IEC61131 Reference
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.

The information in this user guide, including any apparatus, methods, techniques, and concepts described herein, are the proprietary property of Parker Hannifin or its licensors, 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 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

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

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

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

© 2003-2010 Parker Hannifin Corporation
All Rights Reserved
# Table of Contents

**Important User Information** ................................. 2  
**Table of Contents** .................................................. 4  
**Introduction** .......................................................... 5  

**ACR-View IEC PLC Tools** ........................................ 6  
- ACR-View .................................................................. 6  
- Browser ..................................................................... 7  
- Catalog ...................................................................... 13  
- Declaration Editor .................................................. 15  
- ST Editor .................................................................... 20  
- Ladder Diagram Editor ........................................... 23  
- CFC Editor .................................................................. 27  
- IEC PLC Debug ....................................................... 45  
- Documentation ......................................................... 48  
- Libraries .................................................................... 52  
- IEC61131-3 ............................................................... 54  
- Online Features ....................................................... 78  

**Reference Listings** .................................................. 81  
- Keywords (by category) ............................................. 81  
- Keywords (A..Z) ......................................................... 86  
- Errors and Warnings ............................................... 134  
- Shortcuts ................................................................. 182  

**Index** ..................................................................... 184  

---

IEC 61131 User Guide
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 IEC 61131-3. This user guide details the features and commands available for programming the ACR96x0 controller using IEC 61131-3.

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 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:

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 IEC 61131-3.
- **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 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:

![Edit Resource Specifications - parker ACR PLC](image)

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:

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

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**

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Name</th>
<th>Range</th>
<th>Size in Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>Boolean</td>
<td>0 (FALSE), 1 (TRUE)</td>
<td>1 or 8</td>
</tr>
<tr>
<td>SINT</td>
<td>Short Integer</td>
<td>-128 to +127</td>
<td>8</td>
</tr>
<tr>
<td>USINT</td>
<td>Unsigned Short Integer</td>
<td>0 to 255</td>
<td>8</td>
</tr>
<tr>
<td>INT</td>
<td>Integer</td>
<td>-32 768 to +32 767</td>
<td>16</td>
</tr>
<tr>
<td>DINT</td>
<td>Double Integer</td>
<td>-2.147.483.648 to +2.147.483.647</td>
<td>32</td>
</tr>
<tr>
<td>UINT</td>
<td>Unsigned Integer</td>
<td>0 to 65 535</td>
<td>16</td>
</tr>
<tr>
<td>UDINT</td>
<td>Unsigned Double Integer</td>
<td>0 to 4.294.967.295</td>
<td>32</td>
</tr>
<tr>
<td>REAL</td>
<td>Real number</td>
<td>+/-3,4E+-38</td>
<td>32</td>
</tr>
<tr>
<td>TIME</td>
<td>Time duration</td>
<td>00:00:00:00 to 23:59:59.999</td>
<td>32</td>
</tr>
<tr>
<td>STRING</td>
<td>Character String</td>
<td>length of string plus 2 bytes</td>
<td></td>
</tr>
<tr>
<td>Keyword</td>
<td>Name</td>
<td>Range</td>
<td>Size in Bits</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------</td>
<td>------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>WSTRING</td>
<td>2-byte-character String</td>
<td>length of wstring plus 2 bytes</td>
<td></td>
</tr>
<tr>
<td>BYTE</td>
<td>Sequence of 8 bits</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WORD</td>
<td>Sequence of 16 bits</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>DWORD</td>
<td>Sequence of 32 bits</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

**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:

```plaintext
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 A the struct itself consists of another struct called B and an integer array of size 5. Three new datatypes are derived within B: Stationname as string and Value1, Value2 as reals.

```pascal
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

```pascal
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**

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

Example: Enumeration Data Type

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

```plaintext
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:

```plaintext
VAR_EXTERNAL
  st_active : BOOL;
END_VAR

VAR_GLOBAL
  st_counter : INT;

(* ST_PROG - demo ST programm *)
  st_active := TRUE;
  if (st_active) then
    (* next line is a good location for a breakpoint *)
    st_counter := st_counter + 1;
  else
    st_idle := st_idle + 1;
  end_if;
```

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 semicolon.

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.

<table>
<thead>
<tr>
<th>Element</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parentheses</td>
<td>(</td>
</tr>
<tr>
<td>function call</td>
<td></td>
</tr>
<tr>
<td>Exponentiation</td>
<td>**</td>
</tr>
<tr>
<td>Negation</td>
<td>-</td>
</tr>
<tr>
<td>Complement</td>
<td>NOT</td>
</tr>
<tr>
<td>Multiplication</td>
<td>*</td>
</tr>
<tr>
<td>Division</td>
<td>/</td>
</tr>
<tr>
<td>Modulo</td>
<td>MOD</td>
</tr>
<tr>
<td>Addition</td>
<td>+</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
</tr>
<tr>
<td>Comparison</td>
<td>&lt;, &gt;, &lt;=, =&gt;</td>
</tr>
<tr>
<td>Equality</td>
<td>=</td>
</tr>
<tr>
<td>Inequality</td>
<td>&lt;&gt;</td>
</tr>
<tr>
<td>boolean AND</td>
<td>&amp;, AND</td>
</tr>
<tr>
<td>boolean exclusive OR</td>
<td>XOR</td>
</tr>
<tr>
<td>boolean OR</td>
<td>OR</td>
</tr>
</tbody>
</table>

**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.

```
control.speed := 15;
```

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.

```
control.speed := 15;
```
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 and 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:

![Contact Symbols](image)

- 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**

1) 

2) 

• 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 IEC 61131-3 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 be 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 a 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 or 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 (context) 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.

`ADD_INT_FBD`

IN1 → Output

IN2

IN3

IN4

IN5

Undo
Cut
Copy
Delete
Duplicate
Select All
Find/Replace...
Insert Function...
Insert Functionblock...
Insert Operator
Insert Compound Block
Insert Text Block
Replace Function

Append Input
Toggle Unused Connectors
Zoom In
Zoom Out

View/Hide Grid
Properties...
Show documentation
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 simultaneously. **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 (equivalent 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 granted 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 <ctl>+<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**

![Diagram](image)

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 grid 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 connector 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

\(<\text{LEFT}>\) or \(<\text{RIGHT}>\) 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 \(<\text{RIGHT}>\). The same behavior happens for the caret connector column as 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 \(<\text{ALT}> + <\text{UP/DOWN/LEFT/RIGHT}>\).

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 be 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 lose 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 compound 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 connector, 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 & Commissioning'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 IEC 61131-variables are used as connectors, there maybe more than one sourceline. They have the following form:

\[ \text{varname}\{\text{scope}\} \ldots \]

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

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 name D_1 in block chart 1.Block C.Block D.

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.

![Active Document Server Interface](image)

**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 from 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:

```plaintext
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:

```plaintext
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:

<table>
<thead>
<tr>
<th>Predefined character constants</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>`'&quot;'</td>
<td>The Apostrophe <code>'&quot;</code></td>
</tr>
<tr>
<td>`'$'</td>
<td>The $ sign itself</td>
</tr>
<tr>
<td><code>$L</code> or <code>$l</code></td>
<td>Line Feed</td>
</tr>
<tr>
<td><code>$N</code> or <code>$n</code></td>
<td>New Line</td>
</tr>
<tr>
<td><code>$P</code> or <code>$p</code></td>
<td>Form Feed</td>
</tr>
<tr>
<td><code>$R</code> or <code>$r</code></td>
<td>Carriage Return</td>
</tr>
<tr>
<td><code>$T</code> or <code>$t</code></td>
<td>Tabulator</td>
</tr>
</tbody>
</table>

**Example**

<table>
<thead>
<tr>
<th>Character Constant</th>
<th>Meaning and Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>`'A'</td>
<td>Single character A, length=1</td>
</tr>
<tr>
<td>`''</td>
<td>Blank character, length=1</td>
</tr>
<tr>
<td>`&quot;&quot;</td>
<td>No character, length=1</td>
</tr>
<tr>
<td><code>$R$L</code></td>
<td>Carriage Return, Line Feed, length=2</td>
</tr>
<tr>
<td><code>$0D$0A</code></td>
<td>Carriage Return, Line Feed, length=2</td>
</tr>
</tbody>
</table>
# 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

```plaintext
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.

<table>
<thead>
<tr>
<th>Constant Data Type</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT</td>
<td>-13</td>
<td>Integer -13</td>
</tr>
<tr>
<td></td>
<td>45165 or 45_165</td>
<td>Integer 45165 (both)</td>
</tr>
<tr>
<td></td>
<td>+125</td>
<td>Integer 125</td>
</tr>
<tr>
<td>REAL</td>
<td>-13.12</td>
<td>Real -13,12</td>
</tr>
<tr>
<td></td>
<td>123.45</td>
<td>Real 123,45</td>
</tr>
<tr>
<td></td>
<td>0.123</td>
<td>Real 0,123</td>
</tr>
<tr>
<td></td>
<td>-1.23E-3</td>
<td>Real -0,00123</td>
</tr>
<tr>
<td>Dual number</td>
<td>2#0111_1110 or 126</td>
<td>126</td>
</tr>
<tr>
<td>Octal number</td>
<td>8#123 or 83</td>
<td>83</td>
</tr>
<tr>
<td>Hexadecimal number</td>
<td>16#123 or 291</td>
<td>291</td>
</tr>
<tr>
<td>BOOL</td>
<td>0 and 1</td>
<td>Boolean TRUE and FALSE values</td>
</tr>
<tr>
<td></td>
<td>TRUE and FALSE</td>
<td></td>
</tr>
<tr>
<td>STRING</td>
<td>'ABC'</td>
<td>Character string ABC</td>
</tr>
<tr>
<td>WSTRING</td>
<td>ABC&quot;</td>
<td>2-byte-character string ABC</td>
</tr>
<tr>
<td>TIME</td>
<td>T#12.3ms or</td>
<td>Time duration of 12,3 milliseconds</td>
</tr>
<tr>
<td></td>
<td>TIME#12.3ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T#12h34m or</td>
<td>Time duration of 12 hours and 34 minutes</td>
</tr>
<tr>
<td></td>
<td>T#12h_34m</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T#-4m</td>
<td>Negative time duration of 4 minutes</td>
</tr>
<tr>
<td>Constant Data Type</td>
<td>Example</td>
<td>Meaning</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE#1995-12-24 or D#1995-12-24</td>
<td>Date 24.12.1995</td>
</tr>
<tr>
<td>TIME_OF_DAY</td>
<td>TOD#12:05:14.56 or TIME_OF_DAY#12:05:14.56</td>
<td>12 hours 05 minutes and 14,56 seconds PM</td>
</tr>
<tr>
<td>DATE_AND_TIME</td>
<td>DT#1995-12-24-12:05:14.56 or DATE_AND_TIME#1995-12-24-12:05:14.56</td>
<td>Date and time: 12 hours 05 minutes and 14,56 seconds PM on 24.12.1995</td>
</tr>
</tbody>
</table>

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**

```plaintext
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**

```plaintext
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 function block cannot be called, only instances can be called. The function block has a ‘memory’, i.e. all data (input, output, local) will keep its 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 output 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 variables 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:

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Required character set</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Lower case</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3a</td>
<td>Number sign (#)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3b</td>
<td>Pound sign (£)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
### Table 1: Character Set Features

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>4a</td>
<td>Dollar sign ($)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4b</td>
<td>Currency sign</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5a</td>
<td>Vertical bar (</td>
<td>)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5b</td>
<td>Exclamation mark (!)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Subscript delimiters:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6a</td>
<td>brackets [ ]</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>or</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6b</td>
<td>parentheses ( )</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

### Table 2: Identifier Features

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Upper case and numbers</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Upper and lower case, numbers, embedded underlines</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>Upper and lower case, numbers, leading or embedded underlines</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

### Table 3: Comment Features

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Comments</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

### Table 4: Numeric Literals

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Integer literals</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Real literals</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>Real literals with exponents</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>Base 2 literals</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>Base 8 literals</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>Base 16 literals</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>Boolean zero and one</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td>Boolean FALSE and TRUE</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Table 5: Character String Literal Features

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Empty string (length zero)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>String of length one containing the single character A</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>String of length one containing the <code>space</code> character</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>String of length one containing the <code>single quote</code> character</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>String of length two containing CR and LF</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>String of length five which would print as <code>1.00</code></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 6: Two Character Combinations in Character Strings

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

Table 7: Duration Literal Features

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a</td>
<td>Duration literals without underlines:</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>1b</td>
<td>Short prefix</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>Long prefix</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2a</td>
<td>Duration literal with underlines</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2b</td>
<td>Short prefix</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Long prefix</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

No. | Description                                      | Yes | No |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Date literals (long prefix: DATE#)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Date literals (short prefix: D#)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>Time of day literals (long prefix: TIME_OF_DAY#)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>Time of day literals (short prefix: TOD#)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
### Table 8: Date and Time of Day Literals

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Date and time literals (long prefix: DATE_AND_TIME#)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>Date and time literals (short prefix: DT#)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

### Table 9: Elementary Data Types

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

### Table 10: Data Type Declaration Feature

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Direct derivation from elementary types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Enumerated data types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>Subrange data types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>Array data types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>Structured data types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Description</td>
<td>Initial value</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>---------------</td>
<td>-----</td>
<td>----</td>
</tr>
<tr>
<td>BOOL, SINT, INT DINT, LINT,</td>
<td>0</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>USINT, UINT, UDINT, ULLINT</td>
<td>0</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>BYTE, WORD, DWORD, LWORD</td>
<td>0</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>REAL, LREAL</td>
<td>0.0</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>TIME</td>
<td>T#0s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>D#0001-01-01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME_OF_DAY</td>
<td>TOD#00:00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE_AND_TIME</td>
<td>DT#0001-01-01-00:00:00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRING</td>
<td>` (the empty string)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 11: Default Initial Values

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initialization of directly derived types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Initialization of enumerated data types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>Initialization of subrange data types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>Initialization of array data types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>Initialization of structured data types</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>Initialization of derived structured data types</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 12: Data Type Initial Value Declaration Features

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

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

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>VAR</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>VAR_INPUT</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>VAR_OUTPUT</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Keyword</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----</td>
<td>----</td>
</tr>
<tr>
<td>VAR_IN_OUT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>VAR_EXTERNAL</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>VAR_GLOBAL</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>VAR_ACCESS</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>RETAIN</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>CONSTANT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>AT</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 14: Variable keywords for variable declaration

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Declaration of directly represented, non-retentive variables</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Declaration of directly represented, retentive variables</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Declaration of locations of symbolic variables</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Array location assignment</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>Automatic memory allocation of symbolic variables</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Array declaration</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Retentive array declaration</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Declaration of structured variables</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 15: Variable type assignment features

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

Table 16: Variable initial value assignment features
Table 17: Graphical negation of Boolean signals

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Negated input</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Negated output</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 18: Use EN input an ENO output

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Use of EN and ENO</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Use of EN and ENO</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>FBD without EN and ENO</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 19: Typed and overloaded functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Overloaded functions (non type-dependent)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Typed functions</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 20: Type conversion function features

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><em><em>TO</em></em>*</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TRUNC</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>BCD_TO_**</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>*_TO_BCD</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ABS</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>SQRT</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>LN</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>LOG</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>EXP</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>SIN</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>COS</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td>TAN</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>ASIN</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>ACOS</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>11</td>
<td>ATAN</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 21: Standard functions of one numeric variable

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Symbol</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>ADD</td>
<td>+</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>13</td>
<td>MUL</td>
<td>*</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>14</td>
<td>SUB</td>
<td>-</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>15</td>
<td>DIV</td>
<td>/</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>16</td>
<td>MOD</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>17</td>
<td>EXPT</td>
<td>**</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>18n</td>
<td>MOVE</td>
<td>:=</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>18s</td>
<td>:=</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 22: Arithmetic standard functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SHL</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SHR</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ROR</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ROL</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 23: Standard bit shift functions
<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>AND</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>OR</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>XOR</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>NOT</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 24: Standard bitwise Boolean functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SEL</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2a</td>
<td>MAX</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2b</td>
<td>MIN</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>LIMIT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MUX</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 25: Standard selection functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>GT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>GE</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>EQ</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>LE</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>LT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>NE</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 26: Standard comparison functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LEN</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>LEFT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RIGHT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>MID</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CONCAT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>INSERT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>DELETE</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>REPLACE</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>FIND</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 27: Standard character string functions
### Table 28: Functions of time data types

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Operation</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ADD</td>
<td>TIME + TIME = TIME</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>TOD</td>
<td>TIME + TIME = TOD</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>DAT</td>
<td>TIME + TIME = DAT</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>SUB</td>
<td>TIME - TIME = TIME</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>DATE</td>
<td>DATE - DATE = TIME</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TOD</td>
<td>TOD - TIME = TOD</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>TOD</td>
<td>TOD - TOD = TIME</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td>DAT</td>
<td>DAT - TIME = DAT</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>DAT</td>
<td>DAT - DAT = TIME</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>MUL</td>
<td>TIME * ANY_NUM = TIME</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>11</td>
<td>DIV</td>
<td>TIME / ANY_NUM = TIME</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>12</td>
<td>CONCAT</td>
<td>DATE TOD = DAT</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

#### Type conversion functions

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Operation</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>DATE_AND_TIME_TO_TIME_OF_DAY</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>14</td>
<td>DATE_AND_TIME_TO_DATE</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

### Table 29: Standard bistable function blocks

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SEL</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>RS</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SEMA</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

### Table 30: Functions of enumerated data types

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RETAIN qualifier on internal variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>RETAIN qualifier on output variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>RETAIN qualifier on internal function blocks</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4a</td>
<td>Input/output declaration (textual)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4b</td>
<td>Input/output declaration (graphical)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5a</td>
<td>Function block instance name as input (textual)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5b</td>
<td>Function block instance name as input (graphical)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>No.</td>
<td>Description</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>-----</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>6a</td>
<td>Function block instance name as input/output (textual)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6b</td>
<td>Function block instance name as input/output (graphical)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7a</td>
<td>Function block instance name as external variable (textual)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7b</td>
<td>Function block instance name as external variable (graphical)</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 31: Function block declaration features

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R_TRIG</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F_TRIG</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 32: Standard edge detection function blocks

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R_TRIG</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>F_TRIG</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 33: Standard counter function blocks

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TP (Pulse)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2a</td>
<td>TON (on-delay)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2b</td>
<td>T---0 (on-delay)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3a</td>
<td>TOF (off-delay)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3b</td>
<td>0---T (off-delay)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RTC (real-time clock)</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 34: Standard timer function blocks
<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RETAIN qualifier on internal variable</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>RETAIN qualifier on output variable</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>RETAIN qualifier on internal function blocks</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4a</td>
<td>Input/output declaration (textual)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4b</td>
<td>Input/output declaration (graphical)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5a</td>
<td>Function block instance name as input (textual)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5b</td>
<td>Function block instance name as input (graphical)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6a</td>
<td>Function block instance name as input/output (textual)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6b</td>
<td>Function block instance name as input/output (graphical)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7a</td>
<td>Function block instance name as external variable (textual)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7b</td>
<td>Function block instance name as external variable (graphical)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8a</td>
<td>Textual declaration of: - rising edge inputs</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8b</td>
<td>- falling edge inputs</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>9a</td>
<td>Graphical declaration of: - rising edge inputs</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>9b</td>
<td>- falling edge inputs</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>Formal input and output parameters</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>11</td>
<td>Declaration of directly represented, non-retentive variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>12</td>
<td>Declaration of directly represented, retentive variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>13</td>
<td>Declaration of locations of symbolic variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>14</td>
<td>Array location assignment</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>15</td>
<td>Initialization of directly represented, non-retentive variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>16</td>
<td>Initialization of directly represented, retentive variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>17</td>
<td>Location and initial value assignment to symbolic variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>18</td>
<td>Array location assignment and initialization</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>19</td>
<td>Use of directly represented variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>20</td>
<td>VAR_GLOBAL .. END_VAR Declaration within a PROGRAM</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>21</td>
<td>VAR_ACCESS .. END_VAR Declaration within a PROGRAM</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 35: Program declaration features
<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Step graphical</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Step textual</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3a</td>
<td>Step flag general form</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3b</td>
<td>Step flag - direct connection of boolean variable</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>Step elapsed time</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 36: Step features

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Transition condition using ST language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Transition condition using LD language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>Transition condition using FBD language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>Use of connector</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4a</td>
<td>Transition condition using LD language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4b</td>
<td>Transition condition using FBD language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>Textual transition in ST</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>Textual transition in IL</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>Transition name</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7a</td>
<td>Transition condition using LD language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7b</td>
<td>Transition condition using FBD language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7c</td>
<td>Transition condition using IL language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7d</td>
<td>Transition condition using ST language</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 37: Transitions and Transition conditions

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>boolean variable as action</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2l</td>
<td>graphical declaration in LD language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2s</td>
<td>inclusion of SFC elements in action</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2f</td>
<td>graphical declaration in FBD language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3s</td>
<td>textual declaration in ST language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3i</td>
<td>graphical declaration in IL language</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 38: Declaration of actions
<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>action block</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>concatenated action blocks</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>textual step body</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>action block <code>d</code> field</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 39: Step/action association

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>qualifier as per 2.6.4.4</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>action name</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>boolean indicator variables</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>IL language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>ST language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>LD language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>FBD language</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td>action blocks in ladder diagrams</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>action block in function block diagrams</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 40: Action block features

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

Table 41: Action qualifiers

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>single sequence</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2a</td>
<td>divergence of sequence selection (left-to-right)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2b</td>
<td>divergence of sequence selection (with priorities)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2c</td>
<td>divergence of sequence selection (with mutual exclusion)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>Convergence of sequence evolution</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Table 42: Sequence evolution

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>simultaneous sequence divergence</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>simultaneous sequence convergence</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5a</td>
<td>sequence skip (left-to-right)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5b</td>
<td>sequence skip (with priorities)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5c</td>
<td>sequence skip (with mutual exclusion)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6a</td>
<td>sequence loop (left-to-right)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6b</td>
<td>sequence loop (with priorities)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6c</td>
<td>sequence loop (with mutual exclusion)</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>directional arrows</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 43: Instruction list (IL) operators

<table>
<thead>
<tr>
<th>No.</th>
<th>Operator</th>
<th>Modifiers</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LD</td>
<td>N</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ST</td>
<td>N</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>S</td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>AND</td>
<td>N,(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>&amp;</td>
<td>N,(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>OR</td>
<td>N,(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>XOR</td>
<td>N,(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ADD</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>SUB</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>MUL</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>DIV</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>GT</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>GE</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>EQ</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>NE</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>LE</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>LT</td>
<td>(</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>JMP</td>
<td>C, N</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>CAL</td>
<td>C, N</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>RET</td>
<td>C, N</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>)</td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
### Table 44: Function block invocation features for IL language

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CAL with input list</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>CAL with load/store of inputs</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Use of input operators</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

### Table 45: Operators of the ST language

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

### Table 46: ST language statements

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Assignment</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Function block invocation and FB output usage</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RETURN</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IF</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>CASE</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>FOR</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>WHILE</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>REPEAT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>EXIT</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Empty Statement</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
### Table 47: Representation of lines and block

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Horizontal lines:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ISO/IEC 646 ‘minus’ character</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>graphic or semigraphic</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>Vertical lines:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ISO/IEC 646 ‘vertical line’ character</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>graphic or semigraphic</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>Horizontal/vertical connection:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>ISO/IEC 646 characters</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Connected and non-connecte corners:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ISO/IEC 646 characters</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Blocks with connecting lines</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>ISO/IEC 646 characters</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Connectors using ISO/IEC 646 characters</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Connector, Continuation of a connected line</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>graphic or semigraphic</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

### Table 48: Graphic execution control elements

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Unconditional Jump</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FBD language</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Conditional Jump (FBD language)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Conditional Jump (LD language)</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Conditional Return</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>LD language</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Unconditional Return</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Alternative Representation in LD language</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>from Function</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>from Function Block</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

ACR-View IEC PLC Tools
<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Left power rail</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Right power rail</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 49: Power rails

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Horizontal link</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>vertical link with attached horizontal links</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 50: Link Elements

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normally open contact</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Normally closed contact</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>Positive transition-sensing contact</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>Negative transition-sensing contact</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 51: Contacts

<table>
<thead>
<tr>
<th>No.</th>
<th>Description</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Coil</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td>Negated Coil</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>3</td>
<td>SET (latch) coil</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>4</td>
<td>RESET (unlatch) coil</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>Retentive (Memory) coil</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>6</td>
<td>SET retentive (Memory) coil</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>RESET retentive (Memory) coil</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>8</td>
<td>Positive transition-sensing coil</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>Negative transition-sensing coil</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Clause</th>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5.1</td>
<td>Error handling procedures</td>
<td>see next chapter</td>
</tr>
<tr>
<td>2.1.1</td>
<td>National characters used</td>
<td>see table 1 above</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Maximum length identifiers</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>Significant length identifiers</td>
<td>64</td>
</tr>
<tr>
<td>2.1.5</td>
<td>Maximum comment length</td>
<td>&gt;512</td>
</tr>
<tr>
<td>2.2.3.1</td>
<td>Range of values of duration</td>
<td>+/- 24,85 days</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Range of values for variables of type TIME</td>
<td>+/- 24,85 days</td>
</tr>
<tr>
<td></td>
<td>Precision of representation of seconds</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>in type</td>
<td>TIME_OF_DAY and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATE_AND_TIME</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Maximum</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- number of array subscripts</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>- array size</td>
<td>&lt; 4KB per POU</td>
</tr>
<tr>
<td></td>
<td>- number of structure elements</td>
<td>&lt; 8KB per POU</td>
</tr>
<tr>
<td></td>
<td>- structure size</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- number of variables per declaration</td>
<td></td>
</tr>
<tr>
<td>2.3.3.1</td>
<td>Maximum number of enumerated values</td>
<td>&lt; 64 KB per POU</td>
</tr>
<tr>
<td>2.3.3.2</td>
<td>Default maximum length of STRING variables</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Maximum permissible length of STRING variables</td>
<td>253 [see note 1]</td>
</tr>
<tr>
<td>2.4.1.1</td>
<td>Maximum number of hierarchical levels</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Logical or physical mapping</td>
<td></td>
</tr>
<tr>
<td>2.4.1.2</td>
<td>Maximum number of subscripts</td>
<td>-</td>
</tr>
<tr>
<td>Clause</td>
<td>Parameter</td>
<td>Values</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>Maximum number of subscript values</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Maximum number of levels of structures</td>
<td>&gt;512</td>
</tr>
<tr>
<td>2.4.2</td>
<td>Initialization of system inputs</td>
<td>The value of the system inputs corresponds to their physical values</td>
</tr>
<tr>
<td>2.4.3</td>
<td>Maximum number of variables per declaration</td>
<td>&lt; 64 KB per POU</td>
</tr>
<tr>
<td>2.5</td>
<td>Information to determine execution times of program organization units</td>
<td>No</td>
</tr>
<tr>
<td>2.5.1.1</td>
<td>Method of function representation</td>
<td>Textual</td>
</tr>
<tr>
<td>2.5.1.3</td>
<td>Maximum number of function specifications</td>
<td>limited only by available memory</td>
</tr>
<tr>
<td>2.5.1.5</td>
<td>Maximum number of inputs of extensible functions</td>
<td>IL: 2, LD/FBD: unlimited</td>
</tr>
<tr>
<td>2.5.1.5.1</td>
<td>Effects of type conversions on accuracy</td>
<td>Truncated</td>
</tr>
<tr>
<td>2.5.1.5.2</td>
<td>Accuracy of functions of one variable</td>
<td>Currently not supported</td>
</tr>
<tr>
<td></td>
<td>Implementation of arithmetic functions</td>
<td></td>
</tr>
<tr>
<td>2.5.2</td>
<td>Maximum number of function blocks and instantiations</td>
<td>ca. 8000</td>
</tr>
<tr>
<td>2.5.2.3.3</td>
<td>PVmin, PVmax of counters</td>
<td>minimum/maximum value of respective data type</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Program size limitations</td>
<td>limited only by available memory</td>
</tr>
<tr>
<td>2.6</td>
<td>Timing and postability effects of execution control elements</td>
<td>-</td>
</tr>
<tr>
<td>2.6.2</td>
<td>Precision of step elapsed time</td>
<td>-</td>
</tr>
<tr>
<td>2.6.3</td>
<td>Maximum number of steps per SFC</td>
<td>-</td>
</tr>
<tr>
<td>2.6.4</td>
<td>Action control mechanism</td>
<td>-</td>
</tr>
<tr>
<td>2.6.4.2</td>
<td>Maximum number of action blocks per step</td>
<td>-</td>
</tr>
<tr>
<td>2.6.5</td>
<td>Graphic indication of step state Transition clearing time Maximum width of diverge/converge constructs</td>
<td>-</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Content of RESOURCE libraries</td>
<td>-</td>
</tr>
<tr>
<td>Clause</td>
<td>Parameter</td>
<td>Values</td>
</tr>
<tr>
<td>--------</td>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Maximum number of tasks</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Task interval resolution</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pre-emptive or non-pre-emptive scheduling</td>
<td></td>
</tr>
<tr>
<td>3.3.1</td>
<td>Maximum length of expressions</td>
<td>unlimited</td>
</tr>
<tr>
<td></td>
<td>Partial evaluation of Boolean expressions</td>
<td>no</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Maximum length of statements</td>
<td>Unlimited</td>
</tr>
<tr>
<td>3.3.2.3</td>
<td>Maximum number of CASE selections</td>
<td>Unlimited</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Graphic/semigraphic representation</td>
<td>Graphic</td>
</tr>
<tr>
<td></td>
<td>Restrictions on network topology</td>
<td></td>
</tr>
<tr>
<td>4.1.3</td>
<td>Evaluation order of feedback loops</td>
<td>-</td>
</tr>
</tbody>
</table>

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

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

- Prioritized transitions in a selection divergence
- Side effects in evaluation of transition condition
- Action control contention error
- "Unsafe" or "Unreachable" SFC
- Data type conflict in VAR_ACCESS
- Tasks require too many processor resources
  - Execution deadline not met
  - Other task scheduling conflicts
- Numerical result exceeds range for data type
- Division by zero
  - Invalid data type for operation
    - Return from function without value assigned
    - Iteration fails to terminate
- Same identifier as connector label and element name
- Uninitialized feedback variable
- Numerical result exceeds range for data type
  - Division by 0

### Syntax error reported for initialization in declaration; ignored at runtime

<table>
<thead>
<tr>
<th>2.6.3</th>
<th>Side effects in evaluation of transition condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.6.4.5</td>
<td>Action control contention error</td>
</tr>
<tr>
<td>2.6.5</td>
<td>&quot;Unsafe&quot; or &quot;Unreachable&quot; SFC</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Data type conflict in VAR_ACCESS</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Tasks require too many processor resources</td>
</tr>
<tr>
<td></td>
<td>Execution deadline not met</td>
</tr>
<tr>
<td></td>
<td>Other task scheduling conflicts</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Numerical result exceeds range for data type</td>
</tr>
<tr>
<td></td>
<td>Scan via functions</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Division by zero</td>
</tr>
<tr>
<td></td>
<td>Invalid data type for operation</td>
</tr>
<tr>
<td>3.3.2.1</td>
<td>Return from function without value assigned</td>
</tr>
<tr>
<td>3.3.2.4</td>
<td>Iteration fails to terminate</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Same identifier as connector label and element name</td>
</tr>
<tr>
<td>4.1.4</td>
<td>Uninitialized feedback variable</td>
</tr>
<tr>
<td>4.1.5</td>
<td>Numerical result exceeds range for data type</td>
</tr>
<tr>
<td></td>
<td>Division by 0</td>
</tr>
</tbody>
</table>

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 optimization 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.
<table>
<thead>
<tr>
<th>POU Change</th>
<th>POU's Variables Reset?</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Program</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Declaration</td>
<td>YES</td>
<td>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.</td>
</tr>
<tr>
<td>Code</td>
<td>NO</td>
<td>Code changes never lead to a reset of any variable values.</td>
</tr>
<tr>
<td><strong>Function Block</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Declaration</td>
<td>YES</td>
<td>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.</td>
</tr>
<tr>
<td>Code</td>
<td>NO</td>
<td>Code changes never lead to a reset of any variable values.</td>
</tr>
<tr>
<td>Resource Global Declarations</td>
<td>YES</td>
<td>Variables in the VAR_GLOBAL section will be reset. Again, this does not include globally defined function block instances (see above).</td>
</tr>
<tr>
<td>Functions</td>
<td>-</td>
<td>Strictly, functions are also POUs. Since they are stateless, they need not be treated by Online Edit, however.</td>
</tr>
</tbody>
</table>

**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_hhmssErrorlog.txt and will be stored in the current project directory.
# Reference Listings

## Keywords (by category)

### IEC 61131 Standard Function Blocks
ACR-View implements the following function blocks of IEC 61131-3:

- CTD
- CTU
- CTUD
- F_TRIG
- R_TRIG
- RS
- SR
- TOF
- TON
- TP

### IEC 61131-3 Standard Functions
ACR-View implements the following functions of IEC 61131-3:

- ABS
- ACOS
- AND
- ASIN
- ATAN
- CONCAT
- COS
- DELETE
- EQ
- EXP
- FIND
- GE
- GT
- INSERT
- LE
IEC61131-3 Operations

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

- ADD
- ADD (time)
- DIV
- DIV (time)
- MUL
- MUL (time)
ACR-View Functions and Function Blocks

The following functions and function blocks are provided by ACR-View in addition to IEC 61131-3:

- GetTaskInfo
- GetTime
- GetVarData
- GetVarFlatAddress

Data Types

The following elementary data types are defined by IEC 61131-3:

- 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 IEC 61131-3:

- 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
ENDRESOURCE
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
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
IEC 61131-3

* _to_bool
0 is converted to false, everything 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 IEC 61131-3 have a defined range of
values which is asymmetric, for example, SINT from -128..+127. As defined
by IEC 61131-3, 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 IEC 61131-3 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 IEC 61131-3.

**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 IEC 61131-3.

**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 IEC 61131-3.

**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 IEC 61131-3.
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:

```plaintext
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.

```plaintext
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**

```plaintext
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 IEC 61131-3.

**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 IEC 61131-3.

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 IEC 61131-3. 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 IEC 61131-3.

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 IEC 61131-3.

CASE
Though IF instructions may be nested, checking for one of many conditions can look quite complicated using IF. CASE, instead, can check for more than one value with one instruction. The ‘expression’ 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**

```plaintext
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 IEC 61131-3.

**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**

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>In1: STRING</td>
<td>First String</td>
</tr>
<tr>
<td>In2: STRING</td>
<td>Second String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRING CONCATENATION of both Strings</td>
</tr>
</tbody>
</table>

**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:

```plaintext
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.
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  C O U nte r pulse
- RESET: bool  R e s et c ounter
- PV: int  C ounter u p per limit

Outputs
- Q: bool  S ignals if c ounter h a s reached upper limit
- CV: int  C urrent c ounter v alue

Notes: Standardization—this function block is defined by IEC 61131-3.

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  C ounting impulses fo r c ounting u p, rising edge
- CD: bool  C ounting impulses fo r counting d own, rising edge
- RESET: bool  R e s et c ounter
- LOAD: bool  L o ad c ounter
- PV: int  L o ad v alue
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 IEC 61131-3.

**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 IEC 61131-3.

**DATE_AND_TIME**

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC 61131-3.

**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 IEC 61131-3.
**DINT**

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC 61131-3.

**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 IEC 61131-3.

**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 IEC 61131-3.

**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 IEC 61131-3.
ELSE

See CASE and IF

ELSIIF

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 IEC 61131-3 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 IEC 61131-3 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 IEC 61131-3 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 IEC 61131-3.
**END_VAR**

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC 61131-3.

**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 IEC 61131-3.

**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 : BOOL TRUE: 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:

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

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;
```
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

\[ \text{In} : \text{REAL} \]

Returns

\[ \text{REAL} : e^{\text{In}} \]

**EXPT**

Inputs:

\[ \text{In1} : \text{ANY}\_\text{REAL} \]

\[ \text{In2} : \text{ANY}\_\text{NUM} \]

Returns

\[ \text{ANY}\_\text{REAL} : \text{In1}^{\text{In2}} \]

**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

<table>
<thead>
<tr>
<th>VAR_INPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{X} : \text{BOOL F_EDGE} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VAR_OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{Z} : \text{BOOL} )</td>
</tr>
</tbody>
</table>

\( \text{Z} := \text{X AND Y} \); (* ST language example *)

END\_FUNCTION\_BLOCK

**F\_TRIG**

Inputs

\( \text{CLK} : \text{bool} \) Input operand whose falling edge is detected

Outputs

\( \text{Q} : \text{bool} \) 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 IEC 61131-3.

FALSE

Constant value of type BOOL.

FBD

FBD is the abbreviation of Function Block Diagram, one of the programming languages of IEC 61131-3.

FIND

Find one character string within another character string.

Inputs

In1: String Basic 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 IEC 61131-3.

FOR

With the FOR loop, a loop control variable will be set to a specified starting value, then incremented (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 IEC 61131 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 (STRUCTs, ARRAYs, 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:

![Change function type dialog](image)

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

**FUNCTION BLOCK**

IEC 61131-3 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

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

Notes: Standardization—this function is defined by IEC 61131-3.

**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 IEC 61131-3.
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 IEC 61131-3.

**IF**

The IF-instruction has following syntax:

**IF** expression **THEN** Block

{ **ELSEIF** 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 instruction after END_IF.

Note: It is recommended to use the absolute value ABS() of a floating point number if a comparison 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 IEC 61131-3.

IL
IL is the abbreviation of Instruction List, one of the programming languages of IEC 61131-3.

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 IEC 61131-3 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 Character 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' characters of 'IN1', the complete string 'IN2' and the rest of 'IN1'.

Notes: Standardization—this function is defined by IEC 61131-3.

**INT**

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC 61131-3.

**Interval**

This keyword is defined by IEC 61131-3 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 IEC 61131-3.

**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 IEC 61131-3.

**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 IEC 61131-3.

**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 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 IEC 61131-3.

**LD (Ladder Diagram)**

LD is the abbreviation of Ladder Diagram, one of the programming languages of IEC 61131-3.

**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 IEC 61131-3.

**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 IEC 61131-3.

**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 IEC 61131-3.

**LINT**

This is the name of an elementary data type, which is defined by IEC 61131-3, 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 IEC 61131-3.

**LT**

Inputs

- IN1: ANY  Input 1
- IN2: ANY  Input 2

Returns
**BOOL**

TRUE if Input 1 is less than Input 2

Notes: Standardization—this function is defined by IEC 61131-3.

**Lword**

This is the name of an elementary data type, which is defined by IEC 61131-3, 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 IEC 61131-3.

**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 IEC 61131-3.

**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 IEC 61131-3.
**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 IEC 61131-3.

**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 IEC 61131-3.
**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 IEC 61131-3.

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 IEC 61131-3.

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 IEC 61131-3. 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 address of a variable, the address 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(Reset)**

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 IEC 61131-3.

---

**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**

```plaintext
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 IEC 61131-3.

---

**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 IEC 61131-3 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 IEC 61131-3.

**REPLACE**

**Inputs**

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

**Returns**

- **STRING**  
  New compositred.
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 IEC 61131-3.

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 IEC 61131-3.

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 IEC 61131-3.
**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 function block sets the output CDT to the input PDT if EN=1. Otherwise CDT is invalid.

**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(set)**

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

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.

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.

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.

The function “SetSystemDateAndTime” sets the actual system time in IDT. Notes: Standardization: this function block is not defined by IEC 61131-3.

SFC

SFC is the abbreviation of Sequential Function Chart, one of the programming languages of IEC 61131-3.

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 IEC 61131-3.
**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 IEC 61131-3

**Signed_toUnsigned**

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 IEC 61131-3 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 IEC 61131-3.

**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 IEC 61131-3.

**STEP**

This keyword is defined by IEC 61131-3 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 IEC 61131-3.

**STRING**

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC 61131-3.

**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 character or the word is finished.

Examples

'1' → -1

'213hallo' → 213

'23.5' → 23

**BYTE, DWORD, WORD and USINT, UINT, UDINT**

The conversion is analogous 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 IEC 61131-3 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 IEC 61131-3.

**TIME_OF_DAY**

See Elementary Data Types

Notes: Standardization—this is a data type defined by IEC 61131-3.

**TIME_TO_***

Inputs
original data type time

Returns
converted data type *

The function block converts 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 IEC 61131-3. 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.

![Diagram](image)

Inputs:
IN: Start condition
PT: time Initial time value

Outputs

Reference Listings 129
Q: bool binary state of the timer
ET: time current time value

Notes: Standardization—this function block is defined by IEC 61131-3

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.

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 IEC 61131-3.

**Transition**

This keyword is defined by IEC 61131-3 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 IEC 61131-3.

**TYPE**

See Declaration Sections and Derived Data Types

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC 61131-3.
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 IEC 61131-3.

**VAR_ACCESS**

This keyword is defined by IEC 61131-3 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 IEC 61131-3.

**VAR_OUTPUT**

See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC 61131-3.
**VAR_IN_OUT**
See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC 61131-3.

**VAR_GLOBAL**
See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC 61131-3.

**VAR_EXTERNAL**
See Declaration Sections

Notes: This is a keyword only for declaration parts of POUs. This is defined by IEC 61131-3.

**VARINFO**

VARINFO is defined as

```
VARINFO : Struct
    TYP   : UINT;
    SIZE  : UINT;
    PROG  : UINT;
    SG  : 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 IEC 61131-3.

**WITH**
This keyword is defined by IEC 61131-3 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 IEC 61131-3.

**WSTRING**
See Elementary Data Types
Notes: Standardization—this is a data type defined by IEC 61131-3.

**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 IEC 61131-3.

**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 IEC 61131-3.

**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:
  - 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 IEC 61131-3 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 as 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 or 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**

```plaintext
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;
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).

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

```plaintext
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
```

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.

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

```plaintext
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**

```plaintext
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**

```plaintext
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**

```plaintext
FUNCTION_BLOCK Example
VAR_INPUT
  In1 : int;
  In2 : int;
END_VAR
FUNCTION_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**

```plaintext
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:

```plaintext
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:
```plaintext
TYPE
    Mystruct : struct end_struct;
END_TYPE
```

Allowed:
```plaintext
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:

```plaintext
TYPE
  MyInt1, MyInt2, MyInt3 : int;
END_TYPE
```

Allowed:

```plaintext
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 IEC 61131-3. 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 IEC 61131-3 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**
```plaintext
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**

```plaintext
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
```plaintext
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
```plaintext
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
```plaintext
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
```plaintext
FUNCTION_BLOCK Fb1
  VAR_INPUT
    InParam1 : int;
    InParam2 : int;
END_VAR
```
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 initialization.

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;
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 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**

```plaintext
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**

```plaintext
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 := IntVar2 ) (* Correct: every formal IN_OUT parameter of FB1 is supplied with an actual parameter *)
  (* Code *)
END_PROGRAM
```
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;
  END_TYPE
  VAR_GLOBAL
    GlobVar : StructType;  (* Not allowed because StructType is not known in other POU's *)
  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 addresses divisible by 4.

**L12996**
Unknown command: `<Command>`.
An unknown command line argument has been used with ITLINK.

**L12997**
Unknown 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 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.

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 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
```pascal
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
```pascal
TYPE
  DayOfWeek : STRUCT
    Name : STRING;
    DayNumber : UINT;
  END_STRUCT;

  DayDescriptions : ARRAY[1..100] OF DayOfWeek; (* Error: Day of Week is a complex data type.
  Presence : STRUCT
    Name : STRING;
    OursPerDay : ARRAY[1..31] OF UINT; (* Error: ARRAY is a complex data type.
    Structures 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*)

```plaintext
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

C 10092
Memory allocation failure.

C 10093
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.

C 10094
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.

C 10095
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.

C 10100
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.

C 10108
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 reason 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 Commissioning
- ALT+4: Output
- Ctrl+Enter: Fullscreen

**Insert Variable Submenu**
- ALT+SHIFT+V: All Variables
- ALT+SHIFT+H: 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**
- CTRL+ALT+S: Insert Step/Transition
- CTRL+ALT+L: Insert Step/Transition left
- CTRL+ALT+R: Insert Step/Transition right
- CTRL+ALT+J: Insert Jump
- CTRL+ALT+B: Insert Functionblock
- CTRL+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

) (Right-paranthesis-operator) .................. 86
* _TO ** ........................................ 128
* _to _bool ...................................... 87
ABS ............................................. 87
ABS_DINT ...................................... 87
ABS_DINT_FBD ................................ 87
ABS_INT ........................................ 87
ABS_INT_FBD .................................. 87
ABS_REAL ...................................... 87
ABS_REAL_FBD ................................ 87
ABS_SINT ...................................... 87
ABS_SINT_FBD ................................ 87
ABS_UDINT .................................... 87
ABS_UDINT_FBD ................................ 87
ABS_UINT ...................................... 87
ABS_UINT_FBD ................................ 87
ACOS ........................................... 87
ACOS_REAL ..................................... 87
ACOS_REAL_FBD ................................ 87
ACTION ......................................... 88
Active Document Server .................. 51, 52
ADD ............................................. 88
ADD (time) ...................................... 88
Add files ....................................... 13
Add Task ......................................... 10
ADD_DINT ...................................... 88
ADD_DINT_FBD ................................ 88
ADD_INT ........................................ 88
ADD_INT_FBD .................................. 88
ADD_REAL ...................................... 88
ADD_REAL_FBD ................................ 88
ADD_SINT ...................................... 88
ADD_SINT_FBD ................................ 88
ADD_TIME ...................................... 88
ADD_TIME_FBD ................................ 88
ADD_UDINT .................................... 88
ADD_UDINT_FBD ................................ 88
ADD_UINT ...................................... 88
ADD_UINT_FBD ................................ 88
ADD_USINT .................................... 88
ADD_USINT_FBD ................................ 88
Adding a Library to a project .............. 53
Adding input or output to compound block ..................................................... 45
Alias names ..................................... 33
AND .............................................. 88
AND_BOOL ...................................... 88
AND_BOOL_FBD ................................ 88
AND_BYTE ....................................... 88
AND_BYTE_FBD ................................ 88
AND_DWORD .................................... 88
AND_DWORD_FBD ................................ 88
AND_WORD ....................................... 88
AND_WORD_FBD ................................ 88
ANDN ........................................... 88
ANDN_BOOL .................................... 88
ANDN_BYTE ..................................... 88
ANDN_DWORD .................................. 88
ANDN_WORD ..................................... 88
ANY ............................................. 89
ANY_BIT ........................................ 89
ANY_DATE ....................................... 89
ANY_INT ......................................... 89
ANY_NUM ......................................... 89
ANY_REAL ......................................... 89
ARRAY ........................................... 89, 90
ASIN ............................................. 90
ASIN_REAL ......................................... 90
ASIN_REAL_FBD ................................ 90
Assignment ..................................... 90
AT ................................................. 91
ATAN ............................................. 91
ATAN_REAL ......................................... 91
ATAN_REAL_FBD ................................ 91
Automatic positioning of the caret ...... 36
Block specific help ............................ 31
Block Type Program Function          Block ........................................... 57
BOOL ........................................... 91
Bool_to _* ........................................ 91
BOOL_TO_BYTE ................................ 91
BOOL_TO_BYTE_EN ................................ 91
BOOL_TO_DINT ................................ 91
BOOL_TO_DINT_EN ................................ 91
BOOL_TO_DWORD ................................ 91
BOOL_TO_DWORD_EN ................................ 91
BOOL_TO_INT .................................. 91
BOOL_TO_REAL ................................ 91
BOOL_TO_REAL_EN ................................ 91
BOOL_TO_SINT .................................. 91
BOOL_TO_SINT_EN ................................ 91
BOOL_TO_STRING ................................ 91
BOOL_TO_STRING_EN ................................ 91
BOOL_TO_TIME .................................. 91
BOOL_TO_TIME_EN ................................ 91
BOOL_TO_UDINT ................................ 91
BOOL_TO_UDINT_EN ................................ 91
BOOL_TO_UINT .................................. 91
BOOL_TO_UINT_EN ................................ 91
BOOL_TO_USINT .................................. 91
BOOL_TO_USINT_EN ................................ 91
BOOL_TO.Word .................................. 88
BOOL_TO_WORd .................................. 91
BOOL_TO_WOR_d .................................. 91
BOO_L_TO_WOR_d_En ................................ 91
Breakpoints .................................. 78, 79
Browser Introduction ......................... 7
Build active resource ......................... 10
BY ............................................. 92
BYTE ........................................... 92
BYTE_TO_BOOL ................................ 87, 92
BYTE_TO_BOOL_EN ................................ 87, 92
BYTE_TO_DINT ................................ 92
BYTE_TO_DINT_EN ................................ 92
BYTE_TO_DWORD ................................ 92
BYTE_TO_DWORD_EN ................................ 92
BYTE_TO_INT ................................... 92
BYTE_TO_IN T_EN ................................ 92
BYTE_TO_REAL .................................. 92
BYTE_TO_REAL_EN ................................ 92
BYTE_TO_SINT .................................. 92
BYTE_TO_SIN T_EN ................................ 92
BYTE_TO_STRING ................................ 92
BYTE_TO_STRING_EN ................................ 92
BYTE_TO_TIME .................................. 92
BYTE_TO_TIME_EN ................................ 92
BYTE_TO_UDINT ................................ 92
BYTE_TO_UDINT_EN ................................ 92

<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
</tr>
<tr>
<td>Index</td>
</tr>
<tr>
<td>Index</td>
</tr>
<tr>
<td>Index</td>
</tr>
<tr>
<td>Index</td>
</tr>
<tr>
<td>Index</td>
</tr>
<tr>
<td>Index</td>
</tr>
<tr>
<td>Index</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>92</th>
<th>93</th>
</tr>
</thead>
<tbody>
<tr>
<td>BYTE_TO_UDINT_EN</td>
<td>BYTE_TO_UINT_EN</td>
</tr>
<tr>
<td>BYTE_TO_UART</td>
<td>BYTE_TO_USINT_EN</td>
</tr>
<tr>
<td>BYTE_TO_WORD</td>
<td>BYTE_TO_WORD_EN</td>
</tr>
<tr>
<td>C10006</td>
<td>C10007</td>
</tr>
<tr>
<td>C10008</td>
<td>C10009</td>
</tr>
<tr>
<td>C10010</td>
<td>C10012</td>
</tr>
<tr>
<td>C10014</td>
<td>C10015</td>
</tr>
<tr>
<td>C10017</td>
<td>C10019</td>
</tr>
<tr>
<td>C10020</td>
<td>C10021</td>
</tr>
<tr>
<td>C10024</td>
<td>C10025</td>
</tr>
<tr>
<td>C10026</td>
<td>C10028</td>
</tr>
<tr>
<td>C10030</td>
<td>C10031</td>
</tr>
<tr>
<td>C10034</td>
<td>C10035</td>
</tr>
<tr>
<td>C10036</td>
<td>C10038</td>
</tr>
<tr>
<td>C10043</td>
<td>C10045</td>
</tr>
<tr>
<td>C10046</td>
<td>C10047</td>
</tr>
<tr>
<td>C10049</td>
<td>C10055</td>
</tr>
<tr>
<td>C10057</td>
<td>C10060</td>
</tr>
<tr>
<td>C10063</td>
<td>C10064</td>
</tr>
<tr>
<td>C10067</td>
<td>C10068</td>
</tr>
<tr>
<td>C10069</td>
<td>C10070</td>
</tr>
<tr>
<td>C10075</td>
<td>C10078</td>
</tr>
<tr>
<td>C10083</td>
<td>C10084</td>
</tr>
<tr>
<td>C10092</td>
<td>C10093</td>
</tr>
<tr>
<td>C10094</td>
<td>C10095</td>
</tr>
<tr>
<td>C10096</td>
<td>C10097</td>
</tr>
<tr>
<td>C10098</td>
<td>C10099</td>
</tr>
<tr>
<td>C10100</td>
<td>C10101</td>
</tr>
<tr>
<td>C10102</td>
<td>C10103</td>
</tr>
<tr>
<td>C10104</td>
<td>C10105</td>
</tr>
<tr>
<td>C10110</td>
<td>C10112</td>
</tr>
<tr>
<td>C10113</td>
<td>C10114</td>
</tr>
<tr>
<td>C10115</td>
<td>C10120</td>
</tr>
<tr>
<td>C11001</td>
<td>C11007</td>
</tr>
<tr>
<td>CAL</td>
<td>CALCN</td>
</tr>
<tr>
<td>Caret and selection</td>
<td>Caret navigation</td>
</tr>
<tr>
<td>Caret position by selected moves</td>
<td>CASE</td>
</tr>
<tr>
<td>catalog</td>
<td>Catalog</td>
</tr>
</tbody>
</table>
DINT_TO_USINT_EN .......................... 97, 124
 dint_TO_WORD ................................. 97
DINT_TO_WORD_EN ............................. 97
DINT_TO_WORD ................................. 97
Directly represented variables .............. 18
DIV ............................. 97
DIV (time) ........................................ 97
DIV_DINT ....................................... 97
DIV_DINT_FBD ...................... 97, 97
DIV_INT ......................................... 97
DIV_INT_FBD .................................. 97
DIV_REAL ......................................... 97
DIV_REAL_FBD .................................. 97
DIV_SINT ......................................... 97
DIV_SINT_FBD .................................. 97
DIV_UDINT ....................................... 97
DIV_UINT ......................................... 97
DIV_UINT_FBD .................................. 97
DIV_USINT ....................................... 97
DIV_USINT_FBD .................................. 97
DO .................................................. 97
Download ........................................ 11
DS .................................................. 97
DT .................................................. 97
DWORD ............................................ 97
DWORD_TO_BYTE ................................ 97
DWORD_TO_BYTE_EN ........................... 87, 97
DWORD_TO_BOOL ................................ 87, 97, 97
DWORD_TO_BOOL_EN ......................... 87, 97
DWORD_TO_DINT ................................ 97
DWORD_TO_DINT_EN ............................ 97
DWORD_TO_DINT ............................... 97
DWORD_TO_INT .................................. 97
DWORD_TO_INT_EN ............................ 97
DWORD_TO_REAL ...................... 97, 97, 97
DWORD_TO_REAL_FBD ....................... 97, 97
DWORD_TO_REAL ................................ 97
DWORD_TO_REAL_EN .......................... 97, 97
DWORD_TO_SINT ...................... 97, 97, 97
DWORD_TO_SINT_EN .......................... 97, 97
DWORD_TO_STRING ................................ 97
DWORD_TO_TIME ................................ 97
DWORD_TO_TIME_EN ......................... 97, 97
DWORD_TO_UBOOL ................................ 97
DWORD_TO_UDINT ...................... 97, 97, 97
DWORD_TO_UDINT_EN ....................... 97, 97
DWORD_TO_UDINT ............................. 97
DWORD_TO_USINT ...................... 97, 97, 97
DWORD_TO_USINT_EN ....................... 97, 97
DWORD_TO_WORD ...................... 97, 97, 97
DWORD_TO_WORD_EN ....................... 97, 97
Edit resource .................................. 9
Editor depending Shortcuts .................. 182
Elementary Data Types ...................... 17
ELSE .............................................. 98
ELSIIf ............................................. 98
EN .................................................. 98
END_ACTION .................................. 98
END_CASE ....................................... 98
END_CONFIGURATION ....................... 98
END_FOR ....................................... 98
END_FUNCTION .................................. 98
END_FUNCTION_BLOCK ....................... 99
END_IF ............................................ 99
END_PROGRAM .................................. 99
END_REPEAT ..................................... 99
ENDRESOURCE .................................. 99
END_STEP ....................................... 99
END_STRUCT .................................... 99
END_TRANSITION ............................... 99
END_TYPE ....................................... 99
END_VAR ....................................... 100
END_WHILE .................................... 100
ENO ............................................... 100
EQ ............................................... 100
EQ_BOOL_FBD .................................. 100
EQ_BYTE_FBD .................................. 100
EQ_DINT_FBD .................................. 100
EQ(DWORD) FBD ............................... 100
EQ(DWORD) FBD .................................. 100
EQ(DWORD) FBD .................................. 100
EQ(DWORD) FBD .................................. 100
EQ(DWORD) FBD .................................. 100
EQ(DWORD) FBD .................................. 100
EQ(DWORD) FBD .................................. 100
EQ(DWORD) FBD .................................. 100
EQ(DWORD) FBD .................................. 100
FND .............................................. 100
Error Logs ..................................... 80
ET .......................................... 100, 101
ETR .......................................... 100, 101
Event Task Run Control ..................... 100
Execution Order ................................ 30
EXIT .......................................... 101
EXP ............................................ 102
EXP_REAL ........................................ 102
Expressions in ST ............................. 21
EXP ............................................ 102
EXP_DINT ........................................ 102
EXP_INT ........................................ 102
EXP_REAL ........................................ 102
EXP_SINT ........................................ 102
Extensible inputs ............................ 31
EXPT ............................................ 102
F .............................................. 102
F EDGE ........................................... 102
F EDGE ........................................... 102
F EDGE ........................................... 102
F EDGE ........................................... 102
F EDGE ........................................... 102
F EDGE ........................................... 102
FALSE .......................................... 103
Fast navigation with the caret ............. 41
FILE .............................................. 103
FBD .............................................. 103
File .............................................. 9
File Operations ............................... 9
File-Pane ...................................... 8
FIND ............................................ 103
FIND ............................................ 103
FIND ............................................ 103
FIND ............................................ 103
Finding Errors in CFC ....................... 31
For ............................................. 103, 104
Force Variables ............................... 103, 46
FROM ............................................ 104
Function .................................... 25, 104, 105
FUNCTION BLOCK ............................ 105
Functionblock .................................. 25
Functionblocks .................................. 25
Functionblocks and Functions ............. 25
Functions ..................................... 25
Functions with negatable inputs .......... 31
Fundamentals for keyboard usage .......... 35
G10001 ......................................... 135
G10001 ......................................... 135
GE .............................................. 105
GE .............................................. 105
GE .............................................. 105
GE .............................................. 105
GE .............................................. 105
GE .............................................. 105
GE .............................................. 105
GE .............................................. 105
GE .............................................. 105
INT_TO_UINT_EN ...................... 109, 124
INT_TO_UDINT_EN .................. 109, 124
INT_TO_USINT_EN ...... 109, 124
INT_TO_WORD_EN ........... 109
GETSYSTEMDATEANDTIME ......... 105
GetTaskInfo ......................... 106
GetTime .................................. 106
GetVarData ............................ 106
GetVarFlatAddress ................. 107
Going Online .......................... 11
GT ........................................... 107
GT_BOOL_FBD ....................... 107
GT_BYTE_FBD ....................... 107
GT_DINT_FBD ....................... 107
GT_DWORD_FBD ..................... 107
GT_INT_FBD ......................... 107
GT_REAL_FBD ....................... 107
GT_SINT_FBD ....................... 107
GT_STRING_FBD ................. 107
GT_UDINT_FBD .................. 107
GT_USINT_FBD .................. 107
GT_WORD_FBD ..................... 107
Help-Pane ............................. 9
How to Read Error Message ...... 134
IEC61131 Standard Function Blocks ... 81
IEC61131-3 operations .......... 82
IEC61131-3 Standard Functions ... 81
IF 107, 108
IL 108
INITIAL_STEP ....................... 108
Inline edit at the caret position 42
INSERT ................................. 109
Insert connections by keyboard .. 43
Insertion of blocks by keyboard ... 42
Install a Library ..................... 53
Instructions in ST ............... 21
INT ......................................... 109
int_TO_BOOL ......................... 87, 109
INT_TO_BOOL_EN ............... 87, 109
INT_TO_BYTE_EN ................. 109
INT_TO_DINT ...................... 109
INT_TO_UDINT_EN .......... 109
INT_TO_DWORD .................. 109
INT_TO_DWORD_EN ........... 109
INT_TO_REAL ..................... 109
INT_TO_REAL_EN ............... 109
int_TO_sint ...................... 109
INT_TO_SINT_EN ................. 109
INT_TO_STRING_EN .......... 109
INT_TO_TIME_EN .............. 109
int_TO_udint .................. 109, 124
INT_TO_UDINT_EN .......... 109, 124
INT_TO_UINT_EN ............. 109, 124
int_TO_usint .................. 109, 124
INT_TO_USINT_EN .......... 109, 124
int_TO_WORD ...................... 109
INT_TO_WORD_EN .............. 109
Interval .............................. 109
Introduction CFC Editor 27
JMP ......................................... 109
JMP ......................................... 109
JMP ......................................... 109
Keyboard combinations for navigating the caret ................................ 43
L10001 .................................. 168
L10004 .................................. 168
L10026 .................................. 168
L10027 .................................. 169
L10029 .................................. 169
L10030 .................................. 169
L10031 .................................. 169
L10032 .................................. 169
L10033 .................................. 169
L10034 .................................. 170
L10035 .................................. 170
L10036 .................................. 170
L10063 .................................. 170
L10105 .................................. 170
L10106 .................................. 170
L12001 .................................. 171
L12002 .................................. 171
L12003 .................................. 171
L12005 .................................. 171
L12006 .................................. 171
L12007 .................................. 172
L12008 .................................. 172
L12064 .................................. 172
L12065 .................................. 172
L12066 .................................. 172
L12996 .................................. 172
L12997 .................................. 172
L12998 .................................. 172
L12999 .................................. 173
L13000 .................................. 173
L13001 .................................. 173
L15001 .................................. 173
Ladder Editor Online .......... 26
Ladder Logic introduction .... 23
LD .......................................... 109
LD (Ladder Diagram) .......... 110
LDN ......................................... 110
LE .......................................... 110
LE_BOOL_FBD ...................... 110
LE_BYTE_FBD ...................... 110
LE_DINT_FBD ...................... 110
LE_DWORD_FBD .................. 110
LE_INT_FBD ...................... 110
LE_REAL_FBD .................... 110
LE_SINT_FBD .................... 110
LE_STRING_FBD .................. 110
LE_TIME_FBD .................... 110
LE_UNIT_FBD ..................... 110
LE_USINT_FBD .................. 110
LE_WORD_FBD .................... 110
LEFT ...................................... 110
LEFT_DINT ......................... 110
LEFT_INT ......................... 110
LEFT_SINT ....................... 110
LEFT_STRING_FBD ........ 110
LEFT_UDINT ...................... 110
LEFT_UINT ...................... 110
LEFT_USINT ..................... 110
LEN ...................................... 110
LEN_STRING ...................... 110
Index 187
<table>
<thead>
<tr>
<th>Function Block Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>NE_DINT_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NE_DWORD_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NE_INT_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NE_REAL_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NE_SINT_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NE_STRING_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NE_TIME_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NE_UINT_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NE_USINT_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NE_WORD_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NEG</td>
<td>114</td>
</tr>
<tr>
<td>Nested Comments</td>
<td>57</td>
</tr>
<tr>
<td>Network</td>
<td>23</td>
</tr>
<tr>
<td>NOT</td>
<td>114</td>
</tr>
<tr>
<td>NOT_BOOL_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NOT_BYTE_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NOT_DWORD_FBD</td>
<td>114</td>
</tr>
<tr>
<td>NOT_WORD_FBD</td>
<td>114</td>
</tr>
<tr>
<td>OF</td>
<td></td>
</tr>
<tr>
<td>On</td>
<td>115</td>
</tr>
<tr>
<td>Online Change</td>
<td>79</td>
</tr>
<tr>
<td>Online Edit</td>
<td>80</td>
</tr>
<tr>
<td>OPC</td>
<td>115</td>
</tr>
<tr>
<td>OpenPCS Function Blocks</td>
<td>83</td>
</tr>
<tr>
<td>Operators</td>
<td>23</td>
</tr>
<tr>
<td>OR</td>
<td>115</td>
</tr>
<tr>
<td>OR_BOOL</td>
<td>115</td>
</tr>
<tr>
<td>OR_BYTE</td>
<td>115</td>
</tr>
<tr>
<td>OR_BYTE_FBD</td>
<td>115</td>
</tr>
<tr>
<td>OR_DWORD</td>
<td>115</td>
</tr>
<tr>
<td>OR_DWORD_FBD</td>
<td>115</td>
</tr>
<tr>
<td>OR_WORD</td>
<td>115</td>
</tr>
<tr>
<td>OR_WORD_FBD</td>
<td>115</td>
</tr>
<tr>
<td>ORN</td>
<td>115</td>
</tr>
<tr>
<td>ORN_BOOL_FBD</td>
<td>115</td>
</tr>
<tr>
<td>ORN_BYTE_FBD</td>
<td>115</td>
</tr>
<tr>
<td>ORN_DWORD_FBD</td>
<td>115</td>
</tr>
<tr>
<td>ORN_DWORD</td>
<td></td>
</tr>
<tr>
<td>ORN_WORD_FBD</td>
<td>115</td>
</tr>
<tr>
<td>ORN_WORD</td>
<td></td>
</tr>
<tr>
<td>Others</td>
<td>84</td>
</tr>
<tr>
<td>Output Window</td>
<td>6</td>
</tr>
<tr>
<td>Passing Output Parameters</td>
<td>56</td>
</tr>
<tr>
<td>POINTER</td>
<td>115</td>
</tr>
<tr>
<td>Positioning of the caret</td>
<td>35</td>
</tr>
<tr>
<td>POU</td>
<td>116</td>
</tr>
<tr>
<td>Print Form</td>
<td>51</td>
</tr>
<tr>
<td>Print IEC61131 Configuration</td>
<td>48</td>
</tr>
<tr>
<td>Printing CFC charts</td>
<td>29</td>
</tr>
<tr>
<td>Priority</td>
<td>116</td>
</tr>
<tr>
<td>PROGRAM</td>
<td>116</td>
</tr>
<tr>
<td>PT116</td>
<td></td>
</tr>
<tr>
<td>PV</td>
<td>116</td>
</tr>
<tr>
<td>Q1</td>
<td>116</td>
</tr>
<tr>
<td>QQ</td>
<td>116</td>
</tr>
<tr>
<td>QU</td>
<td>116</td>
</tr>
<tr>
<td>R(Action Qualifier)</td>
<td>116</td>
</tr>
<tr>
<td>R(eset)</td>
<td>117</td>
</tr>
<tr>
<td>R_EDGE</td>
<td>117</td>
</tr>
<tr>
<td>R_TRIG</td>
<td>117</td>
</tr>
<tr>
<td>R1</td>
<td>117</td>
</tr>
<tr>
<td>READ_ONLY</td>
<td>117</td>
</tr>
<tr>
<td>READ_WRITE</td>
<td>118</td>
</tr>
<tr>
<td>REAL</td>
<td>118</td>
</tr>
<tr>
<td>Real_to_*</td>
<td>118</td>
</tr>
<tr>
<td>REAL_TO_BOOL</td>
<td>87</td>
</tr>
<tr>
<td>REAL_TO_BOOL_EN</td>
<td>87</td>
</tr>
<tr>
<td>REBUILD ALL RESOURCES</td>
<td>11</td>
</tr>
<tr>
<td>REBUILD ALL RESOURCES</td>
<td>11</td>
</tr>
<tr>
<td>Release</td>
<td>119</td>
</tr>
<tr>
<td>REPEAT</td>
<td>119</td>
</tr>
<tr>
<td>REPLACE</td>
<td>119</td>
</tr>
<tr>
<td>Replacement of Blocks</td>
<td>30</td>
</tr>
<tr>
<td>Representation of the caret</td>
<td>35</td>
</tr>
<tr>
<td>Resource</td>
<td>120</td>
</tr>
<tr>
<td>Resource global variables</td>
<td>12</td>
</tr>
<tr>
<td>Resource Pane</td>
<td>8</td>
</tr>
<tr>
<td>Resources introduction</td>
<td>9</td>
</tr>
<tr>
<td>RET</td>
<td>120</td>
</tr>
<tr>
<td>RETAIN</td>
<td>120</td>
</tr>
<tr>
<td>RETC</td>
<td>120</td>
</tr>
<tr>
<td>Retain</td>
<td>120</td>
</tr>
<tr>
<td>Return</td>
<td>121</td>
</tr>
<tr>
<td>Right</td>
<td>121</td>
</tr>
<tr>
<td>RIGHT_DINT</td>
<td>121</td>
</tr>
<tr>
<td>RIGHT_INT</td>
<td>121</td>
</tr>
<tr>
<td>RIGHT_SINT</td>
<td>121</td>
</tr>
<tr>
<td>RIGHT_STRING_FBD</td>
<td>121</td>
</tr>
<tr>
<td>RIGHT_UDINT</td>
<td>121</td>
</tr>
<tr>
<td>RIGHT_UINT</td>
<td>121</td>
</tr>
<tr>
<td>RIGHT_USINT</td>
<td>121</td>
</tr>
<tr>
<td>ROL</td>
<td>121</td>
</tr>
<tr>
<td>ROL_BYTE</td>
<td>121</td>
</tr>
<tr>
<td>ROL_BOOL</td>
<td>121</td>
</tr>
<tr>
<td>ROL_BOOL_FBD</td>
<td>121</td>
</tr>
<tr>
<td>ROL_BYTE_FBD</td>
<td>121</td>
</tr>
<tr>
<td>ROL_DWORD</td>
<td>121</td>
</tr>
<tr>
<td>ROL_DWORD_FBD</td>
<td>121</td>
</tr>
<tr>
<td>ROL_WORD</td>
<td>121</td>
</tr>
<tr>
<td>ROL_WORD_FBD</td>
<td>121</td>
</tr>
<tr>
<td>ROR</td>
<td>121</td>
</tr>
<tr>
<td>ROR_BYTE</td>
<td>121</td>
</tr>
<tr>
<td>ROR_BYTE_FBD</td>
<td>121</td>
</tr>
<tr>
<td>ROR_DINT</td>
<td>121</td>
</tr>
<tr>
<td>ROR_DINT_FBD</td>
<td>121</td>
</tr>
<tr>
<td>ROR_DWORD</td>
<td>121</td>
</tr>
<tr>
<td>ROR_DWORD_FBD</td>
<td>121</td>
</tr>
<tr>
<td>S</td>
<td>122</td>
</tr>
<tr>
<td>S(Actuation Qualifier)</td>
<td>122</td>
</tr>
<tr>
<td>S(et)</td>
<td>122</td>
</tr>
<tr>
<td>S1</td>
<td>123</td>
</tr>
<tr>
<td>S100</td>
<td>135</td>
</tr>
<tr>
<td>S1001</td>
<td>135</td>
</tr>
<tr>
<td>S1002</td>
<td>135</td>
</tr>
<tr>
<td>S1003</td>
<td>136</td>
</tr>
<tr>
<td>S1004</td>
<td>136</td>
</tr>
<tr>
<td>S1005</td>
<td>136</td>
</tr>
<tr>
<td>S1006</td>
<td>136</td>
</tr>
<tr>
<td>S1007</td>
<td>136</td>
</tr>
<tr>
<td>S1008</td>
<td>136</td>
</tr>
<tr>
<td>S1009</td>
<td>136</td>
</tr>
<tr>
<td>S1010</td>
<td>136</td>
</tr>
<tr>
<td>S1011</td>
<td>136</td>
</tr>
<tr>
<td>S1012</td>
<td>137</td>
</tr>
<tr>
<td>S1013</td>
<td>137</td>
</tr>
<tr>
<td>S1014</td>
<td>137</td>
</tr>
<tr>
<td>S1015</td>
<td>137</td>
</tr>
<tr>
<td>S1016</td>
<td>137</td>
</tr>
<tr>
<td>S1017</td>
<td>138</td>
</tr>
<tr>
<td>S1018</td>
<td>138</td>
</tr>
<tr>
<td>S1019</td>
<td>138</td>
</tr>
<tr>
<td>Page number</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>138</td>
<td>S1020</td>
</tr>
<tr>
<td>138</td>
<td>S1021</td>
</tr>
<tr>
<td>138</td>
<td>S1022</td>
</tr>
<tr>
<td>139</td>
<td>S1023</td>
</tr>
<tr>
<td>139</td>
<td>S1024</td>
</tr>
<tr>
<td>139</td>
<td>S1025</td>
</tr>
<tr>
<td>139</td>
<td>S1026</td>
</tr>
<tr>
<td>140</td>
<td>S1027</td>
</tr>
<tr>
<td>140</td>
<td>S1028</td>
</tr>
<tr>
<td>140</td>
<td>S1029</td>
</tr>
<tr>
<td>140</td>
<td>S1030</td>
</tr>
<tr>
<td>140</td>
<td>S1031</td>
</tr>
<tr>
<td>140</td>
<td>S1032</td>
</tr>
<tr>
<td>141</td>
<td>S3000</td>
</tr>
<tr>
<td>141</td>
<td>S3001</td>
</tr>
<tr>
<td>141</td>
<td>S3002</td>
</tr>
<tr>
<td>142</td>
<td>S3003</td>
</tr>
<tr>
<td>142</td>
<td>S3004</td>
</tr>
<tr>
<td>142</td>
<td>S3005</td>
</tr>
<tr>
<td>142</td>
<td>S3006</td>
</tr>
<tr>
<td>143</td>
<td>S3007</td>
</tr>
<tr>
<td>143</td>
<td>S3008</td>
</tr>
<tr>
<td>143</td>
<td>S3009</td>
</tr>
<tr>
<td>143</td>
<td>S3010</td>
</tr>
<tr>
<td>144</td>
<td>S3011</td>
</tr>
<tr>
<td>144</td>
<td>S3012</td>
</tr>
<tr>
<td>144</td>
<td>S3014</td>
</tr>
<tr>
<td>144</td>
<td>S3016</td>
</tr>
<tr>
<td>144</td>
<td>S3017</td>
</tr>
<tr>
<td>145</td>
<td>S3018</td>
</tr>
<tr>
<td>145</td>
<td>S3019</td>
</tr>
<tr>
<td>145</td>
<td>S3020</td>
</tr>
<tr>
<td>145</td>
<td>S3021</td>
</tr>
<tr>
<td>145</td>
<td>S3022</td>
</tr>
<tr>
<td>145</td>
<td>S3023</td>
</tr>
<tr>
<td>145</td>
<td>S3024</td>
</tr>
<tr>
<td>145</td>
<td>S3025</td>
</tr>
<tr>
<td>146</td>
<td>S3026</td>
</tr>
<tr>
<td>146</td>
<td>S3027</td>
</tr>
<tr>
<td>146</td>
<td>S3028</td>
</tr>
<tr>
<td>146</td>
<td>S3029</td>
</tr>
<tr>
<td>147</td>
<td>S3030</td>
</tr>
<tr>
<td>147</td>
<td>S3031</td>
</tr>
<tr>
<td>147</td>
<td>S3032</td>
</tr>
<tr>
<td>147</td>
<td>S3033</td>
</tr>
<tr>
<td>147</td>
<td>S3034</td>
</tr>
<tr>
<td>148</td>
<td>S3035</td>
</tr>
<tr>
<td>148</td>
<td>S3036</td>
</tr>
<tr>
<td>148</td>
<td>S3037</td>
</tr>
<tr>
<td>148</td>
<td>S3038</td>
</tr>
<tr>
<td>148</td>
<td>S3039</td>
</tr>
<tr>
<td>148</td>
<td>S3040</td>
</tr>
<tr>
<td>149</td>
<td>S3041</td>
</tr>
<tr>
<td>149</td>
<td>S3042</td>
</tr>
<tr>
<td>149</td>
<td>S3043</td>
</tr>
<tr>
<td>149</td>
<td>S3044</td>
</tr>
<tr>
<td>149</td>
<td>S3045</td>
</tr>
<tr>
<td>149</td>
<td>S3046</td>
</tr>
<tr>
<td>150</td>
<td>S4000</td>
</tr>
<tr>
<td>150</td>
<td>S4001</td>
</tr>
<tr>
<td>150</td>
<td>S4002</td>
</tr>
<tr>
<td>151</td>
<td>S4003</td>
</tr>
<tr>
<td>151</td>
<td>S4004</td>
</tr>
<tr>
<td>151</td>
<td>S4005</td>
</tr>
<tr>
<td>151</td>
<td>S4006</td>
</tr>
<tr>
<td>151</td>
<td>S4007</td>
</tr>
<tr>
<td>151</td>
<td>S4008</td>
</tr>
<tr>
<td>151</td>
<td>S4009</td>
</tr>
<tr>
<td>152</td>
<td>S4010</td>
</tr>
<tr>
<td>152</td>
<td>S4011</td>
</tr>
<tr>
<td>152</td>
<td>S4012</td>
</tr>
<tr>
<td>152</td>
<td>S4013</td>
</tr>
<tr>
<td>152</td>
<td>S4014</td>
</tr>
<tr>
<td>152</td>
<td>S4015</td>
</tr>
<tr>
<td>153</td>
<td>S4016</td>
</tr>
<tr>
<td>153</td>
<td>S4017</td>
</tr>
<tr>
<td>153</td>
<td>S4018</td>
</tr>
<tr>
<td>153</td>
<td>S4019</td>
</tr>
<tr>
<td>153</td>
<td>S4020</td>
</tr>
<tr>
<td>153</td>
<td>S4021</td>
</tr>
<tr>
<td>153</td>
<td>S4022</td>
</tr>
<tr>
<td>154</td>
<td>S4023</td>
</tr>
<tr>
<td>154</td>
<td>S4024</td>
</tr>
<tr>
<td>154</td>
<td>S4033</td>
</tr>
<tr>
<td>154</td>
<td>S4034</td>
</tr>
<tr>
<td>155</td>
<td>S4035</td>
</tr>
<tr>
<td>155</td>
<td>S4036</td>
</tr>
<tr>
<td>155</td>
<td>S5000</td>
</tr>
<tr>
<td>155</td>
<td>S5001</td>
</tr>
<tr>
<td>155</td>
<td>S5002</td>
</tr>
<tr>
<td>155</td>
<td>S5003</td>
</tr>
<tr>
<td>156</td>
<td>S5004</td>
</tr>
<tr>
<td>156</td>
<td>S5005</td>
</tr>
<tr>
<td>156</td>
<td>S5006</td>
</tr>
<tr>
<td>156</td>
<td>S5007</td>
</tr>
<tr>
<td>156</td>
<td>S5008</td>
</tr>
<tr>
<td>156</td>
<td>S5009</td>
</tr>
<tr>
<td>157</td>
<td>S5010</td>
</tr>
<tr>
<td>157</td>
<td>S5011</td>
</tr>
<tr>
<td>157</td>
<td>S5012</td>
</tr>
<tr>
<td>157</td>
<td>S5013</td>
</tr>
<tr>
<td>158</td>
<td>S5014</td>
</tr>
<tr>
<td>158</td>
<td>S5015</td>
</tr>
<tr>
<td>158</td>
<td>S5016</td>
</tr>
<tr>
<td>158</td>
<td>S5017</td>
</tr>
<tr>
<td>159</td>
<td>S5018</td>
</tr>
<tr>
<td>159</td>
<td>S5019</td>
</tr>
<tr>
<td>159</td>
<td>S5020</td>
</tr>
<tr>
<td>159</td>
<td>S5021</td>
</tr>
<tr>
<td>159</td>
<td>S5022</td>
</tr>
<tr>
<td>160</td>
<td>S5023</td>
</tr>
<tr>
<td>160</td>
<td>S5024</td>
</tr>
<tr>
<td>160</td>
<td>S5025</td>
</tr>
<tr>
<td>160</td>
<td>S5026</td>
</tr>
<tr>
<td>160</td>
<td>S5027</td>
</tr>
<tr>
<td>160</td>
<td>S5028</td>
</tr>
<tr>
<td>161</td>
<td>S5029</td>
</tr>
<tr>
<td>161</td>
<td>S5030</td>
</tr>
<tr>
<td>162</td>
<td>S5031</td>
</tr>
<tr>
<td>162</td>
<td>S5032</td>
</tr>
<tr>
<td>162</td>
<td>S5033</td>
</tr>
<tr>
<td>163</td>
<td>S5034</td>
</tr>
<tr>
<td>163</td>
<td>S5035</td>
</tr>
<tr>
<td>164</td>
<td>S5036</td>
</tr>
<tr>
<td>164</td>
<td>S5037</td>
</tr>
<tr>
<td>165</td>
<td>S5038</td>
</tr>
<tr>
<td>165</td>
<td>S5039</td>
</tr>
<tr>
<td>166</td>
<td>S5040</td>
</tr>
<tr>
<td>166</td>
<td>S5041</td>
</tr>
<tr>
<td>167</td>
<td>S5042</td>
</tr>
<tr>
<td>167</td>
<td>S5043</td>
</tr>
<tr>
<td>167</td>
<td>S6002</td>
</tr>
<tr>
<td>167</td>
<td>S6004</td>
</tr>
<tr>
<td>167</td>
<td>S6005</td>
</tr>
<tr>
<td>80</td>
<td>Save System</td>
</tr>
<tr>
<td>80</td>
<td>SaveSystemCmd</td>
</tr>
<tr>
<td>123</td>
<td>SD</td>
</tr>
<tr>
<td>123</td>
<td>SEL</td>
</tr>
<tr>
<td>123</td>
<td>SEMA</td>
</tr>
</tbody>
</table>
usint_TO_BOOL .................................. 87, 132
USINT_TO_BOOL_EN ............................ 87, 132
usint_TO_BYTE .................................. 132
USINT_TO_BYTE_EN ...................................132
usint_TO_dint ..................................... 132
USINT_TO_DINT_EN ........................... 132
usint_TO_DWORD .................................. 132
USINT_TO_DWORD_EN ........................... 132
usint_TO_int ...................................... 132
USINT_TO_INT_EN .................................. 132
usint_TO_REAL .................................... 132
USINT_TO_REAL_EN ................................ 132
usint_TO_sint ..................................... 132
USINT_TO_SINT_EN ............................ 132
USINT_TO_STRING_EN........................ 132
USINT_TO_TIME_EN ............................. 132
usint_TO_udint ................................... 132
USINT_TO_UDINT_EN ......................... 132
usint_TO_uint ................................... 132
USINT_TO_UINT_EN ................................ 132
usint_TO_WORD .................................... 132
USINT_TO_WORD_EN ................................ 132
VAR................................................... 132
VAR_ACCESS...................................... 132
VAR_EXTERNAL .................................. 133
VAR_GLOBAL...................................... 133
VAR_IN_OUT...................................... 133
VAR_INPUT ....................................... 132
VAR_OUTPUT...................................... 132
Variablecatalog .................................... 14
Variablegrid......................................... 14
Variabletable ....................................... 14
VARINFO............................................ 133
Watch variables ................................... 46
Watching variables ................................ 11
Watchlist............................................ 47
WHILE............................................... 133
WITH................................................. 134
WORD................................................ 134
WORD_TO_BOOL .......................... 87, 134
WORD_TO_BOOL_EN..................... 87, 134
WORD_TO_BYTE .................................134
WORD_TO_BYTE_EN ........................... 134
WORD_TO_dint................................... 134
WORD_TO_DINT_EN ........................... 134
WORD_TO_DWORD ............................... 134
WORD_TO_DWORD_EN ....................... 134
WORD_TO_int .................................... 134
WORD_TO_INT_EN.............................. 134
WORD_TO_REAL .................................... 134
WORD_TO_REAL_EN............................ 134
WORD_TO_sint ................................... 134
WORD_TO_SINT_EN............................ 134
WORD_TO_STRING_EN........................ 134
WORD_TO_TIME_EN............................ 134
WORD_TO_udint ................................... 134
WORD_TO_UDINT_EN ......................... 134
WORD_TO_uint ................................... 134
WORD_TO_UINT_EN............................ 134
WORKING TO_usint ................................. 134
WORKING TO_USINT_EN..................... 134
Working with Blocks ............................. 27
Working with watchlists ........................ 47
WSTRING........................................... 134
XOR .................................................. 134
XOR_BOOL_EN ................................... 134
XOR_BOOL_FBD ................................... 134
XOR_BYTE_EN .................................... 134
XOR_BYTE_FBD ................................... 134
XOR_DWORD_EN ................................ 134
XOR_DWORD_FBD ............................... 134
XOR_WORD_EN .................................. 134
XOR_WORD_FBD ............................... 134
XORN ................................................ 134
XORN_BOOL_FBD ................................ 134
XORN_BYTE_FBD ............................... 134
XORN_DWORD_FBD ............................... 134
XORN_WORD_FBD ............................... 134