, the caret must be moved to the insert position. **Navigation is only allowed between grid cells.** The caret will be shown as described as in Caret in empty grid cells (EVEN if there is a block in it).

If the caret is moved to a position at which inserting a block is not allowed, the caret will change its figure according to properties for exception situations (see caret properties).

If a valid location for inserting a block was chosen, the block is inserted by using <SPACE> and the caret is placed on the block field.

If an invalid position was chosen and < SPACE > pressed, an event is sent to the automation suite that the insert operation was not successful. The insert operation is aborted and the standard caret is shown.

Moving or Copying Blocks and Margin Connectors by Keyboard

- Blocks can be moved by using <CTRL>+<SHIFT>+<UP/DOWN/LEFT/RIGHT>. As soon as the <CTRL>+<SHIFT> keys are released, the insert operation at the current caret position is made (equivalent to releasing the left mouse button while moving a block/margin connector by mouse). The figure of the caret on invalid positions is according to inserting blocks.
- Margin connectors can by moved by using <CTRL>+<SHIFT>+<UP/DOWN>. As soon as the <CTRL>+<SHIFT> keys are released, the insert operation at the current position of the caret is made. (equivalent to releasing the left mouse button while moving a block/margin connector by mouse). The figure of the caret on invalid position is according to inserting blocks.
- Copying blocks and margin connectors is made by using copy and paste. **Thereby you can only move between grid cells.**

Insert Connections by Keyboard

For inserting a connection by keyboard, two „compoundable elements (block connectors and/or margin connectors) have to be marked by the caret. Afterwards a new connection can be inserted by using the shortcut for the menu „Insert -> Connection.

More comfortable and faster: If the shift key is released while two or more connectors are selected, which allow a connection, this connection is inserted automatically.

Keyboard Combinations for Navigating the Caret

Alt + arrow keys : fast navigation for blocks
Ctrl + arrow keys : multiple selection (for example, connectors or blocks)
Alt + Ctrl + arrow keys : fast multiple selection only for blocks
Shift + arrow keys : release the caret from selection
Shift + Alt + arrow keys : release the caret from selection using fast navigation
Ctrl + shift + arrow keys : moving of blocks or margin connectors

Compound Blocks

Compound Blocks Introduction

Compound Blocks are a way to structure your application.

The work area of the CFC-Editor is limited to one page width. By selecting the paper size, you determine the number of blocks that can be placed horizontally. Vertically, a function chart can grow unlimited.

Although in fact you are not limited in the length of your CFC chart, it is easy to loose overview on a too lengthy chart. Compound Blocks are a means to finer structure your application, hiding groups of logically related blocks inside one `Compound Block`.

Signals between the blocks inside a Compound Block are not visible to the outside. Outside a Compound Block, only those signals are visible that enter or leave the Compound Block.

On screen, double-click the Compound Block to see it's contents. Use ***`View*** > ***Level up`*** or in the toolbar to get back to the location where the Compound Block is being invoked.

Compound Blocks can be nested, i.e. inside a Compound Block you can define, or use, other com pound blocks. The contents of a Compound Block can be edited, you can add or delete blocks, rewire connections, add, modify or delete connections leaving or entering the Compound Block.

On screen, the last input and output connector of a Compound Block is shorter than any other con nector, so you can easily distinguish a Compound Block from other Blocks.

Create Compound Block

To create a new, empty Compound Block,

- Select `Insert > Compound block...`
- The mouse cursor changes.
- Click the mouse where you want to insert the new Compound Block.

You can now fill the Compound Block first, by double-clicking and editing it just like any other function chart. Or, add inputs and outputs to the Compound Block first, editing its contents later using the already provided inputs and outputs then.

Whenever you run out of space on a chart, or think readability would be increased by more hierarchically grouping, you can collapse some of your already wired blocks into a Compound Block:

- Have the Block(s) selected.
- Select `Insert > Compound block...`
- CFC-Editor will prompt you to verify you want to convert the blocks to a Compound Block.

- The selected Blocks will be removed from the chart and replaced by a Compound Block. All signals between these blocks will be moved with the Blocks, all signals to other blocks will be kept and changed to interface signals of the Compound Block.

Note: Currently there is no support for reverting the process of converting a group of blocks to a compound block.

Adding Input or Output to Compound Block

You can edit the contents of a Compound Block just like any other function chart. When you need to provide additional inputs, or need to provide additional outputs, you need to change the interface of the Compound Block accordingly. You can do this from the surrounding (top-down) or from within the Compound Block (bottom-up).

Top-Down

1. Any Compound Block has one very last connector which is shorter than the others. This is always the last connector, one on the left side as an input, one on the right side as an output.
2. Wire this last input or output
3. As soon as you use this last connector, it will be shown in full length, and another shorter connector will be added to the end.

Bottom-Up

1. Double-click a compound block you want to add a connector.
2. Wire a connection of a block inside the compound block to the left or right margin bar (depending whether you want create an in- or output)
3. Click right on the connector and open the ´Properties...´ dialog box via the context menu.
4. Mark the items ´CFC-Connector´ and ´Compound block connector´ name it and close the dialog box by clicking ´OK´.

If you go one level up by clicking the appropriate symbol you see that another shorter unused connector has been added to the compound block.

IEC PLC Debug

Introduction

Test and Commissioning is the tool to maintain all online operation of ACR-View. Use the T+C to monitor the value of variables, to start and stop your controller, and to change online blocks while running the application.

Start and Stop

Test and Commissioning supports three different ways of starting the application: "Cold Start" will reset all variables to their initial value, "Hot

Start" will not reset any variable, while a "Warm Start" will re-initialize only those variables which are not declared RETAIN.

Watch Variables

During a program test, it is important to know which values the variables have, or which value produce an error. Therefore, we have the possibility to watch variables.

- Change to the Resource-Pane.
- Open the branch of the task the variables you want to watch belong to.
- Double click on the variable which you want to watch.

The variable appears in the IEC PLC Debug window where instance path, type, value, and status are displayed. These variables are permanently updated during the program execution on the PLC. If ACR-View can't get a value for a variable from the runtime system (for example, the variable is not available in the currently running program), a "-!-" is shown

To remove variables from the list you have three possibilities as well. Mark the variable by clicking it with the left mouse button then: click on the corresponding symbol in the toolbar or use the `del`-key or select the item **Remove Variable** in the menu `Edit`.

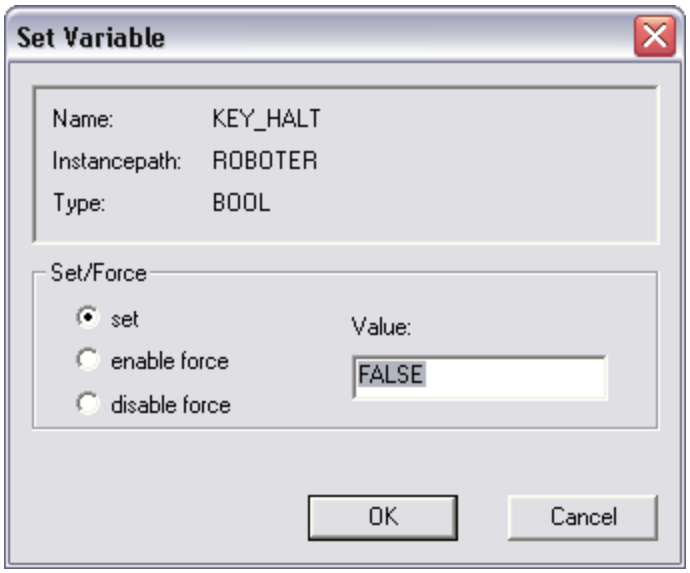
Double click on an array variable opens a dialog where you should enter the index you want to watch. Indexes for multi-dimensional arrays have to be comma separated.

Set Variables

To influence the behavior of your control program for test cases, you can set variables to specific values. Mark the variable in the T+C, and select the menu item ***`PLC→Set variable`***, or click directly on the variable in the T+C. Enter the new value and accept by `Set`-button. See also Force Variables

Force Variables

Besides watching and setting values of variables, ACR-View supports "forcing" of variables. If a variable is forced, the value will be reset to the value specified at the end of each cycle (before writing to the outputs). Forcing is controlled by three buttons labelled "set", "enable force" and "disable force" in the variable set dialog:



In the column "Force" of the IEC PLC Debug window, ACR-View will display if a variable is currently forced or not.

The action performed when pressing OK depends on which of the three buttons "set", "enable force" and "disable force" is selected:

if the variable is currently **not forced**, "set" will once set the variable to the value specified. If the variable is modified by the application, this might have a very short effect only. "enable force" will force the variable to the value specified, i.e. set the variable to the specified value at the end of each cycle, "disable force" will have no effect

if the variable is currently **forced**, "set" will disable forcing for this variable and set the variable once to the value specified, "enable force" will continue to force the variable, but with the value specified now, "disable force" will not set the variable, but only disable forcing for the variable

Please note the following:

- Forcing only resets the variable at the end of each cycle. Modifications during one cycle are possible and not prevented.
- Forcing is not restricted to directly represented variables (AT %...)
- Removing a variable from the watchlist will automatically disable forcing this variable

Working with Watchlists

The Test & Comissioning's list of variables can be saved to a so-called Watch List file. This allows for switching between different Watch Lists while being online.

There is always a default Watch List file with the name *<name of your resource>.WL* in the project root directory.

While online, a Watch List is saved through the main menu command: ***SPS -> Save Watch List As...***

The saved Watch List will then show up in the Browser's File pane. After saving, all subsequent modifications of the variable list will be stored in this Watch List.

To restore a different saved Watch List simply open it by double-clicking it in the Browser. Or by choosing File->Open while the Watch List is selected in the Browser.

An empty Watch List can be created by selecting **File->New / Others / Watch List.**

Documentation

Cross-Reference

See also Cross-Reference (per variable) and CFC Cross-reference.

To create a cross reference list for your project, right-click the active resource and select "crossreference list..." from the context menu.

A preview of the cross reference will be displayed, which can either be viewed and navigated online, or printed.

Cross-Reference (per variable)

Use Cross-Reference list for visualising Cross-Reference information.

Print IEC61131 Configuration

In order to get a printed documentation of the configuration of your resource and tasks, select the configuration in the Browser's resource view an choose "Print Configuration" in the context-menu.

CFC Cross Reference

The CFC cross-reference is a valuable aid in debugging and understanding execution of CFC charts.

The ACR-View standard cross-reference is of limited use to CFC programmers, as most symbols listed in that cross-reference will be symbols which names have been created automatically by the CFC Editor and have no meaning to the programmer.

To create the CFC cross-reference, select File --> Crossreference, or print the chart to see the cross-reference on paper. The cross-reference stored in file is less legible, but better suited to automatic post-processing with third party tools (like grep, awk).

The CFC cross-reference is listed in the form:

source: name [chart] page line

destination1: name [chart] page line

destination2: name [chart] page line

where

- *source* is a name on the right margin bar, i.e., designs a signal leaving one compound block
- *destination* is a name on the left margin bar, i.e. designs a signal entering a compound block
- *name* is the variable name automatically generated by the CFC editor for that signal. Use that name to find this signal in the IEC and PLC Debug Tool to monitor the value of that signal.
- *chart* is a path of names of compound blocks. Use that to find the location either in CFC-Editor by opening one sub-compound block after the other in the specified order, or by locating the printed chart via the table of contents.
- *page* is the page of the printout, where the corresponding source/destination is found.
- *line* is the position of the connection at the block corresponding to the marginbar.

The entries are sorted by source/destination, refer to the file stored if you need other sort sequences.

Note: If IEC61131-variables are used as connectors, there maybe more than one sourceline. They have the following form:

varname{scope}: ...

where

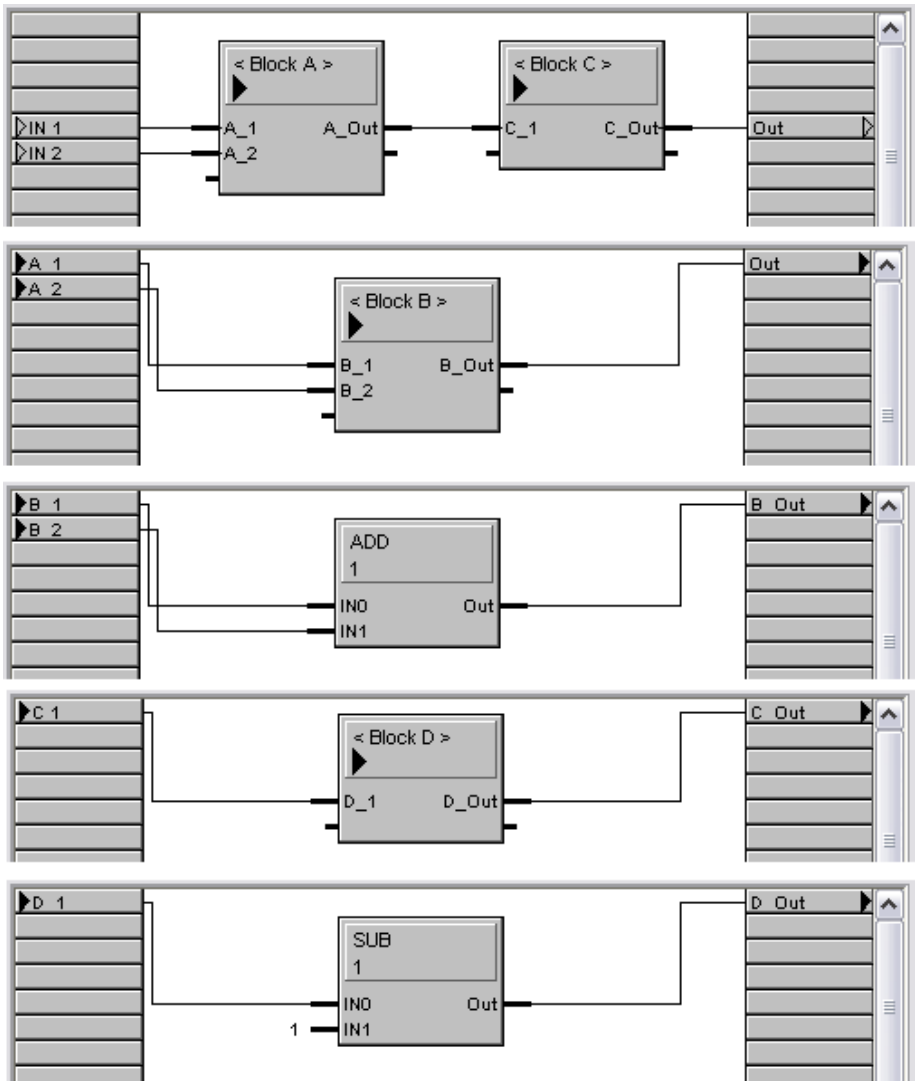
- varname is just the name of the variable.
- scope is represents the declaration section of the variable.

CFC Cross Reference sample

We use a small sample to demonstrate the CFC cross reference.

Set up a small CFC program, using two blocks (ADD and SUB), to add 23 to one input variable, then subtracting one from the result:

Now move block ADD into a compound block A and block SUB into a compound block C. Open block A and move ADD further down into a new compound block B. Open block C and move the SUB block further down into a new compound block D. Enter reasonable names for all margin bar entries. If you open all blocks, the result will look like that:



With this small sample, output of the CFC cross-reference will look like this:

B_Out: FCT_10_10_10_1_ADD_OUT [SAMPLE.chart 1.Block A.Block B] page 4 line 5

D_1: FCT_10_30_10_1_SUB.IN0 [SAMPLE.chart 1.Block C.Block D] page 6 line 5

B_Out: FCT_10_10_10_1_ADD_OUT [SAMPLE.chart 1.Block A.Block B] page 4 line 5

D_1: FCT_10_30_10_1_SUB.IN0 [SAMPLE.chart 1.Block C.Block D] page 6 line 5

In1{VAR}:

B_1: FCT_10_10_10_1_ADD.IN0 [SAMPLE.chart 1.Block A.Block B] page 4 line 5

in2{VAR}:

B_2: FCT_10_10_10_1_ADD.IN1 [SAMPLE.chart 1.Block A.Block B] page 4 line 6

With this, the following questions are easily answered:

Looking at the ADD block: where does this output signal go to? Find the name of the output signal, B_Out. See cross-reference to find it goes to nameD_1 in block chart1.BlockC.BlockD.

looking at the SUB-block: where does the input signal come from? Find the name of the input signal D_1, locate D_1 in the cross-reference and find it

comes from B_Out. (as the list is sorted by source names, this is easier to find by opening the file with some editor than by looking at the printed cross-reference)

How can I monitor that signal entering the SUB-block online? Find the name of the SUB-blocks input in the margin bar (D_1), locate that in the cross-reference and read the name of the IEC61131-variable associated to it (FCT_10_30_10_1_SUB.IN0). Find that variable in the Browser's instance tree and double click it to have it added to the watch list.

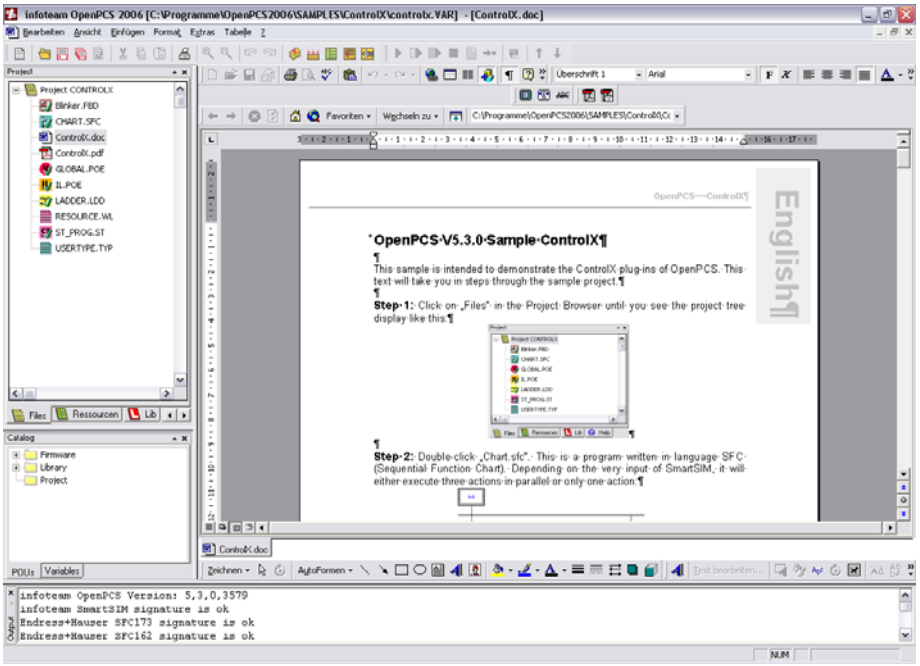
Print Form

All ACR-View tools support forms for printing, and will automatically use the currently "active" print form. To change the active print form, choose **Project** > **Settings** > **Set active form.** You can now choose an available print form (*.wmv).

Active Document Server

ACR-View contains an Active Document Server Interface, this means that all registred active documents are supported by ACR-View, can be opened by ACR-View and can be edited by ACR-View.

When opening such a file, the document is opened in the editor window part of ACR-View as in the figure below.



Attention: Depending on the system configuration and installed applications with active document server, the files that can be edited by ACR-View may vary from PC to PC.

Warning: If the active document server is not stable, this will also lead to an unstable performance of ACR-View.

Libraries

Library Overview

Libraries are collections of functions and function blocks that can be re-used over different ACR-View projects.

Working with libraries involves several steps: a library is first created, pretty much like any other ACR-View project. If creator and user are different, it is then distributed via Floppy Disk, CD-ROM, or Internet, and made available to the user. The user will install the library, i.e. transfer the library to his own PC. To use a library with an ACR-View project, the library has to be added to this project, this making the contents of the library available for use.

To get rid of a library within a project, the library can be removed form this project. This can be necessary if a different implementation of the same library should be used instead.

To remove a library completely from a PC, the library can be uninstalled. This can be necessary if the library should be used on a different PC and licensing conditions require it to be removed prior.

The following chapters will give a sample on how to do a library of your own.

Create a Library

To create a library, proceed just like creating any normal ACR-View project. Be sure to perform a syntax check when finished creating POUs (functions or function blocks) in your library project.

Example

Start the Browser and create a new project named `MyLib` using **Project** > **New...** *Create a function block named `det_edge` (for edge detection):* ***New*** > ***Functionblock*** > ***IL****. Implement this function block as shown below:*

```
VAR_INPUT
    input       :       BOOL ;
END_VAR
VAR_OUTPUT
    output :    BOOL ;
END_VAR
VAR
    tempvar :   BOOL ;
END_VAR
LD      input
ANDN            tempvar
ST      output
LD      input
ST      tempvar
```

Invoke a syntax check with ***File*** > ***Syntaxcheck*** .

Install a Library

Before you can use a library, you have to install it on your PC. Use ***Project*** > ***Library*** > ***Install New...***

Use the `browse`-button to locate the .VAR file representing your project. If you created the library yourself, this will be in the directory you specified when creating the library project with ***Project*** > ***New...***. If you received the library on a disk, this can be something beginning with `**A:\**`. During installation, the library project will be copied into a sub-directory of <windows>\ openpcs.500\Lib.

Example

Create a new project in the Browser using ***Project*** > ***New...***. Name that new project `TEST`.

Select ***Project*** > ***Library*** > ***Install New...***.

Now use the browse-button to locate the MyLib-project you created just before and press `Ok`.

Adding a Library to a Project

After installation, all files needed for the library will be present on your computer. But the functions and function blocks in that library will not be automatically available in your projects. You have to `add` the library to the project first using ***Project*** > ***Library*** > ***Use in current project***.

Example

Mark the Library "MyLib" in the Library-Pane and select ***Project*** > ***Library*** > ***Use in current project*** .

Create a new POU of type PROGRAM, named `main`. Select ***Insert*** > ***Functionblocks....*** to see your library functions. To use your function block DET_EDGE, implement program `main` as shown below:

```
VAR
    sig1 AT %I0.0 : BOOL ;
    anEdge : DET_EDGE;
    count : UINT ;
END_VAR

CAL     anEdge (
        input := sig1
        |
        :=output
)
LDN anEdge.output
JMPC    ende

LD      count
ADD 1
ST      count

ende:
```

Compile that program, add it to a resource of your choice and execute it. Change input %i0.0 and see variable count incremented.

Uninstall Library

If you want to get rid of a library installed on your PC, make sure the library is not used any more, mark it and select ***Project*** > ***Library*** > ***Uninstall***. In the dialog shown, select the library to get rid of and press OK.

Example

Mark the Library "MyLib" in the Library-pane.

Select ***Project*** > ***Library*** > ***Uninstall***. In the dialog, select <*Windows*>\openpcs.500\MyLib`.

Press OK, and `MyLib` is no longer available as a library.

IEC61131-3

IEC61131-3 Details

Character String Literals

A string constant is sequence of characters enclosed in ‘’’. Special characters can be embedded within a character string literal by using escape sequences starting with the $ sign, as listed in the following table:

Predefined character constants	Meaning
'$''	The Apostrophe ‘’’
'$$'	The $ sign itself
'$L' or '$l'	Line Feed
'$N' or '$n'	New Line
'$P' or '$p'	Form Feed
'$R' or '$r'	Carriage Return
'$T' or '$t'	Tabulator

Example

Character Constant	Meaning and Length
'A'	Single character A, length=1
' '	Blank character, length=1
''	No character, length=1
'RL'	Carriage Return, Line Feed, length=2
'$0D$0A'	Carriage Return, Line Feed, length=2

Maximum String Length

Each string is delimited by a maximum length. The default maximum length of a string is 32 characters. It can be changed setting an individual maximum string length in round brackets immediately after the keyword STRING.

The maximum string length can be set to all values from 0 to 251. However this may differ at other hardwares.

Examples

TYPE

```
    name: STRING(15) := 'John Q. Public';     (*maximum string length 15*)
    address: STRING(50) := 'Main Street 1, 12345 Springfield, ???'; (*maximum
string length 50*)
END_TYPE

VAR
    user: name;                               (*maximum string length 15*)
    id: string(8) := '12345678';              (*maximum string length 8*)
    phone : STRING;                           (*maximum string length 32*)
END_VAR
```

Constants

Within a literal constant, underscores are allowed to increase readability. Such underscores have no meaning regarding the value of a constant. Literal constants for some data types require a special prefix.

Constant Data Type	Example	Meaning
INT	-13 45165 or 45_165 +125	Integer -13 Integer 45165 (both) Integer 125
REAL	-13.12 123.45 0.123 -1.23E-3	Real -13,12 Real 123,45 Real 0,123 Real -0,00123
Dual number	2#0111_1110 or 126	126
Octal number	8#123 or 83	83
Hexadecimal number	16#123 or 291	291
BOOL	0 and 1 TRUE and FALSE	Boolean TRUE and FALSE values
STRING	'ABC'	Character string ABC
WSTRING	ABC"	2-byte-character string ABC
TIME	T#12.3ms or TIME#12.3ms	Time duration of 12,3 milliseconds
	T#12h34m or T#12h_34m	Time duration of 12 hours and 34 minutes
	T#-4m	Negative time duration of 4 minutes

Constant Data Type	Example	Meaning
DATE	DATE#1995-12-24 or D#1995-12-24	Date 24.12.1995
TIME_OF_DAY	TOD#12:05:14.56 or TIME_OF_DAY#12:05:14.56	12 hours05 minutes and 14,56 seconds PM
DATE_AND_TIME	DT#1995-12-24-12:05:14.56 or DATE_AND_TIME#1995-12-24-12:05:14.56	Date and time: 12 hours05 minutes and 14,56 seconds PM on 24.12.1995

Literal constants of data types TIME, DATE and DATE_AND_TIME uses keywords plus a hash sign "#". The keywords can be written in long (for example, DATE_AND_TIME) or short form (for example, DT).

Note: DATE, TIME_OF_DAY and DATE_AND_TIME are currently not supported by ACR-View.

See also Elementary Data Types

Single Bit Access

With ACR-View, each individual bit of BYTE or WORD variable can be accessed by writing the bitnumber, separated by a dot, after the variable name

Example

```
PROGRAM Only_1_Bit
VAR
     Bitpattern1 : BYTE := 2#10101010;
     Bitpattern2 AT %IW0.0 : WORD;
END_VAR
LD Bitpattern2.15  (* Copy bit 15 *)
  ST Bitpattern1.0   (* into bit 0 *)
   .
   .
END_PROGRAM
```

Please note that this feature might not be available on all hardware platforms for all data types due to implementation restrictions.

Passing Output Parameters

IEC61131 defines two ways of passing parameters. ACR-View provides, as a legal extension to IEC61131, a means to directly pass output parameters. You can pass output parameters within the line of the CAL instruction by using a vertical slash "|" instead of a comma, and giving the actual parameter on the left side of the assignment:

Example

```
CAL SR_Instance_1(SET1 := On,
                  RESET := Off
                  |
                  Result := Q1)
```

Nested Comments

Comments may be nested, which eases out-commenting of entire program sections which should contain comments on their own.

Block Type: Program, Function, Function Block

A **program** in ACR-View has the following characteristic properties, as defined by IEC61131: Only the program is allowed to declare variables to be mapped to physical addresses; A program is allowed to call functions and instances of function blocks.

A **function block**, as defined by IEC61131, has the following characteristic properties: It may have one, more than one, or no inputs; It may have one, more than one, or no outputs; Multiple instances can be created of a function block, and each instance will keep a private copy of all data associated with that function block (input, output, intermediate data); a func tion block cannot be called, only instances can be called. The function block has a `memory`, i.e. all data (input, output, local) will keep it's value from one call to the next. On a call, it is not necessary to supply all input data; those not provided will simply keep the value from the previous call (or the default value if there was no call before). A function block can call functions and instances of other function blocks.

A **function**, as defined by IEC61131, has the following characteristic properties: It has one or more inputs (but no input is not allowed); It has exactly one output value (which may be a structure); A function has no `memory` from one call to the next, and it will return always the same out put when given the same inputs. On every call to a function, all inputs have to be supplied. A function may use local variables for intermediate storage, but the value of these local vari ables will not be kept from one call to the next. A function may call other functions, but it is not allowed to call instances of function blocks.

IEC61131-3 Compliance Statement

Compliance Statement

The following tables have the same numbering as those in the IEC61131-3/EN 61131-3 standard. Tables showing features not yet supported by this version of ACR-View are not listed. Some tables in IEC61131-3 do not contain features, so missing table numbers do not necessarily imply missing features. To understand this document, you will want to consult IEC61131-3.

This version of ACR-View complies with the requirements of IEC61131-3, for the following language features:

No.	Description	Yes	No
1	Required character set	x	
2	Lower case	x	
3a	Number sign (#)	x	
	or		
3b	Pound sign (£)		x

No.	Description	Yes	No
4a	Dollar sign ($)	x	
	or		
4b	Currency sign		x
5a	Vertical bar (|)		x
	or		
5b	Exclamation mark (!)	x	
	Subscript delimiters:		
6a	brackets [ ]	x	
	or		
6b	parentheses ( )		x

Table 1: Character Set Features

No.	Description	Yes	No
1	Upper case and numbers	x	
2	Upper and lower case, numbers, embedded underlines	x	
3	Upper and lower case, numbers, leading or embedded underlines	x	

Table 2: Identifier Features

No.	Description	Yes	No
1	Comments	x	

Table 3: Comment Features

No.	Description	Yes	No
1	Integer literals	x	
2	Real literals	x	
3	Real literals with exponents	x	
4	Base 2 literals	x	
5	Base 8 literals	x	
6	Base 16 literals	x	
7	Boolean zero and one	x	
8	Boolean FALSE and TRUE	x	

Table 4: Numeric Literals

No.	Description	Yes	No
1	Empty string (length zero)	x	
	String of length one containing the single character A	x	
	String of length one containing the `space` character	x	
	String of length one containing the `single quote` character	x	
	String of length two containing CR and LF	x	
	String of length five which would print as `$1.00`		

Table 5: Character String Literal Features

No.	Description	Yes	No
2	Dollar sign ($$)	x	
3	Single quote ($´)	x	
4	Line feed ($L or $l)	x	
5	New line ($N or $n)	x	
6	New page ($P or $p)	x	
7	Carriage return ($R or $r)	x	
8	Tab ($T or $t)	x	

Table 6: Two Character Combinations in Character Strings

No.	Description	Yes	No
	Duration literals without underlines:		
1a	Short prefix	x	
1b	Long prefix	x	
	Duration literal with underlines		
2a	Short prefix	x	
2b	Long prefix	x	

Table 7: Duration Literal Features

No.	Description	Yes	No
1	Date literals (long prefix: DATE#)		x
2	Date literals (short prefix: D#)		x
3	Time of day literals (long prefix: TIME_OF_DAY#)		x
4	Time of day literals (short prefix: TOD#)		x

No.	Description	Yes	No
5	Date and time literals (long prefix: DATE_AND_TIME#)		x
6	Date and time literals (short prefix: DT#)		x

Table 8: Date and Time of Day Literals

No.	Keyword	Data type	Yes	No
1	BOOL	Boolean	x	
2	SINT	Short integer	x	
3	INT	Integer	x	
4	DINT	Double integer	x	
5	LINT	Long integer		x
6	USINT	Unsigned short integer	x	
7	UINT	Unsigned integer	x	
8	UDINT	Unsigned double integer	x	
9	ULINT	Unsigned long integer		x
10	REAL	Real numbers	x	
11	LREAL	Long real numbers	x	
12	TIME	Duration	x	
13	DATE	Date (only)	x	
14	TIME_OF_DAY or TOD	Time of day (only)	x	
15	DATE_AND_ TIME or TD	Date and time	x	
16	STRING	Variable-length character string	x	
17	BYTE	Bit string of length 8	x	
18	WORD	Bit string of length 16	x	
19	DWORD	Bit string of length 32	x	
20	LWORD	Bit string of length 64		x

Table 9: Elementary Data Types

No.	Description	Yes	No
1	Direct derivation from elementary types	x	
2	Enumerated data types	x	
3	Subrange data types		x
4	Array data types	x	
5	Structured data types	x	

Table 10: Data Type Declaration Feature

Description	Initial value	Yes	No
BOOL, SINT, INT DINT, LINT,	0	x	
USINT, UINT, UDINT, ULINT	0	x	
BYTE, WORD, DWORD, LWORD	0	x	
REAL, LREAL	0.0	x	
TIME	T#0s	x	
DATE	D#0001-01-01		x
TIME_OF_DAY	TOD#00:00:00		x
DATE_AND_TIME	DT#0001-01-01-00:00:00		x
STRING	`(the empty string)	x	

Table 11: Default Initial Values

No.	Description	Yes	No
1	Initialization of directly derived types	x	
2	Initialization of enumerated data types	x	
3	Initialization of subrange data types		x
4	Initialization of array data types	x	
5	Initialization of structured data types	x	
6	Initialization of derived structured data types	x	

Table 12: Data Type Initial Value Declaration Features

No.	Description	Yes	No
1	I: Input location	x	
2	Q: Output location	x	
3	M: Marker location	x	
4	X: (Single) bit size	x	
5	None: (Single) bit size	x	
6	B: Byte (8 bits) size	x	
7	W: Word (16 bits) size	x	
8	D: Double word (32 bits) size	x	
9	L: Long word (64 bits) size		x

Table 13: Location and size prefix features for directly represented variables

Keyword	Yes	No
VAR	x	
VAR_INPUT	x	
VAR_OUTPUT	x	

Keyword	Yes	No
VAR_IN_OUT	x	
VAR_EXTERNAL	x	
VAR_GLOBAL	x	
VAR_ACCESS		x
RETAIN	x	
CONSTANT	x	
AT	x	

Table 14: Variable keywords for variable declaration

No.	Description	Yes	No
1	Declaration of directly represented, non-retentive variables	x	
2	Declaration of directly represented, retentive variables	x	
3	Declaration of locations of symbolic variables	x	
4	Array location assignment		x
5	Automatic memory allocation of symbolic variables	x	
6	Array declaration	x	
7	Retentive array declaration	x	
8	Declaration of structured variables	x	

Table 15: Variable type assignement features

No.	Description	Yes	No
1	Initialization of directly represented, non-retentive variables		x
2	Initialization of directly represented, retentive variables		x
3	Location and initial value assignment to symbolic variables		x
4	Array location assignment and initialization		x
5	Initialization of symbolic variables	x	
6	Array initialization		x
7	Retentive array declaration and initialization		x
8	Initialization of structured variables	x	
9	Initialization of constants	x	

Table 16: Variable initial value assignment features

No.	Description	Yes	No
1	Negated input	x	
2	Negated output		x

Table 17 Graphical negation of Boolean signals

No.	Description	Yes	No
1	Use of EN and ENO		x
2	Use of EN and ENO		x
3	FBD without EN and ENO	x	

Table 18: Use EN input an ENO output

No.	Description	Yes	No
1	Overloaded functions (non type-dependent)	x	
2	Typed functions	x	

Table 19: Typed and overloaded functions

No.	Description	Yes	No
1	*_TO_**	x	
2	TRUNC	x	
3	BCD_TO_**		x
4	*_TO_BCD		x

Table 20: Type conversion function features

Comment:

If you are using TIME-values, only TIME_TO_DINT and DINT TO_TIME are implemented. Other TIME-cast-functions are only available within the Ladder-Diagram-Editor.

For no. 1, (*) is the input variable data type and (**) is the output variable data type. The following data types are supported:

- BOOL
- BYTE
- DINT
- DWORD
- INT
- REAL
- SINT
- STRING

- TIME
- UDINT
- UINT
- WORD

No.	Description	Yes	No
1	ABS	x	
2	SQRT	x	
3	LN	x	
4	LOG	x	
5	EXP	x	
6	SIN	x	
7	COS	x	
8	TAN	x	
9	ASIN	x	
10	ACOS	x	
11	ATAN	x	

Table 21: Standard functions of one numeric variable

No.	Name	Symbol	Yes	No
12	ADD	+	x	
13	MUL	*	x	
14	SUB	-	x	
15	DIV	/	x	
16	MOD		x	
17	EXPT	**		x
18n	MOVE			x
18s		:=	x	

Table 22: Arithmetic standard functions

No.	Name	Yes	No
1	SHL	x	
2	SHR	x	
3	ROR	x	
4	ROL	x	

Table 23: Standard bit shift functions

No.	Name	Yes	No
5	AND	x	
6	OR	x	
7	XOR	x	
8	NOT	x	

Table 24: Standard bitwise Boolean functions

No.	Name	Yes	No
1	SEL		x
2a	MAX	x	
2b	MIN	x	
3	LIMIT		x
4	MUX		x

Table 25: Standard selection functions

No.	Name	Yes	No
5	GT	x	
6	GE	x	
7	EQ	x	
8	LE	x	
9	LT	x	
10	NE	x	

Table 26: Standard comparison functions

No.	Name	Yes	No
1	LEN	x	
2	LEFT	x	
3	RIGHT	x	
4	MID	x	
5	CONCAT	x	
6	INSERT	x	
7	DELETE	x	
8	REPLACE	x	
9	FIND	x	

Table 27: Standard character string functions

No.	Name	Operation	Yes	No
1	ADD	TIME + TIME = TIME	x	
2		TOD + TIME = TOD		x
3		DAT + TIME = DAT		x
4	SUB	TIME - TIME = TIME	x	
5		DATE - DATE = TIME		x
6		TOD - TIME = TOD		x
7		TOD - TOD = TIME		x
8		DAT - TIME = DAT		x
9		DAT - DAT = TIME		x
10	MUL	TIME * ANY_NUM = TIME		x
11	DIV	TIME / ANY_NUM = TIME		x
12	CONCAT	DATE TOD = DAT		x
		Type conversion functions		
13		DATE_AND_TIME_TO_TIME_OF_DAY		x
14		DATE_AND_TIME_TO_DATE		x

Table 28: Functions of time data types

No.	Name	Yes	No
1	SR	x	
2	RS	x	
3	SEMA		x

Table 29: Standard bistable function blocks

No.	Name	Yes	No
1	SEL		x
2	MUX		x
3	EQ		x
4	NE		x

Table 30: Functions of enumerated data types

No.	Description	Yes	No
1	RETAIN qualifier on internal variables	x	
2	RETAIN qualifier on output variables	x	
3	RETAIN qualifier on internal function blocks		x
4a	Input/output declaration (textual)	x	
4b	Input/output declaration (graphical)		x
5a	Function block instance name as input (textual)		x
5b	Function block instance name as input (graphical)		x

No.	Description	Yes	No
6a	Function block instance name as input/output (textual)		x
6b	Function block instance name as input/output (graphical)		x
7a	Function block instance name as external variable (textual)		x
7b	Function block instance name as external variable (graphical)		x
	Textual declaration of		
8a	- rising edge inputs	x	
8b	- falling edge inputs	x	
	Graphical declaration of		
9a	- rising edge inputs		x
9b	- falling edge inputs		x

Table 31: Function block declaration features

No.	Name	Yes	No
1	R_TRIG	x	
2	F_TRIG	x	

Table 32: Standard edge detection function blocks

No.	Name	Yes	No
1	R_TRIG	x	
2	F_TRIG	x	

Table 33: Standard counter function blocks

No.	Name	Yes	No
1	TP (Pulse)	x	
2a	TON (on-delay)	x	
2b	T---0 (on-delay)		x
3a	TOF (off-delay)	x	
3b	0---T (off-delay)		x
4	RTC (real-time clock)		x

Table 34: Standard timer function blocks

No.	Description	Yes	No
1	RETAIN qualifier on internal variable	x	
2	RETAIN qualifier on output variable		x
3	RETAIN qualifier on internal function blocks		x
4a	Input/output declaration (textual)		x
4b	Input/output declaration (graphical)		x
5a	Function block instance name as input (textual)		x
5b	Function block instance name as input (graphical)		x
6a	Function block instance name as input/output (textual)		x
6b	Function block instance name as input/output (graphical)		x
7a	Function block instance name as external variable (textual)		x
7b	Function block instance name as external variable (graphical)		x
	Textual declaration of:		
8a	- rising edge inputs		x
8b	- falling edge inputs		x
	Graphical declaration of:		
9a	- rising edge inputs		x
9b	- falling edge inputs		x
10	Formal input and output parameters		x
11	Declaration of directly represented, non-retentive variables	x	
12	Declaration of directly represented, retentive variables	x	
13	Declaration of locations of symbolic variables	x	
14	Array location assignment		x
15	Initialization of directly represented, non-retentive variables		x
16	Initialization of directly represented, retentive variables		x
17	Location and initial value assignment to symbolic variables		x
18	Array location assignment and initialization		x
19	Use of directly represented variables	x	
20	VAR_GLOBAL .. END_VAR Declaration within a PROGRAM	x	
21	VAR_ACCESS .. END_VAR Declaration within a PROGRAM		x

Table 35: Program declaration features

No.	Description	Yes	No
1	Step graphical	x	
	Initial step graphical	x	
2	Step textual		x
	Initial Step textual		x
3a	Step flag general form		x
3b	Step flag - direct connection of boolean variable		x
4	Step elapsed time		x

Table 36: Step features

No.	Description	Yes	No
1	Transition condition using ST language**Fehler! Textmarke nicht definiert.**		x
2	Transition condition using LD language		x
3	Transition condition using FBD language		x
4	Use of connector		x
4a	Transition condition using LD language**Fehler! Textmarke nicht definiert.**		x
4b	Transition condition using FBD language		x
5	Textual transition in ST		x
6	Textual transition in IL	x	
7	Transition name	x	
7a	Transition condition using LD language		x
7b	Transition condition using FBD language		x
7c	Transition condition using IL language	x	
7d	Transition condition using ST language		x

Table 37: Transitions and Transition conditions

No.	Description	Yes	No
1	boolean variable as action		x
2l	graphical declaration in LD language		x
2s	inclusion of SFC elements in action		x
2f	graphical declaration in FBD language		x
3s	textual declaration in ST language		x
3i	graphical declaration in IL language	x	

Table 38: Declaration of actions

No.	Description	Yes	No
1	action block		x
2	concatenated action blocks		x
3	textual step body	x	
4	action block `d` field		x

Table 39: Step/action association

No.	Description	Yes	No
1	qualifier as per 2.6.4.4		x
2	action name	x	
3	boolean indicator variables		x
4	IL language		x
5	ST language		x
6	LD language		x
7	FBD language		x
8	action blocks in ladder diagrams		x
9	action block in function block diagrams		x

Table 40: Action block features

No.	Description	Yes	No
1	None	x	
2	N (non-stored)	x	
3	R (overriding reset)		x
4	S (set stored)		x
5	L (time limited)		x
6	D (time delayed)		x
7	P (pulse)		x
8	SD (stored and time delayed)		x
9	DS (delayed and stored)		x
10	SL (stored and time limited)		x

Table 41: Action qualifiers

No.	Description	Yes	No
1	single sequence	x	
2a	divergence of sequence selection (left-to-right)	x	
2b	divergence of sequence selection (with priorities)		x
2c	divergence of sequence selection (with mutual exclusion)		x
3	Convergence of sequence evolution	x	

No.	Description	Yes	No
4	simultaneous sequence divergence	x	
5	simultaneous sequence convergence	x	
5a	sequence skip (left-to-right)	x	
5b	sequence skip (with priorities)		x
5c	sequence skip (with mutual exclusion)		x
6a	sequence loop (left-to-right)		x
6b	sequence loop (with priorities)		x
6c	sequence loop (with mutual exclusion)		x
7	directional arrows		x

Table 42: Sequence evolution

No.	Operator	Modifiers	Yes	No
1	LD	N	x	
2	ST	N	x	
3	S		x	
	R		x	
4	AND	N,(	x	
5	&	N,(	x	
6	OR	N,(	x	
7	XOR	N,(	x	
8	ADD	(	x	
9	SUB	(	x	
10	MUL	(	x	
11	DIV	(	x	
12	GT	(	x	
13	GE	(	x	
14	EQ	(	x	
15	NE	(	x	
16	LE	(	x	
17	LT	(	x	
18	JMP	C, N	x	
19	CAL	C, N	x	
20	RET	C, N	x	
21	)		x	

Table 43: Instruction list (IL) operators

No.	Description	Yes	No
1	CAL with input list	x	
2	CAL with load/store of inputs	x	
3	Use of input operators		x

Table 44: Function block invocation features for IL language

No.	Description	Yes	No
1	Parenthesation	x	
2	Function evaluation	x	
3	Exponentiation		x
4	Negation	x	
5	Complement	x	
6	Multiply	x	
7	Divide	x	
8	Modulo	x	
9	Add	x	
10	Subtract	x	
11	Comparison	x	
12	Equality	x	
13	Inequality	x	
14	Boolean AND	x	
15	Boolean AND	x	
16	Boolean Exclusive XOR	x	
17	Boolean OR	x	

Table 45: Operators of the ST language

No.	Description	Yes	No
1	Assignment	x	
2	Function block invocation and FB output usage	x	
3	RETURN	x	
4	IF	x	
5	CASE	x	
6	FOR	x	
7	WHILE	x	
8	REPEAT	x	
9	EXIT	x	
10	Empty Statement	x	

Table 46: ST language statements

No.	Description	Yes	No
	Horizontal lines:		
1	ISO/IEC 646 `minus` character		x
2	graphic or semigraphic	x	
	Vertical lines:		
3	ISO/IEC 646 `vertical line` character		x
4	graphic or semigraphic	x	
	Horizontal/vertical connection:		
5	ISO/IEC 646 `plus` character		x
6	graphic or semigraphic	x	
	Line crossing without connection:		
7	ISO/IEC 646 characters		x
8	graphic or semigraphic	x	
	Connected and non-connecte corners:		
9	ISO/IEC 646 characters		x
10	graphic or semigraphic	x	
	Blocks with connecting lines		
11	ISO/IEC 646 characters		x
12	graphic or semigraphic	x	
	Connectors using ISO/IEC 646 characters:		
13	Connector, Continuation of a connected line		x
14	graphic or semigraphic	x	

Table 47: Representation of lines and block

No.	Description	Yes	No
	Unconditional Jump		
1	FBD language	x	
2	LD language	x	
3	Conditional Jump (FBD language)	x	
4	Conditional Jump (LD language)	x	
	Conditional Return		
5	LD language	x	
6	FBD language	x	
	Unconditional Return		
7	from Function	x	
	from Function Block	x	
8	Alternative Representation in LD language	x	

Table 48: Graphic execution control elements

No.	Description	Yes	No
1	Left power rail	x	
2	Right power rail	x	

Table 49: Power rails

No.	Description	Yes	No
1	Horizontal link	x	
2	vertical link with attached horizontal links	x	

Table 50: Link Elements

No.	Description	Yes	No
	Normally open contact		
1		x	
2		x	
	Normally closed contact		
3		x	
4		x	
	Positive transition-sensing contact		
5			x
6			x
	Negative transition-sensing contact		
7			x
8			x

Table 51: Contacts

No.	Description	Yes	No
1	Coil	x	
2	Negated Coil	x	
3	SET (latch) coil	x	
4	RESET (unlatch) coil	x	
5	Retentive (Memory) coil		x
6	SET retentive (Memory) coil		x
7	RESET retentive (Memory) coil		x
8	Positive transition-sensing coil		x
9	Negative transition-sensing coil		x

Table 52: Coils

Names of data types cannot be used for file or variable names. The following names are also not allowed for variables and/or files:

Names Not Allowed for Variables and Files
D
L
N
P
Q

Table 53: Reserved Names

Clause	Parameter	Values
1.5.1	Error handling procedures	see next chapter
2.1.1	National characters used	see table 1 above
2.1.2	Maximum length identifiers Significant length identifiers	256 64
2.1.5	Maximum comment length	>512
2.2.3.1	Range of values of duration	+/- 24,85 days
2.3.1	Range of values for variables of type TIME Precision of representation of seconds in type TIME_OF_DAY and DATE_AND_TIME	+/- 24,85 days -
2.3.3	Maximum - number of array subscripts - array size - number of structure elements - structure size - number of variables per declaration	 6 < 4KB per POU < 8KB per POU
2.3.3.1	Maximum number of enumerated values	< 64 KB per POU
2.3.3.2	Default maximum length of STRING variables Maximum permissible length of STRING variables	32 253 [see note 1]
2.4.1.1	Maximum number of hierarchical levels Logical or physical mapping	5
2.4.1.2	Maximum number of subscripts	-

Clause	Parameter	Values
	Maximum number of subscript values	-
	Maximum number of levels of structures	>512
2.4.2	Initialization of system inputs	The value of the system inputs corresponds to their physical values
2.4.3	Maximum number of variables per declaration	< 64 KB per POU
2.5	Information to determine execution times of program organization units	No
2.5.1.1	Method of function representation	Textual
2.5.1.3	Maximum number of function specifications	limited only by available memory
2.5.1.5	Maximum number of inputs of extensible functions	IL: 2, LD/FBD: unlimited
2.5.1.5.1	Effects of type conversions on accuracy	Truncated
2.5.1.5.2	Accuracy of functions of one variable	Currently not supported
	Implementation of arithmetic functions	
2.5.2	Maximum number of function blocks and instantiations	ca. 8000
2.5.2.3.3	PVmin, PVmax of counters	minimum/maximum value of respective data type
2.5.3	Program size limitations	limited only by available memory
2.6	Timing and postability effects of execution control elements	-
2.6.2	Precision of step elapsed time	-
	Maximum number of steps per SFC	
2.6.3	Maximum number of transitions per SFC and per step	-
2.6.4	Action control mechanism	-
2.6.4.2	Maximum number of action blocks per step	-
2.6.5	Graphic indication of step state	-
	Transition clearing time	
	Maximum width of diverge/converge constructs	
2.7.1	Content of RESOURCE libraries	-

Clause	Parameter	Values
2.7.2	Maximum number of tasks Task interval resolution Pre-emptive or non-pre-emptive scheduling	-
3.3.1	Maximum length of expressions Partial evaluation of Boolean expressions	unlimited no
3.3.2	Maximum length of statements	Unlimited
3.3.2.3	Maximum number of CASE selections	Unlimited
4.1.1	Graphic/semigraphic representation Restrictions on network topology	Graphic
4.1.3	Evaluation order of feedback loops	-
Note 1: ACR-View is highly configurable, so this parameter may vary depending on your hardware. If in doubt, consult the documentation of your hardware.		

Table 54: Implementation-dependent parameters

2.3.3.1	Value of a variable exceeds the specified subrange	Syntax error reported for initialization in declaration; ignored at runtime
2.4.2	Length of initialization list doesn't match the number of array entries	Syntax error
2.5.1.5.1	Type conversion errors	Ignored
2.5.1.5.2	Numerical result exceeds range for data type Division by zero	firmware blocks report that at ENO, ignored elsewhere
2.5.1.5.4	Mixed input data types to a selection function Selector (K) out of range for MUX function	not supported
2.5.1.5.5	Invalid character position specified. Result exceeds maximum string length	-
2.5.1.5.6	Result exceeds range for data type	Restriction to maximum value (see 2.2.3.1)
2.6.2	Zero or more than one initial step in the SFC network User program attempts to modify step state or time	-
2.6.2.5	Simultaneously true, non-	-

2.3.3.1	Value of a variable exceeds the specified subrange	Syntax error reported for initialization in declaration; ignored at runtime
	prioritized transitions in a selection divergence	
2.6.3	Side effects in evaluation of transition condition	-
2.6.4.5	Action control contention error	-
2.6.5	`Unsafe` or `Unreachable` SFC	-
2.7.1	Data type conflict in VAR_ACCESS	-
2.7.2	Tasks require too many processor resources Execution deadline not met Other task scheduling conflicts	-
3.2.2	Numerical result exceeds range for data type	Scan via functions
3.3.1	Division by zero Invalid data type for operation	Syntax error can be monitored
3.3.2.1	Return from function without value assigned	-
3.3.2.4	Iteration fails to terminate	-
4.1.1	Same identifier as connector label and element name	-
4.1.4	Uninitialized feedback variable	-
4.1.5	Numerical result exceeds range for data type Division by 0	-

Table 55: Error conditions

Online Features

Breakpoints

ACR-View supports Breakpoints in textual languages ST and IL. Breakpoints are currently not supported in Native Code, so set optimzation to "size." Breakpoints are not supported with all targets due to hardware restrictions. Breakpoints are not saved, so set new breakpoints before starting a newly downloaded application.

If a breakpoint is reached in any one task of the ACR-View application, execution of all tasks immediately will be stopped. When single-stepping, continuing to the next breakpoint, etc., it is undefined and left to the controller whether other tasks should be executed in the meantime. Therefore, it is recommended to have one task only when single-stepping intuitively.

Stopping a controller with breakpoints and single-stepping can disable many of the safety precautions in your controller and your application, so be sure to take appropriate measures so guarantee damage to be avoided.

Online Edit

Online Edit (or Online Change) is a feature whereby program changes are applied to the PLC without the need to restart it.

The system should be saved afterward via **PLC** > **Save System...** if the changes should be maintained on the controller. For further Information see the respective section.

Online Edit consists of the following steps:

- The user starts the application of the changes.
- The compilation process is carried out and the changes are downloaded asynchronously to the controller while the program is still being executed.
- Once the download has finished, the changes are applied at the next cycle end.

As a restart is not necessary, variable values of program parts that are not affected by the changes will keep their current values (i.e. they will not be reset to their initial values). This, however, is dependent on the complexity of the changes. A detailed description of the impacts of Online Edit is given below.

To perform an Online Edit, proceed as follows:

- In Online Mode, switch an editor to edit mode by PLC->Monitor/Edit (or use toolbar button Monitor/Edit)
- Modify declarations and code in the editor as required
- Switch back to Monitor Mode by using Monitor/Edit
- Now you will be prompted to update the controller. Select "Yes" to save any modifications, recompile the application, and download your modifications to the controller without stopping the program.
- Select "No" to abort Online Edit and to discard all changes (also: no modifications will be saved to file).

Impact of Changes

Online Edit applies to two components: programs and (firmware) function blocks.

These are unified under the term Program Organization Units (POUs). A POU consists of a declaration section and code section.

POU Change	POU's Variables Reset?	Details
Program		
Declaration	YES	The program's variables are reset to their initial values. This applies to: local variable (*VAR* section) global variables (*VAR_GLOBAL* section) It does *not* apply to: external variables (*VAR_EXTERNAL* section) function block instances (*VAR*, *VAR_GLOBAL* or *VAR_EXTERNAL* section) Since both are external POUs.
Code	NO	Code changes never lead to a reset of any variable values.
Function Block		
Declaration	YES	A change affects *all* instances of the function block! Apart from that, the same as for programs applies: Local variables of the function block will be reset, while external variables and sub function block instances will not be reset.
Code	NO	Code changes never lead to a reset of any variable values.
Resource Global Declarations	YES	Variables in the *VAR_GLOBAL* section will be reset. Again, this does not include globally defined function block instances (see above).
Functions	-	Strictly, functions are also POUs. Since they are stateless, they need not be treated by Online Edit, however.

Save System

PLC > **Save System...** writes the complete system persistent on the controller. This needs to be done if changes were made via online edit.

Error Logs

A detailed Error Log can be uploaded from the controller via **PLC** > **Upload Error Log**. The uploaded file will be named yymmdd_hhmmssErrorlog.txt and will be stored in the current project directory.

Reference Listings

Keywords (by category)

IEC61131 Standard Function Blocks

ACR-View implements the following function blocks of IEC61131-3:

CTD

CTU

CTUD

F_TRIG

R_TRIG

RS

SR

TOF

TON

TP

IEC61131-3 Standard Functions

ACR-View implements the following functions of IEC61131-3:

ABS

ACOS

AND

ASIN

ATAN

CONCAT

COS

DELETE

EQ

EXP

FIND

GE

GT

INSERT

LE

LEFT

LEN

LIMIT

LN

LOG

LT

MAX

MID

MIN

MOD

MUX

NE

NEG

OR

REAL_TO_*

RIGHT

ROL

ROR

SHL

SIN

SHR

SQRT

TAN

TIME_TO_*

TRUNC

XOR

RIGHT

IEC61131-3 Operations

ACR-View implements the following operations of IEC61131-3:

ADD

ADD (time)

DIV

DIV (time)

MUL

MUL (time)

SUB

SUB (time)

ACR-View Functions and Function Blocks

The following functions and function blocks are provided by ACR-View in addition to IEC61131-3:

GetTaskInfo

GetTime

GetVarData

GetVarFlatAddress

Data Types

The following elementary data types are defined by IEC61131-3:

BOOL

BYTE

DATE_AND_TIME

DATE

DINT

DWORD

INT

REAL

SINT

STRING

TIME_OF_DAY

TIME

UDINT

UINT

WORD

The following data types are defined by ACR-View in addition to IEC61131-3:

POINTER

VARINFO

Declaration Keywords

END_TYPE

END_VAR

RETAIN

TYPE

VAR_GLOBAL

VAR_IN_OUT

VAR_INPUT

VAR_OUTPUT

VAR

Structured Text Keywords

ACR-View uses the following keywords in Programming Language Structured Text:

:= (Assignment)

BY

CASE

DO

ELSE

ELSIF

END_CASE

END_FOR

END_IF

END_REPEAT

END_WHILE

EXIT

FOR

IF

OF

REPEAT

RETURN

TO

UNTIL

WHILE

Others

ACTION

ANY

ANY_BIT
ANY_DATE
ANY_INT
ANY_NUM
ANY_REAL
CD
CDT
CLK
CONFIGURATION
CU
CV
D(DATE)
D(Action Qualifier)
DS
DT
END_ACTION
END_CONFIGURATION
END_RESOURCE
END_STEP
END_STRUCT
END_TRANSITION
ET
EXPT
FROM
IN
INITIAL_STEP
Interval
L(Action Qualifier)
Lreal
Lword
N (Action Qualifier)
On
P(Action Qualifier)
Priority
PT

PV

Q(Parameter)

Q1

QD

QU

R(Action Qualifier)

R1

READ_ONLY

READ_WRITE

Release

Resource

RTC

S(Action Qualifier)

S1

SD

SEL

SEMA

Single

SL

STEP

Task

TOD

Transition

ULINT

USINT

VAR_ACCESS

WITH

Keywords (A..Z)

)" (Right-paranthesis-operator)

The right-parenthesis-operator executes an instruction, deferred by the left-parenthesis-modifier.

Example

```
LD  a
OR( b  (* Execution of instruction "OR" is deferred *)
```

```
AND c
)       (* "OR" will be executed now *)
OR( d
AND e
)
ST  f
```

Notes: This is an instruction in language Instruction List. It is defined by IEC61131-3

*_to_bool

0 is converted to false, everthing else to true.
The conversions String_to_bool and Real_to_bool are described in the respective sections.

ABS

Input

In: ANY_NUM

Returns

ANY_NUM

Notes: Returns the absolute value of the input.

Please note the following anomaly of the ABS function: The mathematical understanding of the ABS function is that it will never return a negative value. The signed integer data types in IEC61131-3 have a defined range of values which is asymmetric, for example, SINT from -128..+127. As defined by IEC61131-3, the ABS function will return the same data type that it is provided as an input; for example, when called with an SINT input, ABS will return an SINT output. The absolute value of -128 obviously is +128, but when passed to ABS for type SINT, exceeds the range of SINT and hence cannot be expressed. This overflow is, for performance reasons, silently ignored by ACR-View, the result returned being undefined. If you need to rely on the negative maximum value to be properly handled, use a data type with a wider range, or check inputs.

This does not apply to the ABS function as called by the Ladder Diagram Editor, this ABS function will signal overflow via the ENO output.

ACOS

Input

In: REAL

Returns

REAL: arcus cosine of input

ACTION

This keyword is defined by IEC61131-3 for the textual representation of programming language SFC. ACR-View does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

ADD

Inputs

In1: ANY_NUM

In2: ANY_NUM

Returns

ANY_NUM sum

Addition of two numbers. See Table E.1: Error conditions for result on overflow.

Notes: Standardization: this is an operation defined by IEC61131-3.

ADD (time)

Inputs

In1: TIME time duration value

In2: TIME

Returns

TIME Addition of the two time values provided

Addition of TIME values

Notes: Standardization: this is an operation defined by IEC61131-3.

AND

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT logical, bit by bit AND of Input 1 and Input 2

Notes: Standardization: this function is defined by IEC61131-3.

ANDN

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT logical, bitwise AND of Input 1 and negated Input 2

Notes: Standardization: this function is defined by IEC61131-3.

ANY

ANY_BIT is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: ANY_BIT, ANY_DATE, ANY_INT, ANY_REAL

ANY_BIT

ANY_BIT is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: BOOL, BYTE, WORD, DWORD, LWORD.

ANY_DATE

ANY_DATE is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: DATE, DATE_AND_TIME, TIME_OF_DAY.

ANY_INT

ANY_INT is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT.

ANY_NUM

ANY_NUM is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: ANY_INT, ANY_REAL.

ANY_REAL

ANY_REAL is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: REAL, LREAL.

ARRAY

ARRAY is the keyword to declare arrays of elements, see Derived Data Types

Examples

The following declares an array of five integers and assigns initial values:

```
VAR
x1: ARRAY[0..4] of INT := [1,2,3,4,5];
END_VAR
```

A three-dimensional array of 300 booleans:

```
VAR
x2: ARRAY[0..4, 15..20, 1..10] of BOOL;
END_VAR
```

An array of 100 structures:

```
TYPE
x3: STRUCT
member1: BOOL;
member2: INT;
    END_STRUCT;
END_TYPE
VAR
x4 : ARRAY[1..10,1..10] of x3;
END_VAR
```

Initializing of multidimensional arrays:

To initialize arrays with more than one dimension, give a list of list of initial values, each dimension enclosed in brackets. The dimension given first in declaration will correspond to the outermost brackets.

```
VAR
    x2: ARRAY[0..4, 1..2] of INT := [[1,2], [3,4], [5,6], [7,8], [9,10]];
    x3: ARRAY[0..1, 0..2, 0..3] of INT :=
[[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];
END_VAR
```

Note: ACR-View uses 16bit integers to represent array subscripts for performance reasons. Arrays should not be declared in a way to use subscripts exceeding 16bit address limits, as this would lead to undefined behavior.

ASIN

Input

In: REAL

Returns

REAL: arcus sine of input

Assignment

An Assignment will assign the result of an expression to a variable.

Example

```
VAR
    a: INT;
    b: ARRAY [0..5] OF INT;
    c: REAL;
    e: INT;
END_VAR
    a := 5;
    (* assign 5 to a *)
    b[1]:= a*2; e := a; (* two assignments *)
    e:= REAL_TO_INT( c );
    (* assignment with function call *)
The assignment instruction will evaluate the expression on the right side and
assign the resulting value to the variable given on the left.
```

Notes: This is a keyword only for language ST. This is defined by IEC61131-3.

AT

AT is the keyword to define the memory location where ACR-View should allocate memory for a given variable.
Very first input bit:

```
VAR
x1 at %ix0.0: bool;
END_VAR
```

Output word starting at second output byte:

```
VAR
x2 at %qw1.0: word;
END_VAR
```

ATAN

Input

In: REAL

Returns

REAL: arcus tangens of input

BOOL

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

Bool_to_*

Inputs

original data type bool

Returns

converted data type *

The function block converts the first value of type bool into the same value of type *.

The following data types can be converted:

DINT, INT and SINT

BYTE, DWORD, WORD and USINT, UINT, UDINT

true → 1

false → 0

REAL

true → 1.0

false → 0.0

STRING

true → 'true'

false → 'false'

BY

See FOR

BYTE

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

CAL

The program will be continued at the function block whose name is passed as operand. The unconditioned invocation may only be used as the end of a sequence and is not permitted within bracketing operations.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3. See also EN.

CALC

If the CR holds the value TRUE, the function block specified as operand will be called. If it holds the value 0, there is no invocation. The program flow continues with the instruction following the jump instruction.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

CALCN

If the CR holds the value FALSE, the function block specified as operand will be called. If it holds the value "1!, there is no invocation. The program flow continues with the instruction following the jump instruction.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

CASE

Though IF instructions may be nested, checking for one of many conditions can look quite compli cated using IF. CASE, instead, can check for more than one value with one instruction. The ´ex pression´ of the CASE-instruction is of type INT, and only the instruction will be executed that corresponds to this INT-value. After that the first instruction behind END_CASE will be executed.

IF the expression does not match any of the case-values, the first instruction (block) behind the ELSE will be executed. This partial instruction is optional.

```
CASE expression OF
   case_value1: { instructions; }
   case_value2: { instructions; }
```

...
case_valueN: { instructions; }
[**ELSE** instructions;]
END_CASE;

Example

```
VAR
    number : INT:= 10;
    amount : INT :=2;
END_VAR
CASE number OF
    10: amount := amount +1;
    11: amount := amount -1;
ELSE
    amount := number;
END_CASE;
```

In this example, the value of ´number´ will be determined, and if it is equal to 10, ´amount´ will be incremented, if it is equal to ´11´, ´amount´ will be decreased. In any other case, ´amount´ will be set to equal ´number´.

Notes: This is a keyword only for language ST. This is defined by IEC61131-3.

CD

This is the name of a formal parameter of a standard function block (CTD), and as such defined to be a keyword.

CDT

This is the name of a formal parameter of a standard function block (RTC), and as such defined to be a keyword.

CLK

This is the name of a formal parameter of a standard function block (R_TRIG), and as such defined to be a keyword.

CONCAT

Inputs

In1: STRING	First String
In2: STRING	Second String

Returns

STRING	Concatenation of both StringsPosition of first occurrence

Description

The character strings ´IN1´ and ´IN2´ in the working register are chained to form one character string which is loaded into the working register. The strings IN1 to IN2 are written from the left to the right in ascending order.

Configuration

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With ACR-View, these are defined and configured using property-dialog boxes . You will see this keyword in ACR-View only when printing the definition of a configuration.

CONSTANT

CONSTANT is the keyword to declare variables that should not be modified by the application code. The ACR-View compiler will give an error message if you intent to write to such a variable:

VAR CONSTANT x1 : INT := 15; END_VAR

See declaration sections.

COS

Input

In: REAL

Returns

REAL: Cosine of input

CR

CR is the abbreviation of Current Result, the virtual accumulator used in IEC61131-3 programming languages.

CTD

The function block "CTD" serves for counting down impulses received from the input operand "CD." On initialization, the counter will be set to "0".

If the operand "LOAD" is "1", the value received by the operand "PV" will be taken over as a value into the counter.

Each rising edge at the input "CD" will decrease the counter by "1".

The output operand "CV" contains the current value of the counter. If the counter value is positive, the output operand "Q" will have the boolean value "0". If the counter value reaches zero or becomes negative, the output "Q" will be set to "1".

Inputs

CD: bool	Counter pulse
LOAD: bool	Set initial value
PV: int	Reset value

Outputs

Q: bool	Signal when zero reached
CV: int	Counter value

Notes: Standardization—this function block is defined by IEC61131-3.

CTU

The function block "CTU" serves for counting up impulses received from the input operand "CU". On initialization, the counter will be set to "0".

The counter value will be reset if the operand "RESET" receives the value "1".

Each rising edge at the input "CU" will increase the counter by "1".

The output operand "CV" contains the current value of the counter. If the counter value is below the margin value "PV", the output operand "Q" will have the boolean value "0". If the counter value reaches or passes the margin, the output "Q" will be set to "1".

Inputs

CU: bool	COUnter pulse
RESET: bool	Reset counter
PV: int	Counter upper limit

Outputs

Q: bool	Signals if counter has reached upper limit
CV: int	Current counter value

Notes: Standardization—this function block is defined by IEC61131-3.

CTUD

The function block "CTUD" serves for counting up and down impulses. On initialization, the counter will be set to the value "0". Every rising edge at the input operand "CD". will increase the counter by "1", while every falling edge at the input "CD" will decrease it by "1".

If the operand "LOAD" is "1", the value received by the operand "PV" will be taken over as a value into the counter.

The counter value will be reset if the operand "RESET" receives the value "1". While the static state of the operand "RESET" remains unchanged, the counting conditions or the load condition will have no implication, independent of their value.

The output operand "CV" contains the current value of the counter. If the counter value is below the margin value "PV", the output operand "Q" will have the boolean value "0". If the counter value reaches or passes the margin, the output "Q" will be set to "1". If the counter value is positive, the output operand "QD" will have the boolean value "0". If the counter value reaches zero or becomes negative, the output "QD" will be set to "1".

Inputs

CU:bool	Counting impulses for counting up, rising edge
CD:bool	Counting impulses for counting down, rising edge
RESET: bool	Reset condition
LOAD: bool	Load condition
PV: int	Load value

Outputs

QU: bool	Signals whether counter state has reached PV
QD: bool	Signals whether counter state has reached "0"
CV: int	Counter state

Notes: Standardization—this function block is defined by IEC61131-3.

CU

This is the name of a formal parameter of a standard function block (CTU), and as such defined to be a keyword.

CV

This is the name of a formal parameter of a standard function block (CTD), and as such defined to be a keyword.

D(Date)

nD can be used as an abbreviation to DATE when specifying the data type of a literal constant. As data type DATE is not implemented in ACR-View, you will not be able to use this keyword with ACR-View.

D(Action Qualifier)

This is an Action qualifier, see Table 45 in the compliance statement. As ACR-View only supports actions of type N, you will not need to use this keyword with ACR-View.

DATE

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

DATE_AND_TIME

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

DELETE

Inputs

IN1: STRING	Basic character string in which a part should be deleted
L: UINT	Length of the substring which should be deleted
P: UINT	Starting position of substring

Returns

STRING	Shortened string

The function "DELETE"deletes a substring of length "L" starting at position "P" within the given string "IN1".

Notes: Standardization—this function is defined by IEC61131-3.

DINT

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

DIV

Inputs

In1: ANY_NUM	Value to be divided
In2: ANY_NUM	Value to divide by

Returns

ANY_NUM	Quotient

Divides two numbers. See Table E.1: Error conditions for result if divisor is zero.

Notes: Standardization—this is an operation defined by IEC61131-3.

DIV (time)

Inputs

In1: TIME	Time duration value
In2: ANY_NUM	Divisor

Returns

TIME	Divided time value
Division of TIME	Values

Notes: Standardization—this is an operation defined by IEC61131-3.

DO

See FOR and WHILE

DS

This is an Action qualifier, see Table 45 in the compliance statement. As ACR-View only supports actions of type N, you will not need to use this keyword with ACR-View.

DT

DT can be used as an abbreviation to DATE_AND_TIME when specifying the data type of a literal constant. As data type DATE_AND_TIME is not implemented in ACR-View, you will not be able to use this keyword with ACR-View.

DWORD

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

ELSE

See CASE and IF

ELSIF

See IF

EN

Function Blocks may have an input variable of type BOOL named EN. If this is the case, an invocation of an instance of this function block is performed if and only if the value of the input variable EN of that instance is TRUE.

See also CAL and ENO.

Notes:

5. "EN" is an abbreviation of "Enable."
6. If input and/or output variables are assigned in the same statement as the CAL instruction, these assignments are performed even if the CAL is not taken due to EN=FALSE.
7. By default, EN is TRUE

END_ACTION

This keyword is defined by IEC61131-3 for the textual representation of programming language SFC. ACR-View does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

END_CASE

See CASE

END_CONFIGURATION

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With ACR-View, these are defined and configured using property-dialog boxes. You will see this keyword in ACR-View only when printing the definition of a configuration.

END_FOR

See FOR

END_FUNCTION

See Function.

END_FUNCTION_BLOCK

See Function Block.

END_IF

See IF

END_PROGRAM

See PROGRAM

END_REPEAT

See REPEAT

END_RESOURCE

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With ACR-View, these are defined and configured using property-dialog boxes. You will see this keyword in ACR-View only when printing the definition of a configuration.

END_STEP

This keyword is defined by IEC61131-3 for the textual representation of programming language SFC. ACR-View does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

END_STRUCT

See STRUCT.

END_TRANSITION

This keyword is defined by IEC61131-3 for the textual representation of programming language SFC. ACR-View does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

END_TYPE

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC61131-3.

END_VAR

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC61131-3.

END_WHILE

See WHILE

ENO

Function Blocks may have an output variable of type BOOL named ENO. This typically is set to TRUE to signal correct execution and to FALSE to signal errors during execution. Typically, this ENO is wired to the EN input of another function block.

Notes: ENO" is abbreviated for Enable Output"

EQ

Inputs

IN1: ANY	Input 1
IN2: ANY	Input 2

Returns

BOOL	TRUE if Input 1 is equal to Input 2

Notes: Standardization—this function is defined by IEC61131-3.

ET

This is the name of a formal parameter of a standard function block (TOF), and as such defined to be a keyword.

ETRC

Generally an event task will be executed only once. Since the reaction on a special event can last longer than one cycle, it is necessary to restart the current task again. To perform this action the firmware function block ETRC (Event Task Run Control) can be used. It prolongs the execution of its own event task for another cycle. Additionally the function block provides at its outputs information like the cycle count or elapsed time since the first call on this the ETRC instance. With this information a reaction on errors, which would end up in an endless loop, could be handled.

Input:

IN : BOOLTRUE: The event task should be started for another cycle
FALSE: The event task should not be started again. The function block is called only to get the output information;

Output:

Q : BOOL TRUE: The event task will be executed for one cycle more

FALSE: the event task will be stopped after the current cycle

EVC : USINT The event code (EVC) describes the internal reason for the event task to be called.

ERT : TIME The elapsed runtime (ERT) returns the time since the first start of the current event task

CCV : UDINT The cycle counter value defines the count of event task cycles already executed

ERROR : USINT Return values of the ETRC execution.

0 : successful execution,

1 : execution not possible since function has been called out of a task (not a valid call)

Event codes of the function block:

Code	Description
0	The called task is unknown
1	Cold start executed
2	Warm start executed
3	Hot start executed
4	Single cycle start executed
5	PLC has been stopped by hardware RUN/STOP switch
6	PLC has been stopped by software stop
7	After executing a single cycle the PLC changes to status STOP
8	General error while PLC program execution
9	Division by zero
10	Invalid array index access
11	Error while executing a firmware function block

EXIT

Any of the loops can be ´left´ under program control before the loop condition dictates so. The EXIT instruction will jump to the first instruction after the innermost loop.

Example

```
VAR
    start: INT :=0;
    summe: INT :=0;
    ende : INT := 10;
END_VAR
FOR Start := 1 TO Ende BY 2 DO
    Summe := Summe + 1;
    IF Summe > 4 THEN
         EXIT;
```

```
    END_IF;
END_FOR;
(* Will continue here *)
```

As soon as ´Summe´ is greater than 4, the FOR loop will be left.

Notes: This is a keyword only for language ST. It is defined by IEC61131-3.

EXP

Input

In : REAL

Returns

REAL : e ** In

EXPT

Inputs :

In1 : ANY_REAL

In2 : ANY_NUM

Returns

ANY_REAL : In1 ** In2

F_EDGE

F_EDGE is used to indicate a falling edge detection function on Boolean inputs. This leads to an implicit declaration of a function block of type F_TRIG .

Example

```
FUNCTION_BLOCK AND_EDGE
    VAR_INPUT
        X : BOOL R_EDGE;
        Y : BOOL F_EDGE;
    END_VAR

    VAR_OUTPUT
        Z : BOOL ;
    END_VAR

    Z := X AND Y ; (* ST language example *)
END_FUNCTION_BLOCK
```

F_TRIG

Inputs

CLK: bool	Input operand whose falling edge is detected

Outputs

Q: bool	Output operand; indicates the falling edge of ´CLK´

The function block ´F_TRIG´ detects the status of the input operand ´CLK´. The status change from ´1´ to ´0´ in a processing cycle is detected and indicated in the subsequent cycle with the Boolean value ´1´ via the output ´Q´. The output is ´1´ only in the processing cycle in which the change of the status of ´CLK´ is detected and a falling edge is indicated.

Notes: Standardization—this function block is defined by IEC61131-3.

FALSE

Constant value of type BOOL.

FBD

FBD is the abbreviation of Function Block Diagram, one of the programming languages of IEC61131-3.

FIND

Find one character string within another character string.

Inputs

In1: StringBasic Character string in which a special character sequence is searched for; the string is made available via the working register

IN2: STRING Character sequence which is searched for in the ´IN1´ basic character string.

Returns

INT Position of first occurrence

A special character sequence is searched for in the ´IN1´ basic character string. If this string is found, the position of the first character of this sequence is entered into the working register or, otherwise, the value ´0´ is entered. If there are more than one in the basic character string, the string which was found first is entered.

Invocation of the FIND function in the program "search":

```
PROGRAM search
VAR
Basic_Text : STRING := 'StartupCondition';
Search_Text : STRING := 'Switch';
Position : INT;
END_VAR
LD  Basic_Text
FIND    Search_Text
ST  Position        (* Position: 4 *)
END_PROGRAM
```

Notes: Standardization—this function is defined by IEC61131-3.

FOR

With the FOR loop, a loop control variable will be set to a specified starting value, then incre mented (or decreased), and the loop will be terminated when a given end value is reached.

The syntax is:

FOR assignment **TO** Endvalue **BY** Increment **DO**
Instructions**;**
END_FOR;

Example

```
VAR
    Field : ARRAY[1..5] OF INT :=[2,14,8,12,5];
    Index : INT;
    MaxIndex : INT :=5;
    Maximum : INT :=0;
END_VAR
FOR Index :=1 TO MaxIndex BY 1 DO
    IF Field[Index] > Maximum THEN
         Maximum := Field[Index];
    END_IF;
END_FOR;
```

The loop control variable ´Index´ will start with ´1´, and will be incremented ´BY 1´ on each execution of the loop. This will be done until the end value ´MaxIndex´ (=5) will be reached.

Note: the BY-term is optional and can be omitted. Default then is to increment by 1.

Execution of the FOR-loop:
Initializing of the control variables.

Check of the termination criterion and termination if necessary.

Execution of the instruction block.

Increase/decrease of the control variable about the step size.

Go to step 2.

Notes: This is a keyword only for language ST. It is defined by IEC61131-3.

FROM

See Transition.

Function

IEC61131-3 defines three block types: PROGRAM, FUNCTION and FUNCTION BLOCK. See block types under "Advanced Topics" for more details.

Functions return values by assignment to a variable having the same name and type as the function, for example:

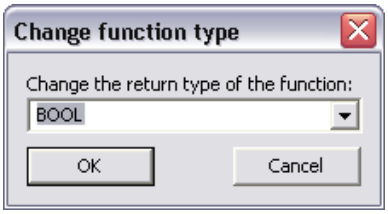
```
FUNCTION MyFun : INT
…
MyFun := 999;
END_FUNCTION
```

Notes:

- Some IEC61131 dialects take the current result at the END_FUNCTION or RETURN as the value to be returned by the function. ACR-View will ignore this value and only use the value assigned to the function name.
- The keywords FUNCTION and END_FUNCTION are typically invisible within ACR-View, as they are maintained by the Editors internally.

- The function return type (INT in the example shown above) is selected in the same dialog box where you specify the function name, at the very bottom. The default is BOOL.
- You can also enter user-defined data types (STRUCT's, ARRAY's, etc.) by entering the name of the data type manually into the input-field.
- To change a return type of a function, open the file in the project browser. Open the change return type dialog by selecting **Edit** > **Change Return Type...**.

The following dialog will pop up:



You can chose one of the given types or type in a user specific one.

FUNCTION BLOCK

IEC61131-3 defines three block types: PROGRAM, FUNCTION and FUNCTION_BLOCK. See block types under "Advanced Topics" for more details.

The keywords FUNCTION_BLOCK and END_FUNCTION_BLOCK are typically invisible within ACR-View, as they are maintained by the editors internally.

GE

Inputs

IN1: ANY	Input 1
IN2: ANY	Input 2

Returns

BOOL	TRUE if Input 1 is greater or equal than Input 2

Notes: Standardization—this function is defined by IEC61131-3.

GETSYSTEMDATEANDTIME

Inputs

EN: BOOL

Outputs

ENO: BOOL

ODT: DATE_AND_TIME

The function "GetSystemDateAndTime" returns the actual system time in ODT.

Notes: Standardization—this function block is *not* defined by IEC61131-3

GetTaskInfo

Output

Count: DWORD; (*number of cycles this task is executed *)

LastCT: TIME; (*time needed for last cycle*)

AverageCT: TIME; (*average time needed for execution*)

MinCT: TIME; (*minimum time needed for execution*)

MaxCT: TIME; (*maximum time needed for execution*)

State: DWORD; (*not yet used

GetTaskInfo returns information about the execution time of the last cycle of the current task. This function block has no input parameters.

GetTime

Input

IN1: TIME previous time

Returns

TIME: Time elapsed since power on, minus IN1

GETTIME will retrieve the time elapsed since the controller has last been switched on, less the time value supplied as an input. This can be used to easily measure time spans.

Example „Stop Watch

```
PROGRAM StopW
VAR
    begin, result : TIME;
END_VAR
start:
    LD t#0ms
    GETTIME
    ST begin
     ...

stop:
    LD begin
    GETTIME
    ST result
END_PROGRAM
```

GetVarData

InOut

VarName: STRING	Name of variable requested

Output

Q: bool	TRUE if VarInfo is valid
VarData: VarInfo	Information on variable

The variable specified as input is located within the memory address space and information on that variable is returned. If the variable cannot be located, Q is returned as FALSE.

Note that for ACR-View to be able to locate variables by name, a MAP file must be generated (resource options).

For the definition of VARINFO, see VARINFO under "keywords".

GetVarFlatAddress

InOut

VarName: STRING	Name of variable requested

Output

Q: bool	TRUE if VarInfo is valid
Address: DWORD	Flat memory address of specified variable

The variable specified as input is located within the memory address space and the address of its location is returned. If the variable cannot be located, Q is returned as FALSE.

Please note:

- For ACR-View to be able to locate variables by name, a MAP file has to be generated (resource options).
- The memory location returned must not be stored and used in another but the current execution cycle.

GT

Inputs

IN1: ANY	Input 1
IN2: ANY	Input 2

Returns

BOOL	TRUE if Input 1 is greater than or equal to Input 2

Notes: Standardization: this function is defined by IEC61131-3.

IF

The IF-instruction has following syntax:

IF expression **THEN** Block
{ **ELSIF** expression **THEN** Block}
[**ELSE** Block]
END_IF;

If the expression after IF evaluates to ´true´, the instructions given after THEN will be executed. If the expression after IF evaluates to ´false´, the instructions after ELSE will be executed or the ELSEIF-condition will be checked. In any case, execution will then continue with the next instruc tion after END_IF.

Note: It is recommanded to use the absolute value ABS() of a floating point number if a comparision with 0.0 is to be done since –0.0 == 0.0 will not return true.

The following IF instruction will compute the maximum of two numbers:

```
IF a>b THEN
    maximum := a;
    ELSE
    maximum := b;
END_IF;
```

IF instructions may be nested, i.e. the THEN-part as well as the ELSE-part may contain other IF instructions.

Example
The following program will again compute the maximum of two numbers, but if this maximum is ´a´ and ´a´ is greater than 10, it will be reduced by 1:

```
VAR
    a: INT :=12;
    b: INT :=5;
    maximum: INT;
END_VAR
IF a>b THEN
    maximum :=a;
    IF (a>10) THEN
    a:=a-1;
    ELSE
    a:=a+1;
    END_IF;
ELSE
    maximum :=b;
END_IF;
```

Notes: This is a keyword only for language ST. It is defined by IEC61131-3.

IL

IL is the abbreviation of Instruction List, one of the programming languages of IEC61131-3.

IN

This is the name of a formal parameter of a standard function block (TOF), and as such defined to be a keyword.

INITIAL_STEP

This keyword is defined by IEC61131-3 for the textual representation of programming language SFC. ACR-View does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

INSERT

Inputs

IN1: STRING	Character string
IN2: STRING	Charcter string to be inserted
P: UINT	Starting position

Returns

STRING	Composed string

The ´INSERT´ function inserts the string ´IN2´ into ´IN1´. The concatenated string consists of the first ´P-1´ charcters of ´IN1´, the completet string ´IN2´and the rest of ´IN1´.

Notes: Standardization—this function is defined by IEC61131-3.

INT

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

Interval

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With ACR-View, these are defined and configured using property-dialog boxes. You will see this keyword in ACR-View only when printing the definition of a configuration.

JMP

The program flow continues at the position specified by the jump target. The jump target must be a sequence start uniquely identified by a label. A jump is possible only within a POU.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

JMPC

If the CR holds the value TRUE, the program flow continues at the position specified by the jump target. If it holds the value 0, there is no jump. The program flow continues with the instruction following the jump instruction.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

JMPCN

If the CR holds the value FALSE, the program flow continues at the position specified by the jump target. If it holds the value 1, there is no jump. The program flow continues with the instruction following the jump instruction.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

L*(Action Qualifier)*

This is an Action qualifier, see Table 45 in the compliance statement. As ACR-View only supports actions of type N, you will not need to use this keyword with ACR-View.

LD

The value of the operand is evaluated and loaded into the current result. This overwrites data stored in CR. The operand is not modi fied. The data

type of the operand determines the permissible data type for consecutive operands.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

LD (Ladder Diagram)

LD is the abbreviation of Ladder Diagram, one of the programming languages of IEC61131-3.

LDN

The operand is evaluated, and the current result is loaded with the negated value. The operand is not modified. The data type of the operand determines the permissible data type for consecutive operands.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

LEFT

Inputs

In: STRING	character string
L: UINT	Number of characters to retrieve

Returns

STRING	the 'L' leftmost characters of IN

The ´LEFT´ function enters the left part of the currently loaded character string into the working register. The input operand ´L´ defines the number of characters to be entered.

LE

Inputs

IN1: ANY	Input 1
IN2: ANY	Input 2

Returns

BOOL	TRUE if Input 1 is less or equal than Input 2

Notes: Standardization—this function is defined by IEC61131-3.

LEN

Inputs

In: STRING	character string

Returns

INT	length of IN

The function ´LEN´ determines the length of the character string in the working register (input operand of data type ´STRING´) and enters the determined value as INT number into the working register.

LIMIT

Inputs

MN: Any_Num	lower limit
IN: Any_Num	Test value
MX: Any_Num	Upper Limit

Returns

Any_Num	One of the input values, see description

The ´MN´ and ´MX´ values define the lowest and highest limit value. The function compares the test value ´IN´ with ´MN´ and ´MX´. If ´IN´ is between the two limit values, it is loaded into the working register. If ´IN´ is smaller than ´MN´, the ´MN´ value is output. If ´IN´ is greater than ´MX´, the ´MX´ value is loaded.

Notes: Standardization—this function is defined by IEC61131-3.

LINT

This is the name of an elementary data type, which is defined by IEC61131-3, but not supported by ACR-View. See Table 10 in the compliance statement.

LN

Input

In: REAL

Returns

REAL:	logarithm to the base of e

LOG

Input

In: REAL

Returns

REAL:	logarithm to the base of 10

LREAL

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

LT

Inputs

IN1: ANY I	Input 1
IN2: ANY	Input 2

Returns

BOOL	TRUE if Input 1 is less than Input 2

Notes: Standardization—this function is defined by IEC61131-3.

Lword

This is the name of an elementary data type, which is defined by IEC61131-3, but not supported by ACR-View. See Table 10 in the compliance statement.

MUX

ACR-View does not implement the MUX function.

Notes: Standardization—this function is defined by IEC61131-3.

MAX

Inputs

In1: Any_Num	Input Value1
In2: Any_Num	Input Value2
...	
InN: Any_Num	Input ValueN

Returns

Any_Num	Maximum of all input values

The ´MAX´ function determines which input operand has the highest value. The selected operand is loaded into the working register.

Notes: Standardization—this function is defined by IEC61131-3.

MID

Inputs

In: STRING	Character string
L: UINT	Number of characters to retrieve
P: UINT	Starting position

Returns

STRING	The next "L" characters of IN, starting at the P-th character

The ´MID´ function enters a middle part of the currently loaded character string into the working register. The input operand ´P´ defines the first character to be entered, ´L´ defines the number of characters to be entered

Notes: Standardization—this function is defined by IEC61131-3.

MIN

Inputs

In1: Any_Num Input Value1

In2: Any_Num Input Value2

...

InN: Any_Num Input ValueN

Returns

Any_Num Minimum of all input values

The ´MIN´ function determines which input operand has the smallest value. The selected operand is loaded into the working register.

Notes: Standardization—this function is defined by IEC61131-3.

MOD

Input

In1: ANY_INT

In2: ANY_INT

Returns

ANY_INT

The first input will be divided by the second input. MOD delivers the residue to current result.

MOVE

Inputs

In: ANY

Outputs

Out: ANY

The function "MOVE" is an arithmetic function that serves for assigning a value.

MUL

Inputs

In1: ANY_NUM Value to be multiplied

In2: ANY_NUM Value to multiply with

Returns

ANY_NUM product

Multiplies two numbers. See Table E.1: Error conditions for result on overflow.

Notes: Standardization—this is an operation defined by IEC61131-3.

MUL (time)

Inputs

In1: TIME time duration value

In2: ANY_NUM multiplicand

Returns

TIME multiplied time value

Multiplication of TIME values

Notes: Standardization—this is an operation defined by IEC61131-3.

N (Action Qualifier)

This is an Action qualifier, see Table 45 in the compliance statement. As ACR-View only supports actions of type N, you will not need to use this keyword with ACR-View.

NCC

NCC is an acronym for native code compiler.

NE

Inputs

IN1: ANY Input 1

IN2: ANY Input 2

Returns

BOOL TRUE if Input 1 is not equal to Input 2

Notes: Standardization—this function is defined by IEC61131-3.

NEG

Input

In: ANY_NUM

Returns

ANY_NUM: negated numeric value of input

NOT

Inputs

IN1: ANYBIT Input

Returns

ANYBIT logical negation (1-complement) of Input

Notes: Standardization—this function is defined by IEC61131-3.

OF

See CASE

On

See RESOURCE.

OPC

The var qualifier OPC allows a user, to mark dedicated variables, to become part of the variable table, already within the declaration editor of ACR-View.

See Declaration Sections

OR

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT logical, bit by bit OR of Input 1 and Input 2

Notes: Standardization—this function is defined by IEC61131-3.

ORN

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT Logical, bitwise OR of Input 1 and negated Input 2

Notes: Standardization—this function is defined by IEC61131-3.

P(Action Qualifier)

This is an Action qualifier, see Table 45 in the compliance statement. As ACR-View only supports actions of type N, you will not need to use this keyword with ACR-View.

POINTER

The datatype pointer is defined by ACR-View in addition to IEC61131-3. Using this datatype, it is now possible to call Functions or Functionblocks with arrays of different sizes. A pointer must be declared as follows:

```
VAR
    IntVar : INT;
pInt : POINTER;
END_VAR
```

To access the adress of a variable, the adress operator ("&") must be written in front of the variable's name.

Example IL: LD &IntVar

Example ST: pInt := &IntVar;

POU

POU is the abbreviation of Program Organization Unit, meaning a Program, Function or Function Block written in one of the programming languages of IEC61131-3.

Priority

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With ACR-View, these are defined and configured using property-dialog boxes. You will see this keyword in ACR-View only when printing the definition of a configuration.

PROGRAM

IEC61131-3 defines three block types: PROGRAM, FUNCTION and FUNCTION BLOCK. See block types under "Advanced Topics" for more details.

The keywords PROGRAM and END_PROGRAM are typically invisible within ACR-View, as they are maintained by the editors internally.

PT

This is the name of a formal parameter of a standard function block (TOF), and as such defined to be a keyword.

PV

This is the name of a formal parameter of a standard function block (CTD), and as such defined to be a keyword.

Q(Parameter)

This is the name of a formal parameter of a standard function block (CTD), and as such defined to be a keyword.

Q1

This is the name of a formal parameter of a standard function block, and as such defined to be a keyword.

QD

This is the name of a formal parameter of a standard function block (CTUD), and as such defined to be a keyword.

QU

This is the name of a formal parameter of a standard function block (CTUD), and as such defined to be a keyword.

R(Action Qualifier)

This is an Action qualifier, see Table 45 in the compliance statement. As ACR-View only supports actions of type N, you will not need to use this keyword with ACR-View.

R(eset)

The operand is reset, if the content of the CR equals 1. If this precondition is not met, operands will not be changed. The CR is not modified.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

R_EDGE

R_EDGE is used to indicate a rising edge detection function on Boolean inputs. This leads to an implicit declaration of a function block of type R_TRIG.

Example

```
FUNCTION_BLOCK AND_EDGE
VAR_INPUT
X : BOOL R_EDGE;
Y : BOOL F_EDGE;
END_VAR

VAR_OUTPUT
Z : BOOL ;
END_VAR

Z := X AND Y ; (* ST language example *)

END_FUNCTION_BLOCK
```

R_TRIG

Inputs

CLK: bool	Input operand whose rising edge is detected

Outputs

Q: bool	Output operand; indicates the rising edge of ´CLK´

The function block ´R_TRIG´ detects the status of the input operand ´CLK´. The status change from ´0´ to ´1´ in a processing cycle is detected and indicated with the Boolean value ´1´ via the output ´Q´. The output is ´1´ only in the processing cycle in which the change of the status of ´CLK´ is detected and a rising edge is indicated.

Notes: Standardization—this function block is defined by IEC61131-3.

R1

This is the name of a formal parameter of a standard function block, and as such defined to be a keyword.

READ_ONLY

This keyword is defined by IEC61131-3 for the definition of Access Paths. ACR-View does not support Access Paths, hence you will not be able to use this keyword with ACR-View.

READ_WRITE

This keyword is defined by IEC61131-3 for the definition of Access Paths. ACR-View does not support Access Paths, hence you will not be able to use this keyword with ACR-View.

REAL

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

Real_to_*

Inputs

original data type real

Returns

converted data type *

The function block converts the first value of type real into the same value of type *.

The following data types can be converted:

BOOL

Values within the interval ±1,175494351e-38 are cast to false all other values to true.

Examples
1.1 → true

-22.33 → true

1.1e-39 → false

DINT, INT and SINT

Values are rounded off, therefore values smaller than x.5 are rounded to the absolute smaller number else to the next larger one.

Examples
0.3 → 0

-0.6 → -1

-1.5 → -2

BYTE, DWORD, WORD and USINT, UINT, UDINT

The conversion is analog to an integer-conversion for positive values Negative values are cast to the new size and the generated bit pattern is interpreted as a positive number

Examples
-1.6 → 254 (USINT), 65534 (UINT), 4294967294 (UDINT); (A sint -2 has the bit pattern: 1111 1110 which is interpreted as 254)

33.3 → 33

STRING

For converting string function Sprintf(str, %#g", value); is used.

Examples
0.0 → '0.000000'

123.45678 → ' 123.456'

-12.345678 → ' –12.3456'

12345678.9 → ' 1.23457e+007'

0.000000123 → ' 1.23000e-007'

Release

This is the name of a formal parameter of a standard function block (SEMA), and as such defined to be a keyword.

REPEAT

In contrast to the other loop types, REPEAT will check the loop expression after execution of the loop. The syntax is:

REPEAT
    instructions;
UNTIL expression
END_REPEAT;

So, the REPEAT loop will always be executed at least once.

Example

```
VAR
    i : INT := -1;
END_VAR
REPEAT
    i:=i-1;
UNTIL  i < 0
END_REPEAT;
(* now, i = -2 *)
```

Although ´i´ will meet the loop condition from the beginning, the REPEAT loop will be executed once anyway.

Notes: This is a keyword only for language ST. This is defined by IEC61131-3.

REPLACE

Inputs

IN1: STRING	Basic character string in which a part should be replaced
IN2: STRING	New character string
L: UINT "IN1"	Length of the substring which should be cut out off
P: UINT	Starting position of the inserted string

Returns

STRING	New composited

The function "REPLACE" replaces a substring of length "L" starting at position "P" within the given string "IN1" by the string "IN2".

Notes: Standardization: this function is defined by IEC61131-3.

Resource

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With ACR-View, these are defined and configured using property-dialog boxes. You will see this keyword in ACR-View only when printing the definition of a configuration.

RET

The RET instruction causes an unconditioned return jump to the calling POU – if this POU is the program POU, a return jump to the system program. When jumping back, the calling POU is resumed at the point of interruption. Delayed operations will be executed.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

RETAIN

RETAIN is the keyword to declare variables as retentive, and is optional after VAR, VAR_GLOBAL. Implementation of retentiveness depends on your controller. See declaration sections.

RETC

Conditional Return

Instruction does not take any operands.

If the CR holds the value 1, a return jump to the calling POU is performed – i.e. to the system program if calling POU is of type program. If the CR holds the value 0, there is no return jump. The program flow continues with the instruction following the jump instruction.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

RETCN

Conditional Return

Instruction does not take any operands.

Conditioned return jump depending on the Boolean content of the CR.

If the CR holds the value 0, a return jump to the calling POU is performed – i.e. to the system program if calling POU is of type program. If the CR holds the value 1, there is no return jump. The program flow continues with the instruction following the jump instruction.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

RETURN

The RETURN instruction will cause the current POU to be left, transferring control back to the caller of the current POU. Note that on working with functions, the function value (variable with the name of the function) must be assigned. If output values of function blocks aren't assigned by local values of the function block, they have the predefined values of their data types.

Example

```
IF a<b THEN
    RETURN;
END_IF;
```

Notes: This is a keyword only for language ST. This is defined by IEC61131-3.

RIGHT

Inputs

In: STRING	character string
L: UINT	Number of characters to retrieve

Returns

STRING	the "L" rightmost characters of IN

The ´RIGHT´ function enters the right part of the currently loaded character string into the working register. The input operand ´L´ defines the number of characters to be entered.

ROL

Inputs

IN: ANY_BIT	Bit Pattern
N: UINT	Number of bits to shift

Returns

ANY_BIT	IN, rotated left N bits

The leftmost bits will be rotated in from right

Notes: Standardization: this function is defined by IEC61131-3.

ROR

Inputs

IN: ANY_BIT	Bit Pattern
N: UINT	Number of bits to shift

Returns

ANY_BIT	IN, rotated right N bits

The rightmost bits will be rotated in from left.

Notes: Standardization: this function is defined by IEC61131-3.

RS

Inputs

Set: bool Set condition

Reset1: bool Reset condition

Outputs

Q1: bool Output state of the bistable element

The characteristic feature of the ´RS´ function module is to statically set a data element - the output Q1 - to the Boolean status ´1´ or ´0´. Depending on the Boolean input operands ´Set1´ and ´ReSet1´ it is changed between the two states.

The output ´Q1´ is initialized with the value ´0´ when starting the process. The first processing of the function block with the value ´1´ of the operand ´Set´ causes the output ´Q1´ to be set to ´1´. A change of the value of ´Set´ no longer then effects the output ´Q1´. The value ´1´ of the input operand ´ReSet1´ sets the output ´Q1´ to ´0´ - the output is reset.

If both input operands have the value ´1´, the fulfilled set condition is dominant, i.e. Q1 is reset with priority.

Notes: Standardization—this function block is defined by IEC61131-3.

RTC

The RTC funtion block sets the output CDT to the inpu PDT if EN=1. Otherwise CDT is unvalid

Inputs:

EN: BOOL

PDT: DATE_AND_TIME Present date and time

Outputs

Q: BOOL copy of EN

CDT: DATE_AND_TIME Current date and time, valid when EN=1

Notes: Standardization—this function block is defined by IEC61131-3

S(Action Qualifier)

This is an Action qualifier, see Table 45 in the compliance statement. As ACR-View only supports actions of type N, you will not need to use this keyword with ACR-View.

S(et)

The operand is set, if the content of the CR equals 1. If this precondition is not met, operands will not be changed. The CR is not modified.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

S1

This is the name of a formal parameter of a standard function block, and as such defined to be a keyword.

SD

This is an Action qualifier, see Table 45 in the compliance statement. As ACR-View only supports actions of type N, you will not need to use this keyword with ACR-View.

SEL

This is the name of a standard function block, which is defined in IEC61131-3, but not provided by ACR-View. See Table 31 in the compliance statement.

SEMA

This is the name of a standard function block, which is defined in IEC61131-3, but not provided by ACR-View. See Table 34 in the compliance statement.

SETSYSTEMDATEANDTIME

Inputs

EN: BOOL

IDT: DATE_AND_TIME

Outputs

ENO: BOOL

The function "SetSystemDateAndTime" sets the actual system time in IDT.

Notes: Standardization: this function block is *not* defined by IEC61131-3.

SFC

SFC is the abbreviation of Sequential Function Chart, one of the programming languages of IEC61131-3.

SHL

Inputs

IN: ANY_BIT Bit Pattern

N: UINT Number of bits to shift

Returns

ANY_BIT IN, shifted left N bits

Rightmost bits will be filled with zeros

Notes: Standardization: this function is defined by IEC61131-3.

SHR

Inputs

IN: ANY_BIT Bit Pattern

N: UINT Number of bits to shift

Returns

ANY_BIT IN, shifted right N bits

Leftmost bits will be filled with zeros

Notes: Standardization: this function is defined by IEC61131-3

Signed_to_Unsigned

Positive values stay untouched. The most significant bits are cut, if the converted variable is smaller than the original one.

The bit pattern of negative values is interpreted as a positive integer.

Note: The value is first converted to the new size then to an unsigned integer

Examples
(sint → uint)

3 (0000 0011) → 3 (0000 0000 0000 0011)

3 (1111 1101) → 65534 (1111 1111 1111 1101)

SIN

Input

In: REAL

Returns

REAL: sine of input

Single

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With ACR-View, these are defined and configured using property-dialog boxes. You will see this keyword in ACR-View only when printing the definition of a configuration.

SINT

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

SL

This is an Action qualifier, see Table 45 in the compliance statement. As ACR-View only supports actions of type N, you will not need to use this keyword with ACR-View.

SQRT

Input

In: REAL

Returns

REAL: square root of input

SQRT will compute the square root of the input

SR

Inputs

Set1: bool Set condition

Reset: bool Reset condition

Outputs

Q1: bool Output state of the bistable element

The characteristic feature of the ´SR´ function module is to statically set a data element - the output ´Q1´ - to the Boolean status ´1´ or ´0´.

Depending on the Boolean input operands ´Set1´ and ´ReSet´ it is changed between the two states.

The output ´Q1´ is initialized with the value ´0´ when starting the process. The first processing of the function block with the value ´1´ of the operand ´Set1´ causes the output ´Q1´ to be set to ´1´. A change of the value of ´Set1´ no longer then effects the output ´Q1´. The value ´1´ at the input operand ´ReSet´ sets the output ´Q´ to ´0´ - the output is reset.

Notes: Standardization—this function block is defined by IEC61131-3.

ST

The content of the CR register is assigned to the operand. This overwrites the value of the operand. The data type of the operand must match the data type of the data element in the register. The data type of the CR is determined by the data type of the variable first assigned a value. Further assignments will then be possible only if the types of further variables match. An assignment may be followed by another assignment.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3,

ST (Structured Text)

ST is the abbreviation Structured Text, one of the programming languages of IEC61131-3.

STEP

This keyword is defined by IEC61131-3 for the textual representation of programming language SFC. ACR-View does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

STN

The negated content of the CR register is assigned to the operand. This overwrites the value of the operand. The data type of the operand must match the data type of the data element in the register. The CR register is not modified by this operation. An assignment STN may be followed by another ST or STN instruction.

Notes: This is a keyword in language Instruction List. This is defined by IEC61131-3.

STRING

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

String_to_*

Inputs

original data type string

Returns

converted data type *

The function block converts the first value of type string into the same value of type *.

The following data types can be converted:

BOOL

The strings '1' and 'true' are converted to true, the rest to false.

DINT, INT and **SINT**

The string is read from left to right until an illegal charcter or the word is finished.

Examples
'-1' → -1

'213hallo' → 213

'23.5' → 23

BYTE, DWORD, WORD and **USINT, UINT, UDINT**

The conversion is analog to an integer-conversion for positive values

Negative values are cast to the new size and the generated bit pattern is interpreted as a positive number

Examples
'-1.6' → 254 (USINT), 65534 (UINT), 4294967294 (UDINT); (A sint -2 has the bit pattern: 1111 1110 which is interpreted as 254)

'33.3' → 33

REAL

Analog the above conversion. The e-Notation is permitted

Examples
'-123.456' → -123.456

'0.23' → 0.23

'-1.2e-2' → ' –0.012

STRUCT

STRUCT is the keyword to define structured data types, see and Derived Data Types

A variable consisting of two members:

```
VAR
x1: STRUCT
x2: INT;
x3: BOOL;
END_STRUCT;
END_VAR
```

A variable of user defined type:

```
TYPE
x4: STRUCT
x5: REAL;
x6 : BOOL;
END_STRUCT;
END_TYPE
VAR
x7: x4;
END_VAR
```

SUB

Inputs

In1: ANY_NUM

In2: ANY_NUM

Returns

ANY_NUM Difference In1-In2

Subtraction of two numbers.

Notes: Standardization: this is an operation defined by IEC61131-3.

SUB *(time)*

Inputs

In1: TIME time duration value

In2: TIME

Returns

TIME difference between the two time values provided

Subtraction of TIME values

Notes: Standardization: this is an operation defined by IEC61131-3.

TAN

Input

In: REAL

Returns

REAL: tangent of input

Task

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With ACR-View, these are defined and configured using property-dialog boxes. You will see this keyword in ACR-View only when printing the definition of a configuration.

THEN

See IF

TIME

See Elementary Data Types

See also Constants on how to create TIME-constants.

Notes: Standardization—this is a data type defined by IEC61131-3.

TIME_OF_DAY

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

TIME_TO_*

Inputs

original data type time

Returns

converted data type *

The function block onverts the first value of type time into the same value of type *.

The following data types can be converted:

BOOL

BYTE

DINT

DWORD

INT

REAL

SINT

STRING

UDINT

UINT

USINT

WORD

Notes: Standardization: this function is defined by IEC61131-3. Except TIME_TO_DINT and TIME_TO_REAL, all TIME convert functions are only available within the Ladder-Diagram-Editor.

TO

See FOR

TOD

TOD can be used as an abbreviation to TIME_OF_DAY when specifying the data type of a literal constant. As data type TIME_OF_DAY is not implemented in ACR-View, you will not be able to use this keyword with ACR-View.

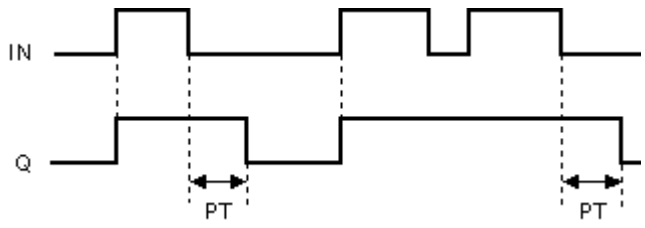
TOF

If the state of the input operand "IN" is "1", this will be passed to the output operand "Q" without any delay. If there is a falling edge, a timer function will be started lasting as long an interval as specified by the operand "PT"

It is after the time is up that the operand "Q" will change to the state "0". If the "PT" value changes after the start, it will have no implications until there is the next rising edge of the operand "IN".

The operand "ET" contains the current timer value. If the time is up, the operand "ET" will keep its value as long as the operand "IN" has the value "0". If the state of the "IN" operand changes to "1", the value of "ET" will switch to "0".

If the input "IN" is switched off, this will switch off the output "Q" after an interval specified by the delay value.



Inputs:

IN:Start condition

PT: time Initial time value

Outputs

Q: bool binary state of the timer

ET: time current time value

Notes: Standardization—this function block is defined by IEC61131-3

TON

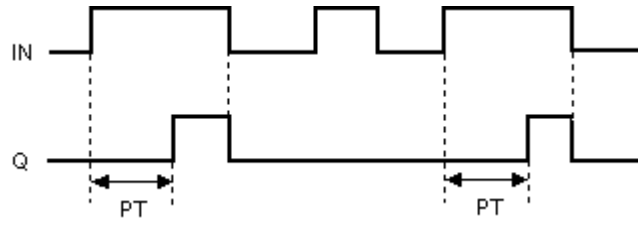
The rising edge of the input operand "IN" will start the timer "TON", and it will run as long a time interval as specified by the operand "PT".

While the timer is running, the output operand "Q" will have the value "0". If the time is up, the state will change to "1" and keep this value until the operand "IN" changes to "0".

If the "PT" value changes after the timer has been started, this will have no implications until the next rising edge of the operand "IN".

The output operand "ET" contains the current timer value. If the time is up, the operand "ET" will keep its value as long as the operand "IN" has the value "1". If the state of the "IN" operand changes to "0", the value of "ET" will switch to "0".

If the input "IN" is switched on, this will switch on the output "Q" after an interval specified by the delay value.



Inputs:

IN:Start condition

PT: time Initial time value

Outputs

Q: bool binary state of the timer

ET: time current time value

Notes: Standardization—this function block is defined by IEC61131-3.

TP

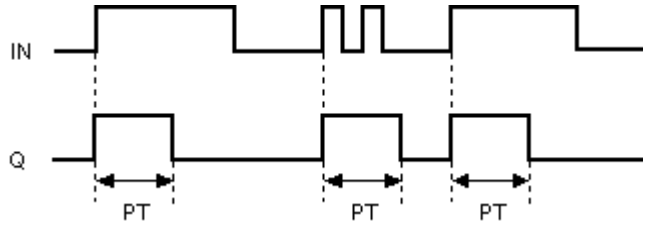
A rising edge of the input operand "IN" will start the timing function of the timer "TP", and it will run as long an interval as specified by the operand "PT".

While the timer is running, the output operand "Q" will have the state "1". Any changes of state at the input "IN" will have no implication on the procedure.

If the "PT" value changes after the start, this will not have any implication before the next rising edge of the "IN" operand.

The output operand "ET" contains the current timer value. If the operand "IN" has the state "1" after the time is up, the operand "ET" will keep its value.

Every edge occurring while the timer is not running will cause an impulse at the output Q that lasts as long as specified.



Inputs

IN: bool start timer

PT: time initial time value

Outputs

Q: bool binary state of timer

ET: time elapsed time

Notes: Standardization—this function block is defined by IEC61131-3.

Transition

This keyword is defined by IEC61131-3 for the textual representation of programming language SFC. ACR-View does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

TRUE

Constant value of type BOOL.

TRUNC

Inputs

In: REAL

Returns

ANY_INT

Returns the integer part of the supplied real value.

Notes: Standardization—this function is defined by IEC61131-3.

TYPE

See Declaration Sections and Derived Data Types

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC61131-3.

Keywords TYPE .. END_TYPE should not be nested within a VAR..END_VAR block, but rather be on top level in the declaration section, or in a type declaration file on project level.

UDINT

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

UINT

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

ULINT

This is the name of an elementary data type, which is defined by IEC61131-3, but not supported by ACR-View. See Table 10 in the compliance statement.

UNTIL

See REPEAT

USINT

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3

VAR

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC61131-3.

VAR_ACCESS

This keyword is defined by IEC61131-3 for the definition of Access Paths. ACR-View does not support Access Paths, hence you will not be able to use this keyword with ACR-View.

VAR_INPUT

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC61131-3.

VAR_OUTPUT

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC61131-3.

VAR_IN_OUT

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC61131-3.

VAR_GLOBAL

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC61131-3.

VAR_EXTERNAL

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC61131-3.

VARINFO

VARINFO is defined as

```
VARINFO : Struct
    TYP   : UINT;
    SIZE  : UINT;
    PROG  : UINT;
    SEG   : UINT;
    OFFSET:UINT;
    BIT: UINT;
    SCOPE: UINT;
end_struct;
```

WHILE

The WHILE loop will execute the loop body as long as the given expression evaluates to ´true´. Syntax:

WHILE expression **DO**
  instructions;
END_WHILE;

The expression given after the keyword **WHILE** will be evaluated before entering the loop. If it is true, the loop body will be executed. This will terminate only when the expression evaluates to ´false´.

Example

```
VAR
    i : INT := 3;
END_VAR
WHILE  i > 0  DO
    i:=i-1;
END_WHILE;
```

Initially, ´i´ equals 3. 3 is greater than 0, so the expression after WHILE is true and the loop body executed. This will decrement the value of ´i´ to 2. 2 is still greater than 0, so the loop body will be executed again. Some time later, the loop body will decrement ´i´ from 1 to 0. On the next check, the expression after WHILE will be false, hence the loop body will not be executed again.

Notes: This is a keyword only for language ST. This is defined by IEC61131-3.

WITH

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With ACR-View, these are defined and configured using property-dialog boxes. You will see this keyword in ACR-View only when printing the definition of a configuration.

WORD

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

WSTRING

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC61131-3.

XOR

Inputs

IN1:	ANY_BIT	Input 1
IN2:	ANY_BIT	Input 2

Returns

ANY_BIT	logical, bitwise XOR of Input 1 and Input 2

Notes: Standardization: this function is defined by IEC61131-3.

XORN

Inputs

IN1:	ANY_BIT	Input 1
IN2:	ANY_BIT	Input 2

Returns

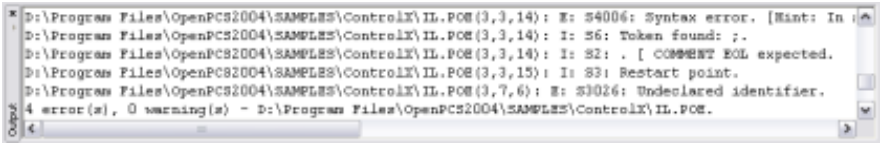
ANY_BIT	logical, bitwise XOR of Input 1 and inverted Input 2

Notes: Standardization: this function is defined by IEC61131-3.

Errors and Warnings

How to Read Error Messages

In the Output Window you will find any error messages from the compiler.



Each error message line fits the following style:

- The file name including path of the source code that caused the error message.
- A triple of numbers where the first number indicates the section the error occurred ("2" for "Declaration" and "3" for "Instruction"), the second is the line and the last the column (within the section mentioned before).
- A capital letter indicates the type of message:

letter	stands for
I	Info
E	Error
W	Warning
F	Fatal Error

- The error number code that allows you to find a detailed error description here in the documentation.
- A short description of the error.

General Errors

G10001

Warning G10001: The file [file name] is inconsistent. You should not use it.
The File is inconsistent. A reason might be that the file name is different from the POU name within the file. This is normally caused by renaming files outside of ACR-View. POUs should always be renamed by using the ACR-View function **File->File->Rename**.

Syntax Errors

S1000

Nested comments are not allowed.

You are using an IEC61131-3 compatible version. In this version nested comments are not allowed.

S1001

Invalid character.

An unsupported character was used. See also Table 1: Character set features

S1002

End of file found in comment.

The end of the file was reached before an open comment has been closed. Please close the comment before calling the syntax check.

S1003

Reserved keyword.

A reserved keyword was used an identifier.

S1004

Invalid value for hour.

The numeric value for the hour unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be an integer in the range [0, 23].

S1005

Invalid value for minute.

The numeric value for the minute unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be an integer in the range [0, 59].

S1006

Invalid value for second.

The numeric value for the seconds unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be a fixed point number in the range [0, 60).

S1008

Invalid value for month.

The numeric value for the month unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be an integer in the range [1, 12].

S1009

Invalid day range.

The numeric value for the day unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be an integer in the range [1, 31], giving the day of the month. I. e. if the respective month has less than 31 days, the maximum number of days in the month is the greatest valid value for the day literal.

S1010

Exponent too large.

The numeric value for the exponent of a real literal must be an integer in the range [–37, 38] and for a LREAL literal an INT in the range [–307, 308].

S1011

Incorrect direct address.

The numeric value for a location field in the hierarchical address of a directly represented variable is hardware dependent integer, but must not exceed 4294967295. Please consult your hardware documentation to determine the maximum value for each field in the address hierarchy.

S1012

Invalid day entry.

The numeric value for the day unit of a TIME literal must be a fixed point number in the range [0, 255].

S1013

Invalid hour entry.

The numeric value for the hour unit of a TIME literal must be a fixed point number in the range [0, 24] if the hour is not the most significant unit of the duration literal. An overflow is only permitted if the hour unit is the most significant unit of the TIME literal.

Example

T#25h_15m is permitted.

T#1d_25h_15m is not allowed. The correct representation of this duration literal is: T#2d_1h_15m.

S1014

Invalid minutes entry.

The numeric value for the minute unit of a TIME literal must be a fixed point number in the range [0, 60] if minute is not the most significant unit of the duration literal. An overflow is only permitted if the minute unit is the most significant unit of the TIME literal.

Example

T#75m is permitted.

T#5h_75m is not allowed. The correct representation of this duration literal is: T#6h_15m.

S1015

Invalid seconds entry.

The numeric value for the seconds unit of a TIME literal must be a fixed point number in the range [0, 60] if seconds are not the most significant unit of the duration literal. An overflow is only permitted if the seconds unit is the most significant unit of the TIME literal.

Example

T#75s is permitted.

T#5m_75s is not allowed. The correct representation of this duration literal is: T#6m_15s.

S1016

Invalid milliseconds entry.

The numeric value for the milliseconds unit of a TIME literal must be a fixed point number in the range [0, 1000] if the milliseconds are not the most

significant unit of the duration literal. An overflow is only permitted if the milliseconds unit is the only unit of the TIME literal.

Example

T#1200s is permitted.

T#1s_1200ms is not allowed. The correct representation of this duration literal is: T#2s_200ms.

S1017

Direct address too complex.

The maximum number of location fields in the address hierarchy of a directly represented variable is hardware dependent but must not exceed 8. Please consult your hardware documentation to determine the maximum depth of the address hierarchy.

S1018

Integer constant too large/small.

A constant's value must be in the range of representable values for its type. The type of an integer constant depends on the type of the variable the constant is assigned to but must not exceed the range of a LINT/ULINT (8 byte integer/unsigned integer) constant.

S1019

Integer constant too large/small (does not fit into 32 bits).

The numeric value of the given constant exceeds the range of values of type DINT/UDINT.

S1020

Numeric value too large/small.

A constant's value must be in the range of representable values for its type. The type of a signed integer constant depends on the type of the variable the constant is assigned to but must not exceed the range of a LINT (8 byte integer) constant.

S1021

Error while processing a floating-point function of the math library.

S1022

Invalid string constant.

The given string constant contains an invalid character. A character string literal is a sequence of zero or more characters prefixed and terminated by the single quote character ('). Valid characters are any printable character except '$'. The three-character combination of the dollar sign ($) followed by two hexadecimal digits shall be interpreted as an hexadecimal representation of the eight bit character code as shown in table Character string literal feature.

Additionally, two-character combinations beginning with the dollar sign shall be interpreted as shown in table Two-character combinations in character strings when they occur in character strings.

S1023

Invalid number (i.e., numerical constant).

The given numeric constant contains an invalid character. See table Numeric literals for examples of valid numeric literals.

S1024

Invalid constant.

The given constant contains invalid characters.

For a list of valid constant representations see Table 53: Function block invocation features for IL language.

S1025

Invalid direct address.

A directly represented variable contains invalid characters.

The direct representation of a variable shall be provided by the concatenation of the percent sign "%", a location prefix, an optional size prefix and one or more unsigned integers separated by periods (.)

The manufacturer shall specify the correspondence between the direct representation of a variable and the physical or logical location of the addressed item in memory, input or output. When a direct representation is extended with additional integer fields separated by periods, it shall be interpreted as a hierarchical physical or logical address with the leftmost field representing the highest level of the hierarchy, with successively lower levels appearing to the right. For instance, the variable %IW2.5.7.1 may represent the first "channel" (word) of the seventh "module" in the fifth "rack" of the second "I/O bus" of a programmable controller system.

The use of directly represented variables is only permitted in programs. The maximum number of levels of hierarchical addressing is hardware dependent and must not exceed 8.

Please consult your hardware documentation to determine the maximum levels of hierarchical addressing.

S1026

Invalid identifier (name, variable, parameter,...)

An identifier contains one ore more invalid characters.

An identifier is a string of letters, digits, and underline characters which shall begin with a letter or underline character. The letters can be upper or lower case. Multiple leading or multiple embedded underlines are not allowed.

Imbedded space characters are not allowed.

S1027

End of file found in file header.

An error occurred while reading the file header. You can fix this error, by opening the file with a text editor and removing all lines preceding the PROGRAM, FUNCTION or FUNCTION_BLOCK keyword. If this error occurs more often, please contact your manufacturer.

S1028

This identifier is too long (> 64 characters).

The length of an identifier is greater than the maximum supported length. In this implementation only identifiers up to 64 characters are supported.

S1029

This word (identifier, constant literal, string, comment) is too long (> 1024 characters).
A token (identifier, constant literal, string, comment) exceeds 1024 characters. In this implementation only tokens up to 1024 characters are supported.

S1030

Too many identifiers.

The maximum number of identifiers has been exceeded. Maximum 65535 identifiers are supported.

S1031

Unallowed usage of EN. Just allowed as an identifier for a bool variable in input section.

A variable with the name "EN" has been declared in the wrong variable section or with incorrect type.

The name "EN" (enable) is reserved for Boolean input variables.

If the value of EN is FALSE when the function or function block is invoked the operations defined by the function/function block shall not be executed. If the Boolean output parameter ENO has been defined too than the value of ENO is reset to FALSE.

If the value of EN is TRUE when the function or function block is invoked the operations defined by the function/function block are executed. These operations can include the assignment of a Boolean value to the Boolean output parameter ENO, if this parameter has been defined too.

S1032

Unallowed usage of ENO. Just allowed as an identifier for a bool variable in output section.

A variable with the name "ENO" has been declared in the wrong variable section or with incorrect type.

The name "ENO" (Enable Out) is reserved for Boolean output variables. The variable "ENO" requires the Boolean input variable "EN".

If the value of EN is FALSE when the function or function block is invoked the operations defined by the function/function block shall not be executed and the output parameter ENO is reset to FALSE.

If the value of EN is TRUE when the function or function block is invoked the operations defined by the function/function block are executed. These operations can include the assignment of a Boolean value to ENO.

S3000

Function block not declared.

A CAL to an unknown function block instance has been found.

An instance of a function block must be declared before it can be used.

Tips:

Make sure that an instance of the requested function block is declared in one of the variable declaration sections.

Make sure the name of the name of the function block instance is spelled correctly.

S3001

Function not present.

A call to an unknown function has been found.

A function must be declared before it can be used. The parameters that a function uses must be specified in a declaration, or prototype, before the function can be used.

Tips:

Make sure that the file containing the declaration or prototype of the function is in the scope of the project or that the function is part of the firmware.

Make sure the name of the name of the function is spelled correctly.

S3002

Incorrect parameter.

The requested parameter was not found in the formal parameter list of the function block.

Tips:

Make sure the name of the name of the parameter is spelled correctly.

Make sure that the parameter list of the function block-definition contains a parameter with the name used in the assignment.

S3003

Jump label not present.

A JMP instruction to an unknown label has been found.

A label has to be defined in the instruction part of the program unit in which it is used.

Tips:

Make sure that a the label is defined in the same program unit.

Make sure the name of the name of the label is spelled correctly.

S3004

Multiple assignment of a variable/name.

The given identifier was defined more than once.

Tips:

Make sure the identifier has not been defined twice in the same program unit.

Make sure the identifier has not been used in a user type declaration, a global type declaration or as a function, function block or program name.

S3005

This is not a function block instance.

A variable with the name used in a CAL-statement has been found but is not an instance of a function block.

Tips:

Make sure that the identifier is spelled correctly.

Make sure that a function block instance with the specified name has been declared either in the scope of the program unit or in the global scope.

S3006

This is not a struct variable or a function block instance.

An access to a member of a struct or function block variable has been attempted, but the variable specified by the identifier is not a function block or a struct.

Tips:

Make sure that the identifier is spelled correctly.

Make sure that the variable with the given name is a struct or a function block.

S3007

This is not a FUNCTION-POU.

An identifier used as a function name has been defined but is not a function name.

Tips:

Make sure that the identifier is spelled correctly.

Make sure that the identifier is the name of a function and not the name of a function block.

Make sure that a function invocation and not a call of a function block instance has been desired on the specified position.

S3008

No structure element or block parameter.

An access to a member of a struct or function block variable has been attempted, but the member specified by the identifier is not a parameter of the accessed function block or struct instance.

Tips:

Make sure that the identifier is spelled correctly.

Make sure that the right function block or struct instance is used.

If the accessed variable is an instance of a function block make sure that the function block has a parameter with the name given by the identifier.

If the accessed variable is an instance of a struct, make sure that the struct has a member with the name given by the identifier.

S3009

No jump label.

The identifier used in the JMP/JMPC/JMPCN-statement at the given position has been found but is not a label name.

Tips:

Make sure that the identifier is spelled correctly.

Make sure that identifier used after the JMP/JMPC/JMPCN-statement is a label name.

S3010

Type or function block name expected.

A type or a function block name has been expected. The identifier has been found in the current scope but is neither a type nor a function block name.

Tips:

Check if the name is spelled correctly.

Make sure that the identifier is not a variable name (e. g. a function block name).

S3011

Identifier is not a variable or type name.

A variable or a function block instance has been expected. The identifier has been found in the current scope but is neither a variable nor a function block instance.

Tips:

Check if the name is spelled correctly.

Make sure that the identifier is not a type name (e. g. a function block name).

S3012

Variable name or constant expected.

This error occurs, if an identifier, which is not a variable name or an enum constant, is used where a variable name or a constant is expected.

Example

```
TYPE
        Colours : (red, yellow, blue) := red;
END_TYPE
VAR
      Colour : Colours := Colours;    (* Error: Enum constant expected.
        EnumType is a type name *)
END_VAR
```

LD Colours (* Error: constant or variable name expected. EnumType is a type name *)

ST Colour

S3014

Numeric data type expected.

Operator and operand type are incompatible. An operand of an ANYNUM type has been expected.

S3016

Bit data type expected.

Operator and operand type are incompatible. An operand of an ANYBIT type has been expected.

S3017

Boolean value expected.

Operator and operand type are incompatible. An operand of type BOOL has been expected.

S3018

Numeric data type expected.

Illegal operand type. Operand of an ANYNUM type expected.

S3019

Operators of type incompatible.

Operand and result type are incompatible.

S3020

Operand types incompatible.

This error occurs if an illegal combination of time and date data types is used for the input parameters of a SUB operation. For allowed combination of the input and output data types for this operation see Table 30 - Functions of time data types in the IEC61131-3 Compliance Statement.

Example

```
VAR
      TimeVar  : TIME;
      DateVar : DATE;
END_VAR
LD DateVar
SUB TimeVar            (* Error: SUB is not defined for the this combination of
input parameters *)
ST DateVar
```

S3022

Invalid operand type for this operation.

Invalid operand type for the operation on the specified position. An operand of type TIME or of an ANYNUM type has been expected.

S3023

Invalid operand type for this operation.

Invalid operand type for the operation on the specified position. An operand of type TIME, TIME_OF_DAY, DATE_AND_TIME or of an ANYNUM type has been expected.

S3024

Invalid operand type for this operation.

Invalid operand type for the operation on the specified position. An operand of an ANYBIT type has been expected.

S3025

Boolean result required.

Incompatible result type. Result should be of type BOOL.

S3026

Undeclared identifier.

This error occurs, if the identifier at the given position, has not been defined in the scope valid for the compiled program organization unit.

Example

```
TYPE
        Colours : (red, yellow, blue) := red;
END_TYPE
VAR
      Colour : Colours := green;    (* Error: green has not been declared as an
enum constant *)
END_VAR

LD IntVar      (* Error: IntVar has not been declared. *)
ADD 5
ST IntVar
```

S3028

Comparison not defined for the data type of the current result.

The comparison on the given position is not defined for the type of the current result. I. e. the type of the actual parameter is incompatible with the type of the first formal parameter. For more information see Table 28 - Standard comparison functions in the IEC61131-3 Compliance Statement.

Example

```
TYPE
       Day_of_Week : STRUCT
    Name : String;
              DayNo : INT(1..7);
       END_STRUCT;
END_TYPE
VAR
      DayVar1 : Day_of_Week;
      DayVar2 : Day_of_Week;
      BoolVar : BOOL;
END_VAR
LD DayVar1
GT DayVar2     (* Error: comparisons on structured variables are not allowed *)
ST boolVar
```

S3030

Comparison not defined for this type.

The type of the operand at the given position is not allowed for comparisons. I. e. the type of the actual parameter is incompatible with the type of the formal parameter. For more information see Table 28 - Standard comparison functions in the 1131-3 Compliance Statement.

Example

```
TYPE
       Day_of_Week : STRUCT
    Name : String;
              DayNo : INT(1..7);
       END_STRUCT;
END_TYPE
VAR
      DayVar1 : Day_of_Week;
      DayVar2 : Day_of_Week;
      BoolVar : BOOL;
```

```
END_VAR
LD DayVar1
GT DayVar2      (* Error: comparisons on structured variables are not allowed *)
ST boolVar
```

S3032

Self-referencing (i.e., recursive) declarations are not allowed.

Recursion detected. A function can not invoke itself recursively, neither directly nor indirectly (i. e. by invoking another function, that invokes one of the functions in the calling hierarchy). Function blocks and programs can not declare instances of themselves, neither directly nor indirectly (i. e. by calling an instance of another function block that declares an instance of a function block type in the calling hierarchy).

S3033

Operand of type TIME expected.

A constant or a variable of type TIME was expected and the operand at the given position is of another type.

Example

```
VAR
      StartTime : TIME_OF_DAY;
      StopTime : TIME_OF_DAY;
      RunTime : TIME := T#10s;
END_VAR
LD  StartTime
ADD 10000       (* Error: operand must be of type TIME *)
ST StopTime
 LD StartTime
ADD RunTime     (* Correct *)
ST Stop Time
```

S3034

String too long for variable.

A string literal has been assigned to a string variable but the string literal does not fit in the string variable. I. e. the length of the string literal is greater than the allocated length of the string variable.

S3035

Unallowed operand type for this function! Numeric operand or operand of date or time type expected.

The operation at the given position is not defined for the type of the current result (i.e. the first actual parameter).

Example

```
VAR
      BitMake: WORD;
END_VAR
LD  BitMask     (* Error: operand must be of type TIME, ANY_DATE or ANY_NUM *)
SUB 3
ST BitMask
```

S3036

Integer constant is out of range.

The integer constant at the given position is not in the range of the associated data type.

Example

```
VAR
      Range1 : UINT(-1..1000);        (* Error: Sign mismatch. Values for UINT
must not be negative *)
      Range2 : INT(-1..36000);     (* Error: Overflow: the upper range is greater
as the
   maximum valid INT value *)
END_VAR
```

S3037

The lower bound of the subrange must not be greater than the upper bound.

The value of the upper bound in the subrange declaration on the specified position is lower than the value of the lower bound. A subrange declaration restricts the range of an integer type to values between and including the specified upper and lower limits, where the upper limit has to be greater than the lower limit.

S3038

Initialization is out of bounds of subrange (Data type is a subrange type).
A variable of a subrange type has been initialized with a value that is out of the range of this subrange type. A subrange declaration specifies that the value of any data element of this type can only take on values between and including the specified upper and lower limits.

S3039

Index is out of bounds.

An access to a variable of an array type has been attempted with an index whose value is out of the range specified in the type or variable declaration.

S3040

Invalid data type. ANY_NUM required.

The operation at the given position is not defined for the type of the current result (i.e. the first actual parameter).

Example

```
VAR
      BitMake: WORD;
END_VAR
LD  BitMask    (* Error: operand must be of type TIME, ANY_DATE or ANY_NUM *)
NEG
ST BitMask
```

S3041

Unallowed EN/ENO type. Must be of type bool. Must not be RETAIN.

An input variable with the name EN or an output variable with name ENO has been declared with an illegal type or with the RETAIN qualifier.

The identifier "EN" is reserved for input variables of type BOOL

The identifier "ENO" is reserved for output variables of type BOOL This variable must not be declared with RETAIN qualifier.

S3042

Missing EN. Use of ENO allowed only in combination with EN.

An output variable with the name "ENO" has been defined but no input variable with name "EN" has been found. The output variable "ENO" can only be used in combination with "EN".

S3044

Data missing. You either need a load or an expression.

The current result is undefined. Either a LD instruction or an expression must precede the instruction on the current position. This error occurs as a consequence of error Syntax Error S5010 . Please move the instruction out of the parenthesis.

S3046

Type names can not be used as an instance names.

A type name or the name of a program organization unit has been used in a declaration as a variable name. Program organization units and types defined on project level are known in the whole project scope and their names can not be used as identifiers for local variables.

Example

```
FUNCTION Power
(* function block declarations *)
(* statements *)
END_FUNCTION

PROGRAM main
VAR
    Power : REAL;      (* Error: Power is not allowed as a variable name, because
it already has been
   used as a function name *)

END_VAR
(* Code *)
END_PROGRAM
```

S3047

Function parameters must be specified in the order as defined in the Function prototype. Permutated parameter sequences will lead to incorrect code even if parameter names are specified.

If a function block is called in ST, the ST compiler translates the given calling parameter list directly to IL code since it has no knowledge of the function

block's declaration. Because of this, the specified order must match the declaration order of the function blocks Input and Output variables.

Example

```
FUNCTION_BLOCK Example
VAR_INPUT
    In1 : int;
    In2 : int;
END_VAR
FUNCITON_BLOCK_END

Program:
VAR
    Instance : Example;
    Local1 : int;
    Local2 : int;
END_VAR
```

(* correct: parameter order matches declaration order *)

Example(In1 := Local1, In2 := Local2);

(* WRONG: does not match declaration order *)

Example(In2 := Local2, In1 := Local1);

S3048

Possible string truncation in assignment.

This warning is issued if the destination string in a string assignment has a shorter overall length than the source string. This check is done at compile time based on the *declared* lengths of both strings.

Example

```
VAR
    strDestination : string[10];
    strSource : string[40];
END_VAR
```

strDestination := strSource;

S4000

'AT%': Simultaneous declaration of several direct variables is invalid.

A list of identifiers has been used in a located variable declaration. Direct representations can only be associated to a single identifier.

Example

The following declaration is not allowed:

```
VAR
        dirVar1, dirVar2, dirVar3 : at%I0.0;
END_VAR
```

S4001

Too many variables (identifiers). Maximum is 60 identifiers.

Too many identifiers in the identifier list of a variable declaration. Identifier lists with maximum 60 identifiers are supported.

S4003

Array too big.

The element count of a dimension in an array declaration exceeds the maximum number of elements supported by ACR-View. The maximum element count is determined by the supported index range.

S4005

Upper bound must be greater or equal than lower bound.

The value of the upper bound index in the array declaration on the specified position is lower than the value of the lower bound index of the same dimension. The upper bound index of a dimension must be greater or equal than the associated lower bound index.

S4006

Syntax error. [Hint: In some cases, the actual error is located in a previous line (';' missing etc.)].

S4007

Self-referencing (i.e., recursive) declarations are invalid.

Recursion detected. A function can not invoke itself recursively, neither directly nor indirectly (i. e. by invoking another function, that invokes one of the functions in the calling hierarchy). Function blocks and programs can not declare instances of themselves, neither directly nor indirectly (i. e. by calling an instance of another function block that declares an instance of a function block type in the calling hierarchy).

S4008

Too many attributes 'RETAIN' or 'CONSTANT'. You may use only one.

Too many qualifiers used in a variable declaration part.

S4009

A STRUCTure must contain at least one structure element (variable declaration).

An empty structure has been declared. This is not allowed. A structure must contain at least one member variable.

Example

The following is not allowed:

```
TYPE
       Mystruct : struct end_struct;
END_TYPE
```

Allowed:

```
TYPE
      Mystruct : STRUCT
    M1 : int;
     END_STRUCT
END_TYPE
```

S4010

Simultaneous type declarations are not allowed.

The type declaration on the specified position contains a list of identifiers. This is not allowed. Please write a declaration for any new type.

Example

The following is not allowed:

```
TYPE
       MyInt1, MyInt2, MyInt3 : int;
END_TYPE
```

Allowed:

```
TYPE
MyInt1 : int;
MyInt2 : int;
MyInt3 : int;
END_TYPE
```

S4011

Valid only in PROGRAMs and there within VAR- and VAR_GLOBAL-Sections.

A directly represented variable has been declared in a program organization unit or a variable declaration part in which it is not supported. Located variable declarations are supported only in VAR- or VAR_GLOBAL-declaration-parts of PROGRAMs.

S4012

Valid only in PROGRAMs, FUNCTION_BLOCKs, and in FUNCTIONs.

A variable declaration part (VAR <declarations> END_VAR) was found in a unit where it is not supported. Variable declaration parts are allowed in programs, functions and function blocks.

S4013

Valid only in PROGRAMs, FUNCTION_BLOCKs, and in FUNCTIONs.

An input variable declaration (VAR_INPUT <declarations> END_VAR) part was found in a program organization unit where it is not supported.

S4014

Valid only in PROGRAMs and in FUNCTION_BLOCKs.

An in/out variable declaration part (VAR_IN_OUT <declarations> END_VAR) was found in a program organization unit where it is not supported.

S4015

Valid only in PROGRAMs and in FUNCTION_BLOCKs.

An output variable declaration part (VAR_OUTPUT <declarations> END_VAR) was found in a program organization unit where it is not supported.

S4016

Valid only in PROGRAMs and in FUNCTION_BLOCKs.

An external variable declaration part (VAR_EXTERNAL <declarations> END_VAR) was found in a program organization unit where it is not supported. External variable declarations are supported in PROGRAMs and FUNCTION_BLOCKs.

S4017

Valid only in PROGRAMs.

A global variable declaration part (VAR_GLOBAL <declarations> END_VAR) was found in a program organization unit where it is not supported. Global variable declarations are allowed in PROGRAMs only.

S4018

Valid only in VAR- and in VAR_GLOBAL-Sections.

The qualifier "CONSTANT" has been used in a variable declaration part in which it is not supported.

S4019

Valid only in PROGRAMs or in FUNCTION_BLOCKs and there within VAR-, VAR_OUTPUT-, or VAR_GLOBAL-Sections).

The qualifier "RETAIN" has been used in a variable declaration part in which it is not supported.

S4020

Valid only in PROGRAMs or in FUNCTION_BLOCKs and there within VAR_INPUT-Sections with Type "BOOL" without Initialization.
A variable has been declared with an edge qualifier in a program organization unit or variable declaration part where this is not supported.

S4021

Valid only within VAR_INPUT, VAR_OUTPUT, and VAR_IN_OUT-Sections.
A variable has been declared with the ADDRESS qualifier in a program organization unit or variable declaration part where this is not supported.

S4022

Valid only in FUNCTION_BLOCKs or FUNCTIONs and there within VAR..END_VAR-Sections without CONSTANT/RETAIN-Modifiers.
A variable has been declared with the ATTRIBUTES qualifier in a program organization unit or variable declaration part where this is not supported. This attribute is supported only in VAR-Sections without CONSTANT or RETAIN qualifiers of FUNCTIONs and FUNCTION_BLOCKs.

Note: Keyword ATTRIBUTES is supported by ACR-View only in custom versions to define additional attributes for variables in extension to IEC61131-3. You should not see this message in standard ACR-View.

S4023

Valid only in TYPE..END_TYPE-Sections.
A struct declaration was found in a declaration part where this is not supported. Struct declarations are supported only in TYPE declaration parts.

S4024

Valid not within VAR_EXTERNAL-Sections.
A variable has been declared in an EXTERNAL declaration section with an initial value. This is not allowed. Please assign the initial value in the respective GLOBAL variable declaration.

Example

```
VAR_EXTERNAL
         A : INT := 5;
END_VAR

VAR_EXTERNAL
    A : INT;
END_VAR

VAR_GLOBAL
        A : INT := 5
END_VAR
```

S4033

Multiple initialization.
A member of a struct variable has been initialized more than once. This error occurs when both an explicit struct initialization and a per element initialization are made.

Example
The following initialization is not allowed:

```
TYPE
        StructType : Struct
       Member1 : int := 5;
       Member2 : bool;
        END_STRUCT := (Member1 :=  4, Member2 := true);
END_TYPE
```

Use one of the following initializations instead:

```
TYPE
        StructType : Struct
       Member1 : int ;
       Member2 : bool;
        END_STRUCT := (Member1 :=  4, Member2 := true);
END_TYPE
```

or

```
TYPE
        StructType : Struct
       Member1 : int := 5;
       Member2 : bool :=  true;
        END_STRUCT;
END_TYPE
```

S4034

Invalid POU name.
This error occurs when a keyword has been used as a POU name or if no name has been defined.

S4035

Invalid type for function.
The function type must be a predefined type or an identifier. This error occurs most commonly, when a reserved keyword, a IEC61131-3 character string or a number is used as a function type or if no function type has been defined.

S4036

FUNCTIONs need at least one input parameter VAR_INPUT.
A function has been defined without an input parameter. In IEC61131-3 a function needs at least one input-parameter.

S5000

Wrong parameter type.
The type of an actual parameter of a function or a function block instance is incompatible with the type of the formal parameter it has been assigned to.

S5001

Array expected. This is not an array.
An indexed access has been attempted to a variable which is not an array.

Example

```
PROGRAM
VAR
        x : INT;
        y : INT;
END_VAR

LD  x[3]                    (* not allowed if the variable is not an array *)
ST  y

END_VAR
```

S5002

This FUNCTION_BLOCK is called by CAL if EN=TRUE. CALC/CALCN are both invalid.
An instance of a function block with an "EN" input parameter has been called via CALC/CALCN. This is not allowed. Use the CAL-statement instead. The code of a function block with an "EN" parameter is invoked if the value of this parameter is TRUE.

S5003

Function block instances may not be "CONSTANT".
An instance of a function block has been defined in a variable section with CONSTANT attribute. This is not allowed. Please remove the attribute or

move the instance declaration in another variable section, which has no CONSTANT attribute.

S5004

Function blocks instances are invalid in "FUNCTION"-POUs, STRUCTs, and in ARRAYs.
An instance of a function block has been defined in a variable section of a function or as a member of a STRUCT or an ARRAY type. IEC61131-3 doesn't allow declarations of function block instances in functions. Function block instances as members of STRUCT and ARRAY types are not supported by ACR-View.

S5005

Function block instances as function results are not supported.
Function block instances as result type of a function are not supported in ACR-View.

S5006

Function block instances as parameters are not supported.
Parameters of a function block type are not supported in ACR-View.

S5008

Expected an integer or an enum. Invalid array index.
The type variable or constant used as an index in an indexed variable access is invalid. An index must be of type INT or of an enumeration type.

S5009

Invalid sequence beginning. Current result is empty. Use "LD" to initialize current result.
This error occurs when a sequence of statements starts with an instruction that uses the current result. The first instruction usually is a load statement. This error can also occur, if the current result is used in the first instruction after a CAL, a JMP or a label.

Example

```
PROGRAM main
VAR
    Switch : BOOL;

END_VAR
ST Switch      (* Error: Current result is undefined. *)

LD Switch
EQ TRUE
JMPC NextStep

LD TRUE
JMP End (* The value loaded in the previous statement will be lost after the JMP-
statement *)

NextStep:
LD FALSE

END:
ST Switch      (* Error: Current result is undefined after a label *)

(* Code *)
END_PROGRAM
```

S5010

Invalid instruction within a parentheses computation.

The instruction at the given position is not allowed between parentheses. Please replace the instruction or move it out of the parentheses.

Example

```
FUNCTION_BLOCK Count
VAR_INPUT
    StartValue : DINT;
    FReset : BOOL;
END_VAR
VAR_OUTPUT
   CurrentCountValue : DINT;
END_VAR
VAR
      CountValue : DINT;
END_VAR
LD fReset
EQ TRUE
JMPCN Continue

LD StarValue
ST CountValue

Continue:
LD CountValue
ADD 1
ST CountValue
ST CurrentCountValue
END_FUNCTION_BLOCK

PROGRAM main
VAR
Counter  : Count;
StartValue : DINT;
Result : DINT;

END_VAR
LD 5
ADD (StartValue
ST Counter.StartValue
EQ 1000
ST Counter.fReset
CAL Counter            (* Error: CAL is not allowed between parentheses *)
LD Counter.CurrentCounter     (* Error: Load  is not allowed between parentheses
*)
)
ST Result
END_PROGRAM.
```

S5011

ARRAYs of function block instances are invalid.

Arrays of function blocks are not supported.

S5012

Result type and operand type are incompatible.

The result type of the preceding operation and the type of the variable in which this result is stored are incompatible.

Example

```
VAR
      X : INT;
END_VAR
LD 65000
ST x                    (* 65000 is not of type INT *)
```

S5013

Result type and type of the first formal input parameter are incompatible.
The result type of the preceding operation and the type of the first input parameter in a function or function block call are incompatible.

Example

```
FUNCTION Fun1
VAR
     InVar : INT;
END_VAR
(* Code *)
END_FUNCTION

PROGRAM main
VAR
      X : DINT;
END_VAR
LD x
ADD 1000
Fun1    (* Error: result type of the preceding operation is DINT, the type of the
first input parameter of       Fun1 is INT *)
ST x
END_PROGRAM
```

S5014

Wrong number of parameters.
Too many parameters found in a call of a function or a function block.

S5015

Invalid type for direct address.

A located variable has been declared with an unsupported type. Only located variables of type ANY_NUM or ANY_BIT are supported.

S5016

Variable is read-only. Write-access invalid.

A write access has been attempted to a variable, that has only read access.

S5017

Variable is not a STRUCTure.

A initialization value for a structure has been assigned to a variable which is not of a structured type.

Example

```
VAR
       A : INT := (m1 := 5, m2 := TRUE);            (* not allowed *)
END_VAR
```

S5018

Variable is no array.
An array initialization has been assigned to a variable which is not of an array type.

Example

```
VAR
       A : INT := [4];               (* not allowed *)
END_VAR
```

S5019

Initialization value and variable type incompatible.

The type of the initialization value and the type of the variable are incompatible.

Example

```
VAR
      X  : INT := 65000;
END_VAR
```

S5020

Too many initialization values.

The initialization value for an array type or variable has more elements as provided by the array declaration.

Example

```
VAR
       A : ARRAY [1..5] OF INT := [1, 2, 3, 4, 5, 6];       (* too many
initialization values,    array has only 5 elements *)
END_VAR
```

S5021

Formal parameter incorrectly declared.

The name of an output parameter has been expected. The identifier has been found in the current scope but is not the name of an output parameter.

Tips:

- Check if the name is spelled correctly.
- Make sure that the identifier is not an input or in/out parameter.

S5022

Multiple assignments to a parameter in a call of a function block instance.

This error occurs, when in a call of a function block instance a parameter is initialized twice.

Example

```
FUNCTION_BLOCK Fb1
VAR_INPUT
                InParam1 : int;
                InParam2 : int;
```

```
                InParam3 : bool;
END_VAR
(* Code *)
END_FUNCTION_BLOCK

PROGRAM main
VAR
      fbInst : fb1;
END_VAR
(* Code *)
cal fbInst( InParam1 := 1,
                InParam1 := 2,
                InParam3 := true
              )
(* Code *)
END_PROGRAM
```

S5023

Too much initialization data.

This error occurs, when a member of a struct type or instance is initialized twice in an explicit structure intialization.

Example

```
TYPE
      StructType : STRUCT
                Member1 : int;
                Member2 : int;
                Member3 : bool;
      END_STRUCT;
END_TYPE
VAR
      StructVar : StructType := (Member1 := 1, Member1 := 2, Member3 := FALSE);
END_VAR
```

S5024

Unallowed type for this operation.

The operation on the given position is not defined for the type of the current result. I. e. the type of the actual parameter is incompatible with the type of the first formal parameter.

Example

```
VAR
      X  : REAL;
END_VAR
LD 1           (* The constant 1 can be converted implicitly to any integer or any
bit type *)
LN      (* Error: LN is only defined for ANY_REAL types *)
ST X
```

S5025

Unallowed parameter type for this function.

The type of the actual parameter is incompatible with any type allowed for the parameter at the given position.

Example

```
VAR
      X  : STRING;
END_VAR
```

```
LD 'EXAMPLE'
LEFT 3.0                (* Error: the second parameter of LEFT has type UINT *)
ST X
```

S5026

Invalid formal parameter type.

The name of an input or an in/out parameter has been expected. The identifier has been found in the current scope but is neither the name of an input nor of an output parameter.

Tips:

- Check if the name is spelled correctly.
- Make sure that the identifier is not an output parameter.

S5027

Incompatible operand types.

The operands for the operation at the given position must be compatible, i. e., they must have the same type or, if at least one of the parameter is a constant an implicit cast to the type of the other operand has be possible.

Example

```
VAR
      X   : REAL;
END_VAR
LD 1            (* The constant 1 can be converted implicitly to any integer or any
bit type *)
MAX X           (* Error: X is of type REAL *)
ST X
```

S5028

Data type not allowed for this operation.

This error occurs, if the type of an actual parameter is not allowed for the operation at the given position.

Example

```
VAR
      StringVar   : STRING;
END_VAR
LD 1
CONCAT 'EXAMPLE'(* Error: CONCAT expects a STRING operand as first input parameter *)
ST StringVar
```

S5029

Invalid function block call.

This error occurs, if a call to a function block instance is attempted and this instance is an input parameter of the calling function block or program.

Example

```
FUNCTION_BLOCK Fb1
VAR_INPUT
                InParam1 : int;
                InParam2 : int;
                InParam3 : bool;
```

```
END_VAR
(* Code *)
END_FUNCTION_BLOCK

FUNCTION_BLOCK Fb2
VAR_INPUT
      fbInstInput : Fb1;
      (* other input declarations *)
END_VAR
VAR
    (* local variable declarations *)
END_VAR
(* Code *)
cal fbInstInput( InParam1 := 1,
                 InParam2 := 2,
                 InParam3 := true
               )
(* Code *)
END_PROGRAM
```

S5030

Variable is write-only. Read-access invalid.

A read access has been attempted to a variable, that has only write access.

S5031

Bit access allowed only on bit data types.

This error occurs if a bit selection is attempted on a variable that is not of a bit data type or of type BOOL.

Example

```
VAR
      DintVar  : DINT;
      BoolVar : BOOL;
END_VAR
LD DintVar.4   (* Error: bit selection allowed only on variables of type ANY_BIT
except BOOL *)
ST BoolVar
```

S5032

Bit position is greater than the number of bits in the selected variable.

This error occurs, when the bit position given in a bit selection is greater than the number of the most significant bit of the selected variable. The number of bits accessible in a bit selection depends on the variables data type. The bit positions are counted from the least significant bit at position 0 to the most significant bit at position n – 1, where n is the number of bits in the data type.

Example

```
VAR
       wVar : WORD := 5;
       fVar : BOOL := FALSE;
END_VAR
(* Code *)
LD wVar.16             (* The selected variable is of type WORD. I. e. it has 16
bits with bit positions
                  from 0 to 15. *)
ST fVar
(* Code *)
```

S5033

IN_OUT parameter missing. Please supply every formal IN_OUT parameter with a an actual parameter.

This error occurs, if at least one of the IN_OUT parameters of a function block is not supplied with an actual parameter, when calling an instance of the respective function block. IN_OUT parameters are references and have to be supplied with an actual parameter in every call of a function block instance.

Example

```
FUNCTION_BLOCK Fb1
VAR_IN_OUT
                InOutParam1 : INT;
                InOutParam2 : BOOL;
END_VAR
(* Code *)
END_FUNCTION_BLOCK

PROGRAM main
VAR
      fbInst : fb1;
      IntVar1 : INT;
      IntVar2 : INT;
END_VAR
(* Code *)
cal fbInst()   (* Error: none of the IN_OUT variables of FB1 is supplied with an
actual  parameter *)
cal fbInst( InOutParam1 := IntVar1
             )        (* Error: the actual parameter for the second IN_OUT
parameter is missing *)

cal fbInst ( InOutParam1 := IntVar1,
                InOutParam2 := IntVar2
       )   (* Correct: every formal IN_OUT parameter of FB1 is supplied with an
actual parameter *)
(* Code *)
END_PROGRAM
```

S5034

Invalid IN_OUT parameter. IN_OUT parameters must not be expressions or constants.

This error occurs, if an IN_OUT parameter is supplied with an expression or a constant value. This is not allowed because IN_OUT parameters are references.

Example

```
FUNCTION_BLOCK Fb1
VAR_IN_OUT
                InOutParam1 : INT;
                InOutParam2 : BOOL;
END_VAR
(* Code *)
END_FUNCTION_BLOCK

PROGRAM main
VAR
      fbInst : fb1;
      IntVar1 : INT;
      IntVar2 : INT;
END_VAR
(* Code *)
cal fbInst( InOutParam1 := IntVar1,
```

```
                InOutParam2 := 5
              )        (* Error: the actual parameter for the second IN_OUT
parameter is a constant. *)

cal fbInst( InOutParam1 := IntVar1,
                InOutParam2 := (IntVar1
       ADD IntVar2)
              )        (* Error: the actual parameter for the second IN_OUT
parameter is an expression. *)

cal fbInst ( InOutParam1 := IntVar1,
                 InOutParam2 := IntVar2
      )   (* Correct: Both IN_OUT parameters of FB1 are supplied with variables.
*)
(* Code *)
END_PROGRAM
```

S5035

Generic data types are not allowed.

This error occurs, if an ANY data type is used in a variable or parameter declaration. The use of generic data types is allowed only for function overloading and type conversion in standard function or functions provided by the manufacturer.

Example

```
FUNCTION IntegerToString : STRING
VAR_INPUT
      InVar : ANY_INT;               (* Error: User-defined functions cannot be
overloaded *)
END_VAR
(* Code *)
END_FUNCTION
```

S5036

Local types are not allowed in this variable section.

This error occurs, if a local user defined type is used in the declaration of a global or external variable or in the declaration of a parameter. Global and external variables as well as parameters have to be of a predefined type or of a global type. Global types are either hardware dependent types, provided by the firmware or project global user defined types.

Example

```
PROGRAM main
TYPE
     StructType : STRUCT
           Member1 : BOOL;
           Member2 : STRING;
    END_STRUCT;
    (* Other type definitions *)
END_TYPE
VAR_GLOBAL
    GlobVar : StructType;      (* Not allowed because StructType is not known in other
POU's *)
    (* Other global variable definitions *)
END_VAR

VAR
    (* Local variable definitions *)
END_VAR
(* Code *)
END_PROGRAM
```

```
FUNCTION_BLOCK Fb1
TYPE
     StructType : STRUCT
           Member1 : BOOL;
           Member2 : STRING;
    END_STRUCT;
END_TYPE
VAR_EXTERNAL
     GlobVar : StructType;     (* Not allowed because StructType is not known in other POU's *)
      (* Other external declarations *)
END_VAR
VAR_INPUT
       InVar : StructType;  (* Not allowed because StructType is not known in other
POU's *)
      (* Other input declarations *)
END_VAR
(* Code *)
END_FUNCTION_BLOCK
```

S5037

Too many indices within the braces [....] of an array-access.

This error occurs, if an access to an array element is attempted with more indices as dimensions provided in the type definition of the elements data type.

Example

```
PROGRAM main
TYPE
     ArrayType : Array[1..5, 1..20] of INT;
    (* Other type definitions *)
END_TYPE
VAR
    ArrayVar : ArrayType;
             IntVar : INT;
    (* Other variable definitions *)
END_VAR
LD ArrayVar[1, 2, 3]          (* Error: Variables of type ArrayType have only 2
dimensions *)
ST IntVar

 (* Code *)
END_PROGRAM
```

S5038

Directly represented variables are only allowed as parameters in prototypes.

A directly represented variable has been declared in the VAR_INPUT, VAR_OUTPUT or VAR_IN_OUT section of a program organization unit. This is not allowed. Directly represented variables are not allowed in functions and function blocks. VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT variables are not supported in programs.

If you want to access a directly represented variable from a function block, declare the variable with a symbolic name in the VAR_GLOBAL section of a program and use this symbolic name in a declaration in the VAR_EXTERNAL section of the function block.

Functions cannot access directly represented variables.

Example

```
FUNCTION_BLOCK SetOutput
VAR_EXTERNAL
      OutputLocation : BOOL;
END_VAR
VAR_INPUT
    Value : BOOL;
END_VAR
LD Value
ST OutputLocation
END_FUNCTION_BLOCK

PROGRAM main
VAR_GLOBAL
    OutputLocation AT%Q0.0 : BOOL;
END_VAR
VAR
Switch : SetOutput;
CurrentValue : BOOL;
END_VAR
LD CurrentValue
NOT
CAL Switch(Value := CurrentValue)
END_PROGRAM.
```

S5039

'&x' is only allowed if x is a direct variable.

The identifier preceded by the &-operator is not the name of a directly represented variable.

Tips:

- Make sure that the name is spelled correctly.
- Make sure that the variable is a directly represented variable.

S5040

Too few indices within the braces [....] of an array access.

This error occurs, if an access to an array element is attempted with less indices as dimensions provided in the type definition of the elements data type.

Example

```
PROGRAM main
TYPE
     ArrayType : Array[1..5, 1..10, 1..20] of INT;
    (* Other type definitions *)
END_TYPE
VAR
    ArrayVar : ArrayType;
             IntVar : INT;
    (* Other variable definitions *)
END_VAR
LD ArrayVar[1, 2]          (* Error: Variables of type ArrayType have 3 dimensions
*)
ST IntVar

 (* Code *)
END_PROGRAM
```

S5041

Values of type INT24 or REAL48 are invalid in this context.

Operation not supported for this type.

S5042

Function block instances may not be 'RETAIN'.

An instance of a function block has been defined in a variable section with RETAIN attribute. This is not supported. Please remove the attribute or move the instance declaration in another variable section, which has no RETAIN attribute.

S5043

Variables, constants and parameters are not allowed as initialization values in declarations. Please use a literal or enumeration value.

In declarations variables, constants or parameters cannot be used to initialize values.

S6002

No prototype.

An unknown type name has been used in a variable declaration or a function call.

Tips

- Make sure that a type a function or function block with this name is declared in the context of the active project.
- Make sure the name of the type, function or function block is spelled correctly.
- Recompile the whole project.
- Please consult your hardware documentation if none of the above actions eliminates the problem.

S6004

Recursion (i.e., direct or indirect self-reference) detected.

Recursion detected. A function can not invoke itself recursively, neither directly nor indirectly (i. e. by invoking another function, that invokes one of the functions in the calling hierarchy). Function blocks and programs can not declare instances of themselves, neither directly nor indirectly (i. e. by calling an instance of another function block that declares an instance of a function block type already used in the calling hierarchy).

S6005

Too many types and function blocks. For the maximum number of type definitions please consult your hardware documentation.

This error occurs, if too many types functions or function blocks have been used in the calling hierarchy of a program organization unit. For the maximum number of types, functions and function blocks supported see the Table D.1: Implementation-dependent parameters

Linker Messages

L10001

Variable declared twice: <Variable name>.

The variable with the specified name has been declared twice.

Tips:

- If the variable is declared in a PROGRAM POU, check if a resource global variable with the same name has been declared.
- If the variable is a resource global variable check if a global variable with the same name has been declared in a PROGRAM POU of the resource.
- If one of the above cases is true, change the name of one of the variables or move the variable declaration in the PROGRAM POU in a VAR_EXTERNAL section. Attention: if you move the variable into the external section, every access to the external variable accesses the resource-global variable with the same name.

L10004

Unresolved external: <Variable name>.

Either a global variable with the specified name has not been found, or a function block type with the specified name has not been found.

Tips:

- Make sure that the variable name is spelled correctly.
- If the variable is not a function block instance, make sure that a variable with this name is declared in the VAR_GLOBAL section of the calling program or in a file with resource-global variable declarations.
- If the variable is a function block instance, make sure that the function block has been compiled successfully, i. e. an object file for this function block exists.

L10026

Unsupported address: <AddressDescription>.

The address <AddressDescription> is not supported by this hardware.

Tips:

- Check if the address is spelled correctly.
- Check if the syntax of the address description is correct. The syntax of the address description is hardware dependent, but must be a string

formed of the percent sign "%" followed by a location prefix, a size prefix and one or more unsigned integers, separated by periods (.). The size prefix may be empty. For valid location and size prefixes consult your hardware documentation.

L10027

Invalid hardware description: %1..

The hardware description file for the hardware with name <hardware name> has not been found.

Tips:

- Check if the resource specification contains a valid hardware module name.
- Reinstall ACR-View. If this doesn't remove your error, consult your hardware documentation or refer to your hardware manufacturer.

L10029

Hardware configuration error.

An error occurred while getting firmware information. Please check if the hardware configuration file is correct or if the DLL for the specified firmware is installed in your ACR-View directory.

ATTENTION: This file should be altered only by the manufacturer.

L10030

Invalid type for variable: %1.

A directly represented variable of a complex type (array, struct, string) has been found. This is not supported by the hardware.

L10031

Initializations of directly represented variables are not allowed.

An initialization of a directly represented variable has been found. This is not supported by the hardware. Please remove the initialization.

L10032

Address <AddressDescription> invalid in this context.

The address with the specified description is a valid address but not allowed in this context (Task, POU, Resource, Configuration).

L10033

Attribute RETAIN not supported for directly represented variables.

A directly represented variable with RETAIN attribute has been found. This is not supported by the hardware. Please move the variable declaration in another section or remove the attribute from the section.

L10034

Attribute CONST not supported for directly represented variables.

A directly represented variable with CONST attribute has been found. This is not supported by the hardware. Please move the variable declaration in another section or remove the attribute from the section.

L10035

Instance limit for function block <FunctionBlockName> reached.

The maximum number of instances of the specified function block has already been exceed. The maximum number of instances of a firmware function block is hardware dependent and can be changed by the hardware manufacturer by setting or changing the "MaxInstances" entry in the specification section of the function block in the hardware description file. Please consult your hardware documentation, for the maximum number of instances of a firmware function block.

L10036

Invalid process image description. Please contact your manufacturer.

The description of the process image in the hardware configuration file is invalid. Please check if the sizes for the input, output and marker sections are correct and if all size entries are of the same unit. They should be specified either in bits or bytes.

ATTENTION: This file should be altered only by the manufacturer.

L10063

An error occurred while opening a file: %1.

L10105

Internal error while loading function or DLL: <DLL/Function-Name>.

The specified DLL or function could not be loaded. Either your ACR-View directory does not contain a DLL with the specified name, or your DLL has an invalid version. Please reinstall your system or consult your hardware description.

L10106

Native code compiler needed for selected optimization. Please choose another optimization or install a native code compiler.

"Speed only" optimization is activated but no native code compiler is defined for this hardware. "Speed only" optimization is only valid, if a native code compiler is installed. If you do not have a native code compiler please select another optimization in the "Edit Resource Specifications" dialog. For a native code compiler for your hardware please refer to your manufacturer.

L12001

Type conflict. Type of external the variable doesn't match with type of the global variable with the same name.

A global variable with the same name as the external variable has been found, but the types of the global and the external variable are different.

Tips:

- Make sure that the external variable name is spelled correctly.
- Make sure that the type of the external variable is spelled correctly.
- Make sure that the global variable is the requested variable.
- Change the type of the external or the global variable.

L12002

Readable access to this variable is not allowed: <Variable name>.

A read access to a variable that has only write access has been attempted.

Tips:

- Make sure that the specified variable name is spelled correctly.
- The specified variable is an output location. A read access to output locations is not allowed.

L12003

Writable access to this variable is not allowed: <Variable name>.

A write access to a variable that has only read access has been attempted.

Tips:

- Make sure that the specified variable name is spelled correctly
- The specified variable is a constant. Write access to a constant variable is not allowed. Check if the CONSTANT attribute can be removed from the variable.
- The specified variable is an input location. A write access to input locations is not allowed.

L12005

Internal linker error no.: <errorno>. Please contact your manufacturer.

L12006

Memory allocation failure. Not enough memory to perform operation.

L12007

No object information found for task <TaskName>. Please rebuild all.
The object file (<TaskName>.crd) for the specified task has not been found. Please rebuild the whole resource.

L12008

Interpreter stack overflow in task <TaskName>.

Interpreter call-stack-overflow. Please reduce the depth of the calling hierarchy of <TaskName>.

L12064

Error exporting OPC variables to OPC server configuration. Error code: %1.
An OPC variable is erroneous. Please use a proper one.

L12065

Error initializing ConfOPC.DLL. Please contact your manufacturer.

The DLL could not be initialized. Please ask the hardware manufacturer.

L12066

Incorrect alignment for address <address>: variable must be placed at an alignment border."

The direct variable should be moved to a properly aligned address, in order to avoid potential erroneous behavior on some controllers that have an alignment of 2 or 4. With alignment 2, all variables having the size of a WORD (W) or a DWORD (D) should be move to even addresses. With alignment 4, all variables having the size of a WORD (W) should be moved to even addresses and all variables having the size of a DWORD (D) should be moved to adresses divisible by 4.

L12996

Unknown command: <Command>.

An unknown command line argument has been used with ITLINK.

L12997

Unkown object kind: <ObjectKindSpecification>.

An invalid object file has been found. Please rebuild the whole resource.

L12998

Invalid object kind. Kind found/requested: <ObjectKind>.

An invalid object file has been found. Please rebuild the whole resource.

L12999

Invalid object version found. Object version found/expected: <ObjectVersion>.
The object file version and the compiler object version are different. The object file has been created with a different compiler version. Please recompile the whole resource.

L13000

Load of resource global variable information failed.

The object file with the resource global information has not been found. Please rebuild the whole resource.

L13001

No object information found for pou <pouname>

The object file (<pouname>.obj) for the specified POU has not been found. Please rebuild the whole resource.

L15001

An undefined task type has been used or no task type has been defined for task %1.

Check the configuration parameters of the properties of the task type. You may also ask your hardware manufacturer.

Compiler Messages

C10006

Data type 'REAL' is not supported.

Data type ‚REAL' is not supported by the active hardware. For a list of data types supported by ACR-View see the IEC61131-3 Compliance statement Please consult your hardware documentation for a list of data types supported by your hardware.

C10007

Data type 'DATE' is not supported.

Data type ‚DATE' is not supported. For a list of data types supported by ACR-View see IEC61131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

C10008

Data type 'TIME_OF_DAY' is not supported.

Data type ‚TIME_OF_DAY' is not supported. For a list of data types supported by ACR-View see IEC61131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

C10009

Data type 'STRING' is not supported.

Data type ‚STRING' is not supported by the active hardware. For a list of data types supported by ACR-View see the IEC61131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

C10010

Data type 'DATE_AND_TIME' is not supported.

Data type ‚DATE_AND_TIME' is not supported. For a list of data types supported by ACR-View see the IEC61131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

C10012

Data type 'TIME' is not supported.

Data type ‚TIME' is not supported by the active hardware. For a list of data types supported by ACR-View see the IEC61131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

C10017

The sections 'VAR_INPUT', 'VAR_OUTPUT' and 'VAR_IN_OUT' are not supported in programs.

VAR_INPUT, VAR_OUTPUT, and VAR_IN_OUT sections in programs are not supported. For more information about supported variable types see the IEC61131-3 Compliance statement.

C10019

Directly represented variables are not allowed in this POU.

Either the program organization unit is a function or a function block or a file with global symbolic variable definitions. Directly represented variables are not allowed in functions or function blocks. If you want to access a directly represented variable from a function block, declare the variable with a symbolic name in the VAR_GLOBAL section of a program and use this symbolic name in a declaration in the VAR_EXTERNAL section of the function block. Functions cannot access directly represented variables.

Directly represented resource global variables have to be declared in a specific file.

C10020

Bit access not allowed for this variable/parameter.

Variable or parameter has to be of the ANY BIT type.

C10021

Constant must not be negative.

A negative constant has been found where an unsigned operand has been expected. Please change the constant value or the variable type (if possible).

C10024

Constant is out of range.

The constant at the given position is not in the range of the associated data type.

C10025

IN/OUT parameters must always be supplied with actual parameters.

A formal in/out parameter has been declared in a function block, but not supplied with an actual parameter in the CAL statement of an instance. In/out parameters are references and must be supplied with an actual parameter.

C10026

Unsupported address.

The address at the given position is not supported by the active hardware. Please consult your hardware documentation for a list of addresses supported by the hardware.

C10028

Inout-parameters of type struct are not supported.

Structured in/out-parameters are not supported. Please define an input parameter and an output parameter of this kind.

C10031

RETAIN-variables are not supported by this hardware.

Your hardware doesn't support RETAIN variables. Please remove the attribute. For a list of supported variable types consult your hardware documentation.

C10034

Invalid command for this hardware.

The command at the given position is not supported by this hardware. For a list of unsupported commands p consult your hardware documentation. For a list of commands not supported by ACR-View see the IEC61131-3 Compliance statement.

C10035

The operand/parameter must be of type 'UINT'.
An actual parameter of type UINT has been expected in a function call (operation), but the actual parameter is not of this type.

Example

```
VAR
      StringVariable : STRING;
      Length : INT := 32;
END_VAR
LD 'EXAMPLE'
LEFT length    (* Error: this parameter must be of type UINT *)
ST StringVariable
```

C10036

Structs and arrays of complex data types are not supported by this *hardware.*
An array of a structured type, an array of an array type, a structure with a structured member or a structure with an array member has been declared. This is not supported by the hardware. For more information about supported data types for your hardware, consult your hardware documentation.

Example

```
TYPE
      DayOfWeek : STRUCT
           Name : STRING;
           DayNumber : UINT;
     END_STRUCT;

    DayDescriptions : ARRAY[1..100] OF DayOfWeek; (* Error: Day of Week is a
complex data type.
                      Arrays of complex data types are not supported by the
hardware. *)


Presence : STRUCT
    Name : STRING;
    OursPerDay : ARRAY[1..31] OF UINT; (* Error: ARRAY is a complex data type.
        Structs of complex data types are not supported by the hardware *)
END_STRUCT;
```

C10038

Couldn't detect the type of the constant.

The type of a constant could not be determined. Please initialize a variable of the desired type with this constant and use the variable instead of the constant.

C10043

Implementation code is not allowed.

Implementation code has been found in a file with resource global variable declarations. This is not allowed. Please declare the requested variable in another program organization unit as an external variable and move the code in the respective file.

C10045

Function blocks instances are not allowed in this section.

An instance declaration of a function block has been found in a section where this is not allowed. Please move the declaration in a section, where function block instances are supported.

C10046

'VAR_GLOBAL' is not allowed.

A VAR_GLOBAL section has been found in a program organization unit where this section kind is not supported. Please change the section kind or move the variable declaration in a file, where global variables are supported.

According to the IEC61131-3 VAR_GLOBAL sections are supported only in PROGRAMs. However the hardware manufacturer may restrict the declaration of global variables to resource global variable files. I. e. global variables are allowed only in specific files which contain only global variable declarations.

C10047

Only 'VAR_GLOBAL' allowed.

A variable declaration section, which is not a VAR_GLOBAL section, has been found in a file for resource global variable declaration. This is not allowed. Please change the section kind or move the variable declaration in another file, where this kind of declarations are supported.

C10049

String too long.

A string has been declared with a length specification, which exceeds the maximum string length supported by the hardware.

For the maximum string length supported by ACR-View see the IEC61131-3 Compliance statement. However, the hardware-manufacturer can restrict the maximum string length by changing the value of the "MaxStringLength" entry in the [MODULE] section of the hardware description file.

C10055

This variable can not be initialized.

Either an initialization of a directly represented variable has been found or the hardware doesn't support variable initializations. The initialization of directly represented variables is not supported by ACR-View. The initialization of symbolic variables can be forbidden by the manufacturer by changing the value for the "InitVariables" entry in the [MODULE] section of the hardware description file to 0. Please consult your hardware documentation to find out, if variable initialization is supported by your hardware.

C10057

Data type is not supported.

The data type at the given position is not supported. For a list of data types supported by ACR-View see the IEC61131-3 Compliance statement. For a list of data types supported by your hardware, please consult your hardware documentation.

C10060

LD/ST of function block instances is not allowed.

A LD or ST instruction with a function block instance as an operand has been found. This is not allowed.

C10063

An error occurred while opening a file.

C10064

Internal Compiler Error No. %1. Please contact your manufacturer.

An internal compiler error occurred. Please contact your manufacturer.

C10067

Struct declarations are not supported.

A struct declaration has been detected, but is not supported by the hardware. Struct declarations are supported by ACR-View. The hardware manufacturer however, can forbid struct declarations by setting the value of the "StructAllowed" entry in the [MODULE] section of the hardware description file to 0. Please consult your hardware documentation to find out if struct declarations are supported by your hardware.

C10068

Array declarations are not supported.

An array declaration has been detected, but is not supported by the hardware. Array declarations are supported by ACR-View. The hardware manufacturer however, can forbid array declarations by setting the value of the "ArrayAllowed" entry in the [MODULE] section of the hardware description file to 0. Please consult your hardware documentation to find out if array declarations are supported by your hardware.

C10069

Enumerated data type declarations are not supported.

A enumerated data type declaration has been detected, but is not supported by the hardware. Enumerated data type declarations are supported by ACR-View. The hardware manufacturer however, can forbid this declarations by setting the value of the "EnumAllowed" entry in the [MODULE] section of the hardware description file to 0. consult your hardware documentation to find out if enumerated data type declarations are supported by your hardware.

C10075

Invalid array index. It has to range between -32767 and 32767.

An array index is out of the supported range [-32767, 32767].

C10078

Invalid type of a global or directly represented variable.

A directly represented variable of a complex or an user defined type has been declared. This is not supported. Global variable of structured types are also not supported.

C10083

Only directly represented variables are allowed in this POU.

Resource global variables are separated in two kind of files. Files which contain only symbolic variables and files which contain the directly represented variables. In these files symbolic and directly represented variables must not be mixed up.

C10084

Global structs are not supported.

Please declare this variable in a local section and use input and output parameters, if the value should be changed by a function or function block. The type declaration for the desired structure must be done on project level.

Example

```
(* The following structure has to be declared as a project global type*)
TYPE
      DayOfWeek : STRUCT
           Name : STRING;
           DayNumber : UINT;
     END_STRUCT;
END_TYPE

FUNCTION_BLOCK AdjustDayName
VAR_INPUT
      DayIn : DayOfWeek;
END_VAR
VAR_OUTPUT
     DayOut : DayOfWeek;
END_VAR
LD DayIn
ST DayOut

LD DayIn.DayNumber
EQ 1
LD 'MONDAY'
ST DayOut.Name

LD DayIn.DayNumber
EQ 2
LD 'TUESDAY'
ST DayOut.Name

END_FUNCTION_BLOCK

PROGRAM main
```

```
VAR
      Day : DayOfWeek;
      DayNumber : UINT;
END_VAR

LD DayNumber
ST Day.DayNumber
CAL AdjustDayName(DayIn := Day | Day := DayOut)

END_PROGRAM
```

C10092

Memory allocation failure.

C10093

Data Segment Out Of Memory

To much data (for example, variables) for program or function block so the data doesn't fit into a 64 kB segment. Segments are restricted to 64 kB.

Note:
If this error occurs, try to restruct the program/function block and put some variables into other function blocks (FBs can be used as data containers) or use resource global variables.

C10094

Initial Data Segment Out Of Memory

To much data (for example, variables) for program or function block so the data doesn't fit into a 64 kB segment. Segments are restricted to 64 kB.

Note:
If this error occurs, try to restruct the program/function block and put some variables into other function blocks (FBs can be used as data containers) or use resource global variables.

C10095

Code Segment Memory Allocation Failure

This error occurs if the program code (UCode/Native Code) doesn't fit into a 64 kB segment. The size for a segment is restricted to 64 kB.

Note: If this error occurs, it is possible to restruct the program (for example, putting some parts of the code into Function Blocks) so that the program decreases down to 64 kB.

C10100

Invalid expression for parameter.

An invalid expression has been passed as an actual parameter in a call of a function or a function block instance.

C10108

Constant of type TIME is out of range.
For the range of TIME constants supported by ACR-View see the IEC61131-3 Compliance statement.

C10109

Invalid data type for this operation. Integer or real type expected.

The operation at the given position is only supported for integer and real operands.

C10110

Nested functions are not supported.

A function call has been passed as an actual parameter in the call of a function or a function block instance. This is not supported. Please save the return value of the function in a variable and pass this variable as an actual parameter to the called program organization unit.

C10112

Type conflict.

Either the current result is incompatible with the expected data type or the type of an actual parameter is incompatible with the type of the respective formal parameter.

C10113

Operation not supported for this data type.

The data type of an operand is not allowed for the operation at the given position. For more information about allowed data types for this operation see IEC61131-3 and the IEC61131-3 Compliance statement.

C10114

Parameter expressions are not supported for this operation.

An expression has been used as an actual parameter. This is not supported. Please store the result of the expression in a variable and pass this variable to the called function or function block.

C10115

Retain attribute for FB instances forbidden.

RETAIN function block instances are not supported. Please remove the attribute or move the instance declaration out of this section.

C11001

Can't determine unambiguously the type of constant -> take %1.

The type of a numeric constant couldn't be determined unambiguously. In this case usually the biggest supported data type of the expected data type class (ANY_INT, ANY_REAL, ANY_BIT) is presumed.

C11007

Function has no input parameter. Is this intended?
A function call to a function which has no parameters has been detected. Was this the intend? Functions do not contain internal state information

and can be supplied only with input parameters. Generally the return value is computed by using the input parameters. Because of this reasons a function without input parameters usually doesn't make sense. Please check if the called function makes sense.

Make Messages

M21004

Unknown command: %1.

An unknown command line argument has been used with ITMAKE.

Shortcuts

Common Shortcuts

File Submenu

CTRL+N:	New File
CTRL+F4:	Close
CTRL+S:	Save
ALT+F10:	Syntax Check
CTRL+P:	Print
CTRL+O:	Open Project
ALT+F4:	Exit

Edit Submenu

CTRL+Z:	Undo
CTRL+Y:	Redo
CTRL+X/SHIFT+DEL:	Cut
CTRL+C/CTRL+INS:	Copy
CTRL+V/SHIFT+INS:	Paste
DEL:	Delete
F4:	Next Error
SHIFT+F4:	Previous Error
CTRL+F:	Find
CTRL+H:	Replace
CTRL+G:	Goto IL Line (SFC)
CTRL+A:	Select All
ALT+RETURN:	Properties

PLC Submenu

F7:	Build Active Resource
CTRL+F7:	Rebuild Active Resource
F9:	Toggle Breakpoint
F5:	Go
F11:	Step Into
F10:	Step Over
SHIFT+F11:	Step Out

ALT+ENTER:	Resource Properties

Window Submenu

F6:	Next Pane
ALT+1:	Project
ALT+2:	Document
ALT+3:	Test and Comissioning
ALT+4:	Output
Ctrl+Enter:	Fullscreen

Insert Variable Submenu

ALT+SHIFT+V:	All Variables
ALT+SHIFT+I:	Input Variables
ALT+SHIFT+O:	Output Variables
ALT+SHIFT+N:	In/Out Variables
ALT+SHIFT+L:	Local Variables
ALT+SHIFT+G:	Global Variables
ALT+SHIFT+E:	External Variables
ALT+SHIFT+F:	FB-Instance Variables

Editor depending Shortcuts

IL/ST Editor

CTRL+ALT+F:	Insert Function
CTRL+ALT+B:	Insert Functionblock

LADDER Editor

F12:	Insert Network
CTRL+ALT+F:	Insert Function
CTRL+ALT+B:	Insert Functionblock

SFC Editor

CTLR+ALT+S:	Insert Step/Transition
CTLR+ALT+L:	Insert Step/Transition left
CTLR+ALT+R:	Insert Step/Transition right
CTLR+ALT+J:	Insert Jump
CTLR+ALT+B:	Insert Functionblock
CTLR+ALT+F:	Insert Function

CFC/FBD Editor

CTRL+B:	Insert Connection
CTRL+SHIFT+V:	Switches between variable value and variable name at the margins in online mode

Index