# Table of Contents

# Introducing the Automation Collaborative Platform

The Automation Collaborative Platform is a complete suite for building multi-process control projects and Human-Machine Interface applications. The Automation Collaborative Platform's environment includes development tools and other technologies simplifying design, development, and deployment of applications. You develop projects on a Windows development platform, from the Workbench and language editors.

The Workbench graphically represents and organizes devices, and networks. The development process consists of creating projects made up of devices, representing individual target nodes. The development environment is made up of multiple windows and tools:

- Solution Explorer
- Navigation Window
- Language Editors
- Dictionary
- Block Library
- Deployment View
- Controller Status
- Variable Dependencies
- Properties Window
- Collection Editor
- Locked Variables Viewer
- ISaVIEW

- Toolbox
- Variable Selector
- Block Selector
- Parameters View
- Document Generator
- Error list
- Find and Replace Utility
- Output Window
- Spy List
- Add-in Manager
- Description Window

# Solution Explorer

The Solution Explorer is an interface that displays a graphical view of solutions. The Solution Explorer helps manage two types of conceptual containers used by the workbench, the solutions and projects. Projects include sets of program source files and related metadata, such as references and build instructions. Solution contain elements that represent the references, data connections, folders, and files that are needed to create an application. A solution can house multiple projects and a project can house multiple programs. From the Solution Explorer, double-clicking project elements displays their contents in the appropriate editor or tool.

When creating new projects, the solution is automatically created as the container for your project. You can create standalone or temporary projects. A stand-alone project is any solution that contains only one project.

In a multi-project solution, the first project created is designated as the startup project and is displayed in bold font in the Solution Explorer. When you build or debug the solution, startup projects are run first. You can choose to set one or multiple start-up projects to run using the debugger.

The Solution Explorer displays the contents of solutions in a logically organized tree view, providing access to project elements. The physical locations of project files can differ from that represented in the tree view structure. From the tree view, you can conduct many project management tasks using contextual menus for project elements, including adding elements, cutting or copying elements, deleting or removing elements, moving elements, and renaming elements. The Solution Explorer toolbar provides access to available commands for selected elements. You can update the commands available in the toolbar by clicking project elements. You can select multiple project elements and execute batch commands using toolbar commands. When selecting multiple elements, the toolbar and contextual menu only display the options available to all selected files.

 Displays the Properties Window for the element selected

Toggles between simple and regular tree view. In simple view, the device, resource (if supported by the CAM), and library elements are removed from the structure.

 Refreshes the selected item

The Solution Explorer displays all related commands available for stand-alone or multi-project solutions. For stand-alone projects, you can choose to display or hide the solution container within the Solution Explorer.

**To access the Solution Explorer**

- From the View menu, click **Solution Explorer** (or press **Ctrl+Alt+L**).

The Solution Explorer is displayed.

**To cut or copy elements**

1. Select one or more elements, right-click the selection, then do one of the following:
   - To cut the selection, click **Cut**
   - To copy the selection, click **Copy.**
2. Right-click the required destination, then click **Paste**.

**To move elements**

You can move elements using the contextual menus. You can also drag project elements, changing their order of appearance.

1. To move elements using the contextual menus, perform the following:
   a) Select one or more elements, right-click the selection, and then click **Cut.**
   b) Right-click the required destination, then click **Paste**.
2. To move elements by dragging, select an element, then drag it to the required location.

**To delete or remove elements**

1. To permanently delete elements, select one or more elements, right-click the selection, then click **Delete**.
2. To remove elements, select one or more elements, right-click the selection, then click **Cut**.

**To rename elements**

- Right-click an element, click **Rename**, then type a new name in the space provided.

**To hide the solution container**

For stand-alone projects, you can choose to hide the solution container.

1. From the Tool menu, click **Options**

   The Options dialog box is displayed.

2. Expand **Projects**, then click **General**.

3. From the options displayed, clear *Always show solution*.

The solution container is no longer displayed in the Solution Explorer.

## See Also

Creating Projects
Adding New Projects

# Creating Projects

You can create projects in either the current solution or in a new solution. Project templates enable creating projects containing one or more devices as well as files and folder appropriate for the project type. When you select a project type, the available templates are displayed.

When creating a project for a new solution, you provide a name and storage location for the project. The directory structure for the solution is automatically created. The default solution name is the same as the project name. You can define unique names for the solution and project. When creating stand-alone projects, no solution is defined. You create projects based on templates. When creating a project, you can choose to add it to an existing solution or create a new solution. When creating a new solution, you can specify to create the directory structure for the solution. Project names must begin with a letter or underscore followed by letters, digits, and single underscores. Project names cannot contain the following characters:

| | |
|---|---|
| Pound (#) | Double quotation mark (") |
| Percent (%) | Less than (<) |
| Ampersand (&) | Greater than (>) |
| Asterisk (*) | Question mark (?) |
| Vertical bar (\|) | Forward slash (/) |
| Backslash (\) | Leading or trailing spaces (' ') |
| Colon (:) | Names reserved for Windows or DOS such as "nul", "aux", "con", "com1", and "lpt1" |

You can choose to display templates using small  or medium  icons.

**To create a project**

1. From File menu, point to **New**, then click **Project** (or press **Ctrl+Shift+N**).
2. From the New Project dialog box, select the type of project and a template.
3. Define the project name and storage location.
   a) In the *Name* field, type a unique project name.
   b) In the *Location* field, define the project folder by typing the path, selecting from the drop-down combo-box, or browsing for the require location.
4. Create a folder for the solution and define the solution name (optional).

a) Select **Create directory for solution**.

b) Type the desired solution name.

5. Click **OK**.

The project is created.

## Opening Projects and Solutions

You can open existing projects with their solutions. When opening an existing project within a solution, you add the project to the solution. You can also add projects using the contextual menu for the solution item within the Solution Explorer.

When opening solutions, all projects and files associated with the solution are opened.

When opening a project, you can choose to add the project to the solution that is currently open or close the current solutions and open the projects in a new solution.

**To open a project or solution from the Workbench**

1. From the File menu, point to **Open**, then click **Project/Solution** (or press **Ctrl+Shift+O**).
2. In the Open Project dialog box, locate the required project or solution.
3. Specify whether to add the project to the open solution or close the current solution before opening the project.
4. Click **Open**.

The project or solution is displayed.

**To open a project or solution from the Projects directory**

- From the Windows Explorer, access the Projects directory and perform one of the following:
  - Double-click the required *.isasln file.
  - Drag the *.acfproj or *.isasln file onto the desktop Workbench icon.

  The workbench opens displaying the required project or solution.

# Adding New Projects

You can create new projects for inclusion in open solutions. Project templates enable creating projects containing one or more devices as well as files and folders appropriate for the project type. Project templates are organized into types. When selecting a project type, the available templates are displayed.

When adding a project, you provide a name and storage location for the project. You add projects based on templates. Project names cannot contain the following characters:

| | |
|---|---|
| Pound (#) | Double quotation mark (") |
| Percent (%) | Less than (<) |
| Ampersand (&) | Greater than (>) |
| Asterisk (*) | Question mark (?) |
| Vertical bar (\|) | Forward slash (/) |
| Backslash (\) | Leading or trailing spaces (' ') |
| Colon (:) | Names reserved for Windows or DOS such as "nul", "aux", "con", "com1", and "lpt1" |

You can choose to display templates using large or small icons.

**To add a new project to an existing solution**

1. From the File menu, point to **Add**, then click **New Project**.
2. From the Add New Project dialog box, select the required project template.
3. In the *Name* field, type a unique name.
4. In the *Location* field, define the storage location for the project by typing the path or browsing to select an existing folder. When browsing, you can choose to make a new folder.
5. Click **OK**.

The project is displayed.

**To add a temporary project**

1. From the Tool menu, click **Options**

   The Options dialog box is displayed.

2. Expand **Projects**, then click **General**.

3. From the options displayed, clear *Save new projects when created.*

4. From the File menu, point to **Add**, then click **New Project**.

5. In the Add New Project dialog box, select a template, type the required information in the fields provided, and click **OK**.

The temporary project is added.

## Adding Existing Projects

You can add existing projects to open solutions. When adding projects to solutions, you can access projects from local or network directories.

**To add existing projects to a solution**

1. From the File menu, point to **Add**, then click **Existing Project**.

2. From the Add Existing Project dialog box, locate and select the required project file, then click **OK**.

The project is added to the open solution.

## Saving Changes to Solutions and Projects

You can save changes to projects and solutions. When closing projects and solutions, you are prompted to save changes to projects and solutions.

**To save changes to solutions**

1. From the File menu, click **Close Solution**.
2. In the save changes dialog box, click **Yes**.

**To save changes to items open in the workspace**

1. From the File menu, click **Close**.
2. In the save changes dialog box, click **Yes**.

## Solution Properties

You can manage builds using the following options available for solutions:

- Setting the start-up projects

**To access the Solution Property Pages**

1. In the Solution Explorer, select the solution element.
2. From the View menu, click **Property Pages**.

The Solution Property Pages dialog box is displayed.

## Setting Startup Projects

You can set projects to run when you start online debugging. You can also modify the order in which projects run during debugging. The startup feature is not available for simulation.

- Current selection, enables running only the project currently selected
- Single startup project, enables running a single specified project
- Multiple startup projects, enables running more than one project

Running multiple projects requires building the startup projects. You set the debugging order of projects by moving them up or down in the list. You also need to specify the action to apply to individual projects belonging to the solution when the debugger starts:

- None where the project remains in edit mode
- Start where the project runs
- Start without debugging where the project runs without debugging

Projects using startup options must respect the following folder hierarchy structure. This hierarchy is the default structure when creating projects in a solution.

Solution folder (*.isasln)
    Project1 folder (*.isaproj)
    Project2 folder (*.isaproj)
    Project3 folder (*.isaproj)

For projects not using this hierarchy structure, you need to manually build the structure by manually copying the projects, then adding the existing projects to the solution from the Solution Explorer using the contextual menu.

**To set startup projects**

1. In the Solution Explorer, select and right-click the solution element, then click **Set Startup Projects**.
2. From the Solution Property Pages dialog box, expand **Common Properties**, then click **Startup Project**.

3. Specify which projects to run and debug when the debugger starts, then click **OK.**
   - To run and debug the project currently selected in the Solution Explorer, click **Current selection.**
   - To run a single project within the solution, click **Single startup project**, then select the project from the drop-down combo-box.
   - To run multiple projects within the solution, click **Multiple startup projects**, then define debug order and action to apply to each project belonging to the solution.

   Projects run in the order of appearance in the list.

4. To reorder the projects in the list, select the individual projects, then click  or .

5. To save changes, click **Apply**.

## Setting Project Dependencies

**Note:** The Project Dependencies feature is not implemented for use.

For **ISaGRAF** projects, you add dependencies on libraries. For more information, refer to "Using a Library in a Project" for the respective Concrete Automation Model (CAM).

## Setting Configuration Properties

You can define how solutions and projects are built and run. For each project in the solution, you define properties for multiple configurations, including simulation and online configurations. You can define the build and platform options for individual configurations. You can also choose to set the options for all configurations.

When setting the configuration properties, you can define the following information:

- Configuration, enables selecting from the list of configurations:

| Option | Description |
|---|---|
| Active(*configuration*) | Enables defining the platform and build options for the configuration type currently selected in the Solution Configurations drop-down combo-box |
| Online | Enables defining the platform and build options for the online configuration type |
| Simulation | Enables defining the platform and build options for the simulation configuration type |
| All Configurations | Enables changing the platform and build options for all configurations types (simulation and online) for each project in the solution |

- Platform, enables choosing the development platform on which to run projects and solutions

  **ISaGRAF 6** supports the *Any CPU* platform only.

- Project Contexts, for each project listed, enables defining platforms and build options for each configuration:

| Project | Configuration | Platform | Build |
|---|---|---|---|
| Lists of projects making up the solution | Options for the configuration type (simulation or online) | Options for the development platform used for each project when running the solution | Indication of whether to build the project when building the solution |

You can also access the Configuration Manager where you can create and edit configurations.

**To set configuration properties**

1. From the Solution Explorer, click the solution element.

2. From the View menu, click **Property Pages**.

3. From the Solution Properties Pages dialog box, expand **Configuration Properties**, then click **Configuration**.

4. Define the properties for configurations.

   a) In the *Configuration* drop-down combo-box, select the required configuration.

   b) In the *Project contexts* table, for the required project, verify the configuration and platform displayed, then click the check box in the *Build* column.

5. Click **Apply**, then click **OK**.

The configuration properties are set for the solution.

## Configuration Manager

You can create and edit solution configurations. Changes made using the Configuration Manager are reflected in the Solution Property Pages. When creating and editing configurations, you need to define the following information:

- Active solution configuration, the available configurations. You can create solution configurations and rename existing ones.
- Active solution platform, the available platforms. **ISaGRAF 6** supports the *Any CPU* platform only. You can rename the existing solution platform.

  **ISaGRAF 6** supports the *Any CPU* platform type only.
- Project Contexts, for each project listed, enables defining platforms and build options for each configuration:

| Project | Configuration | Platform | Build |
|---|---|---|---|
| Lists of projects making up the solution | Options for the configuration type (simulation or online) | Options for the development platform used for each project when running the solution | Indication of whether to build the project when building the solution |

From the Configuration Manager you can also perform the following tasks:

- Create solution configurations
- Edit solution configurations
- Edit solution platforms

**To access the Configuration Manager**

1. From the Solution Explorer, click the solution element.
2. From the View menu, click **Property Pages**.
3. From the Solution Property Pages dialog box, click **Configuration Manager**.

The Configuration Manager is displayed.

**To set configuration properties**

**1.** From the Configuration Manager, define the properties for configurations.

**a)** In the *Active solution configuration* drop-down combo-box, select the required configuration.

**b)** In the *Project contexts* table, for the required project, verify the configuration and platform displayed, then click the check box in the *Build* column.

**2.** Click **Close**.

The configuration properties are reflected in the Solution property pages.

## See Also

Setting Configuration Properties

## Creating Solution Configurations

You can create solution configurations using the Configuration Manager. When creating solution configurations, you define the name and settings for the solution configurations. You also choose whether to create project configurations corresponding to the solution configuration.

**To create a solution configuration**

**1.** From the Configuration Manager, in the *Active Solution Configuration* drop-down combo-box, select **<New...>**.

**2.** From the New Solution Configuration dialog box, do the following:

   **a)** In the *Name* field, type a name for the build configuration.

   **b)** From the *Copy setting from* drop-down combo-box, copy the settings from another build configuration by selecting the configuration name.

   **c)** To create corresponding project configurations, select **Create new project configurations**.

**3.** Click **OK**.

The solution configuration is ready for use.

## Editing Solution Configurations

You edit solution configurations from the Configuration Manager. When editing solution configurations, you choose to rename or remove these.

**To edit a configuration name or remove a solution build configuration**

1. From the Configuration Manager dialog box, in the *Active Solution Configuration* drop-down combo-box, select **<Edit...>**
2. From the Edit Solution Configurations dialog box, do the following:
   - To change the name of a configuration, select the configuration name, click **Rename**, then type a new name.
   - To remove a configuration, select the configuration name, then click **Remove**.
3. Click **Close**.

The solution configuration is ready for use.

### Editing Solution Platforms

You can rename existing solution platforms from the Configuration Manager.

**To edit a solution platform name**

1. From the Configuration Manager, in the *Active solution platform* drop-down combo-box, select **<Edit...>**
2. From the Edit Solution Platforms dialog box, select the platform name, click **Rename**, then type a new name.
3. Click **Close**.

The solution platform name is displayed in the Solution Property Pages and Configuration Manager.

## Specifying Debug Source Files

**Note:** The Debug Source Files feature is not implemented for use.

# Navigation Window

The Navigation Window is a graphical environment enabling navigation through many aspects and elements making up projects. The available elements vary on the CAM. The environment provides a global view listing the devices contained in one or more projects within a solution. The navigation window consists of vertical links on the left pane and a breadcrumbs trail in the address field. When you click a specific vertical link, the view for that aspect or element is displayed in the workspace. For example, clicking a device instance displays the Device View in the workspace. From the navigation window, you can navigate to the following views:

- Deployment View
- Device View
- Bindings View (if supported by the CAM)
- I/O Device View (if supported by the CAM)
- POU instances
- Parameters View
- Dictionary instances

**To access the Navigation Window**

1. From the View menu, click **Navigation Window**.

The Navigation Window is displayed in the workspace.

**To access various views from the Navigation Window**

The initial aspects and elements displayed vary depending on the item selected in the Solution Explorer.

1. To access the Deployment View, from the Navigation Window, click **Deployment View**.

   The Deployment View is displayed in the workspace.

2. To access the Device View, click the Global arrow in the Navigation View, then click the required device from the available devices.

   The required device is displayed in the Device View.

3. To access the Bindings view, from the Navigation window, in the required Device section, click **Bindings**.

   The Bindings View is displayed in the workspace.

4. To access the I/O Device view, select the required resource in the Solution Explorer, then from the resource section in the Navigation Window, click **I/O Device**.

   The I/O Device view is displayed in the workspace.

5. To access a POU instance, from the Navigation Window, click the required POU.

   The POU is displayed in the language container.

6. To access the Parameters for a user-defined function or function block, select the required instance in the Solution Explorer, then click **Parameters** in the Navigation Window.

   The Parameters for the required user-defined function or function block are displayed.

7. To access Dictionary instances, select the required POU in the Solution Explorer, then from the Navigation Window, click **Global Variables** or **Local Variables**.

   The Dictionary instance is displayed in the workspace.

# Language Editor

The language editor is the environment where you develop the contents of POUs. You develop these POUs using language containers. Language containers hold elements of a given IEC 61131-3 or IEC 61499 programming language. A POU can only have one language container. Description containers hold non-semantic information. When building projects, the compiler excludes information from description and HMI containers.

From the language editor, you can edit multiple POUs simultaneously. Individual POUs are opened in separate workspaces each having a tab indicating the POU name. The tabs enable moving from one POU to another.

When working in the language editor, you can choose to expand the workspace to a full-screen view.

You can edit the contents of language containers in the editor workspace.

The document overview enables focusing on areas within the workspace. When clicking inside the document overview, the workspace displays the area inside the focus box, indicated with a blue outline. Using the focus box you can define the area to display, i.e., zoom, in the workspace. Decreasing the size of the focus box increases the zoom. Whereas, increasing the size of the focus box decreases the zoom. You can focus on another area within the document overview by clicking the location.

**To expand the workspace to the full-screen view**

- From the View menu, click **Full Screen**.

**To use the document overview**

1. From the View menu, click **Document Overview**.
2. To focus on an area of the workspace, click inside the document overview.

   The area inside the focus box is displayed within the workspace.
3. To define the focus area, do one of the following:

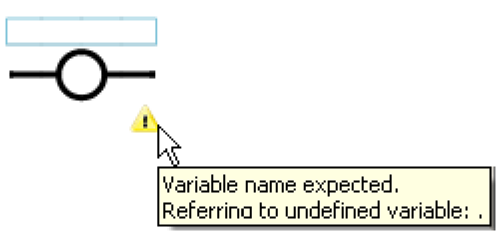
- Increase the zoom by decreasing the size of the focus box.
- Decrease the zoom by increasing the size of the focus box.
- Drag the focus box to another location within the workspace.

# Editing the Contents of Language Containers

You develop POUs using language containers. When developing POUs, you can only insert elements from the corresponding language Toolbox into the open language container. POUs can have only one language container.

- Selecting elements
- Inserting elements
- Inserting identifiers
- Inserting blocks
- Moving elements
- Shifting elements
- Resizing elements
- Deleting elements

For graphical POUs, the workbench displays an error symbol ( ) below elements having errors in the programming logic. Pausing on this symbol displays a description of the error.




**To select elements**

In the workspace, you can select individual or multiple elements within a language container. Selected elements are displayed with a colored handles. When selecting multiple elements, the handles of the first element are green and subsequent elements are turquoise.

When aligning multiple elements, the reference point differs depending on the programming language.

1. To select one element, click the element in the language container.
2. To select multiple elements, do one of the following:
   - Starting from empty workspace, drag the pointer over the elements.
   - While pressing SHIFT, use the pointer to select elements individually.

**To insert elements**

You can insert elements of a given language into its corresponding language container within the workspace.

- From the Toolbox, drag the element into the language container.

**To insert an identifier**

You can insert identifiers, i.e., variables, from the Dictionary. You can also create new variables, enter literal values into a POU, and access the parameters of functions or function blocks. When creating a new variable, you need to assign a unique name, specify its type, and define its scope in relation to the POU.

When inserting identifiers, you can choose to insert a constant or variable automatically via the Variable Selector.

1. From the Toolbox, drag the variable element into the language container.
2. From the Variable Selector, perform one of the following, then click **OK**.:
   - In the *Name* field, type a literal value.

     When inserting literal values that begin with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'
   - Select the required variable from the lists of variables.

The variable is displayed.

### To insert a block

You insert blocks into language containers of graphical programs from the Toolbox. Following insertion, you specify the type of block, i.e., operator, function, or function block, in the Block Selector accessed by double-clicking the block. For programs, the available items are operators (OPE), standard functions (SFU), standard function blocks (SFB), user IEC 61131-3 Functions (IFU), user IEC 61131-3 Function Blocks (IFB) and all "C" Functions (CFU) and Function Blocks (CFB) supported by the target type.

**1.** From the Toolbox, drag the block element into the language container.

**2.** In the Block Selector, in the list of blocks, locate the required block:

   **a)** To limit the blocks displayed, you can sort the block list and filter the block list.

   **b)** From the list of available blocks, select a block, then click **OK**.

### To move elements within a language container

You can move elements within a language container.

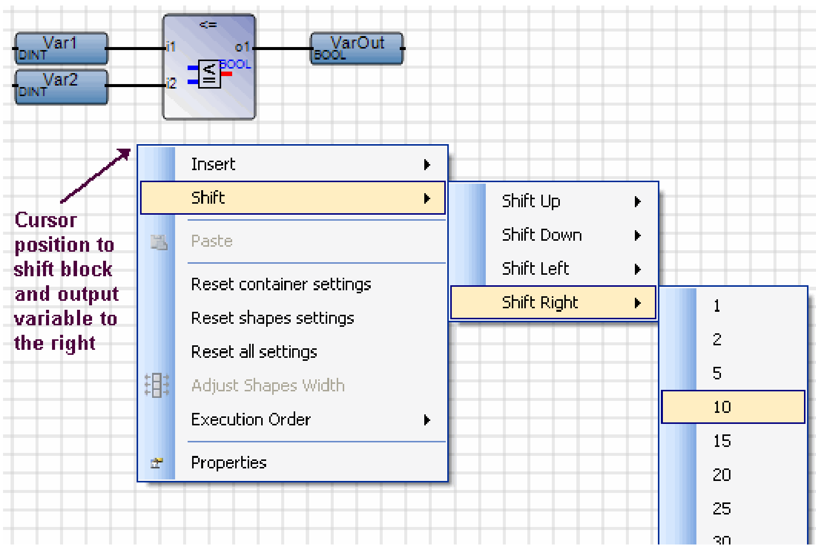
**1.** In the language container, select one or more elements.

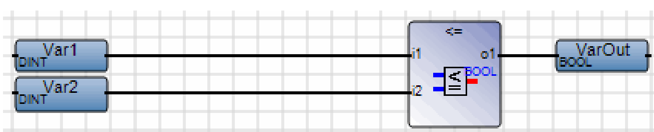
**2.** Drag the elements to another position.

### To shift elements within an FBD language container

You can shift elements within an FBD language container towards the left, towards the right, towards the top (up), or towards the bottom (down) by a number of grid spaces. When shifting elements towards the left or right, you displace all elements located to the right of the cursor position by a number of grid spaces from their leftmost edge. When shifting elements up or down, you displace all elements located below the cursor position by a number of grid spaces from their topmost edge.

The following example shows the location in which to place the cursor and the shift options enabling the displacement of the block and output variable towards the right by 10 grid spaces, in reference to the leftmost edge of the block. The input variables to the left of the cursor remain in the same location following the shift operation.




1. In the language container, right click to access the contextual menu at the location from which to reference the elements to displace.

   - To displace elements towards the left or right, place the cursor to the left of the required elements.
   - To displace elements towards the top (up) or the bottom (down), place the cursor above the required elements.

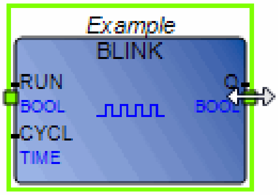
2. Right-click, point to **Shift**, then point to the required shift direction, and then click the number of grid spaces.

**To resize an element**

When developing POUs using certain graphical programming languages, you can change the dimensions of specific individual elements.

1. In the language container, select an element.

2. Place the cursor over the element handles, then drag to the required size.




**To delete elements**

You can delete one or more elements from a language container. Deleting a variable element does not remove the variable from the dictionary.

- In the language container, select one or more elements, then do one of the following:
  - Right-click the selection, then click **Delete**.
  - Press DELETE.

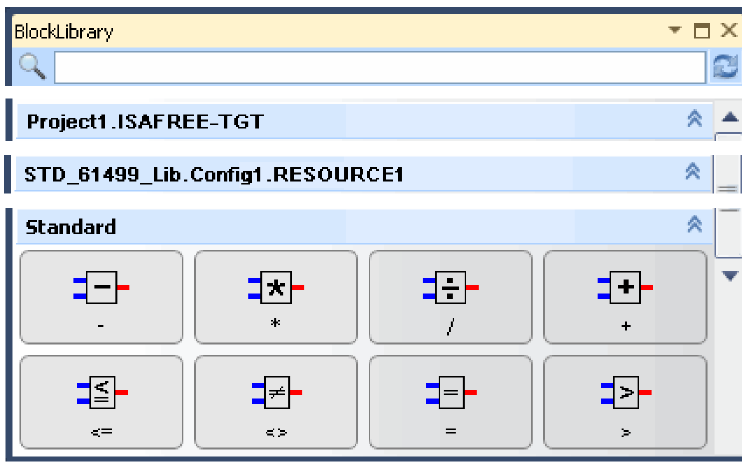
# Block Library

The block library provides a graphical view of all operators, functions, and function blocks available for the POUs of a project. When developing POUs, you can drag and drop blocks from the library to the language container. You can sort blocks displayed in the library according to alphabetical order, categories, or scope as well as limit a search based on names. You can also display the blocks in either tile or list views.

Target C Scope

Library Scope

Standard Scope




The following types of blocks may be available from the block library:

- Standard operators
- Standard functions
- Standard function blocks
- User IEC 61131 functions
- User IEC 61131 function blocks
- User C functions
- User C function blocks

- User functions and function blocks from a library

Blocks are sorted by scope:

- Standard blocks
- Library blocks (a scope for each library dependency)
- Target-specific C blocks

**To access the block library**

The block library displays the blocks applicable to the project template and target.

- From the View menu, click **Block Library**.

**Note:** The block library can also be accessed using the keyboard shortcut **Ctrl+Alt+T**.

**To insert a block in a POU**

- In the block library, locate the required block, then click and hold the mouse on the block while dragging to the destination in the POU container.

**To sort blocks in the library**

- Right-click in the block library window, then click **Category** or **Scope**.

**To limit searches**

You can perform searches for blocks by entering any part a block name. As you type text in the library search field, the library displays only the blocks containing these characters.

- In the field in the block library window, type the required text.

**To toggle the blocks view in the library**

- Right-click in the block library window, then click **Tile View** or **List View**.

# Deployment View

The deployment of a project constitutes the devices, networks, and connections making up the project. The Deployment view graphically displays the devices, networks, and connections of a project. From this view, you can manage the following aspects of a project:

- Devices
- Networks linking devices
- Connections between devices and networks

Devices displayed in the Deployment view are also present in the Solution Explorer. Therefore, modifications to the devices in the Deployment view are reflected in the Solution Explorer.

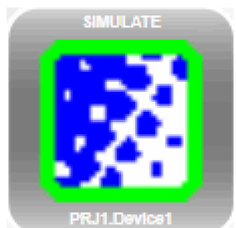
**To access the Deployment view**

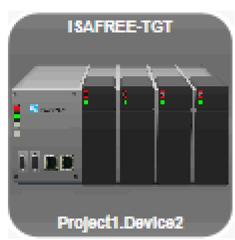
- In the **View** menu, click **Deployment View**.

The Deployment view is displayed in the workspace.

# Devices

A device corresponds to a programmable logic controller. A device must be connected to a network supporting the device's target type. In the Deployment view, a device is represented by a rounded rectangle containing the target name, device icon, and device source. The target name is indicated above the device icon. The device source is displayed as *ProjectName.DeviceName* and is indicated below the device icon. The following targets are supported in **ISaGRAF**:

<table>
<tr><th></th><th>Target</th><th>Device icon</th></tr>
<tr><td>ISaGRAF 3 Concrete Automation Model</td><td>SIMULATE</td><td>


</td></tr>
<tr><td>ISaGRAF 5 Concrete Automation Model</td><td>ISAFREE-TGT</td><td>


</td></tr>
<tr><td></td><td>SIMULATOR</td><td>


</td></tr>
</table>

While debugging, devices in the Deployment View are displayed using default or user-defined colors to represent the following different statuses:

| | |
|---|---|
| Break status | Displayed when the device encounters a breakpoint (if supported by the CAM) |
| Error status | Displayed when the device encounters an error |
| Idle status | Displayed when the device is idle |
| Offline status | Displayed when the device is offline |
| Run status | Displayed when the device is running |
| Stop status | Displayed when the device is stopped |
| Unknown status | Displayed when the device status is unknown |

**To add a new device from template**

1. Right-click in the Deployment view, and then click **Add New Device From Template**.

   The **Add New Project** dialog box appears.

2. From the **Add New Project** dialog box, select the required project template, then click **OK**.

   The device belonging to the new project is added to the Deployment view.

3. In the **Properties** window for the device, define the required properties.

**To add a new device**

1. Right-click in the Deployment view, point to **Add New Device**, then point to the required project, and then click the required target type.

   The new device is added to the Deployment view.

2. In the **Properties** window for the device, define the required properties.

**To delete a device**

1. In the **Deployment** view, click the device.

2. From the **Edit** menu, click **Delete**.

# Networks

Networks provide the means for communication between devices.The target attached to the device must support the network connected to the device. You define network properties at the time of creation. These properties are specific to the network type.

A project can have an unlimited number of networks.

In the Deployment view, networks are displayed as horizontal lines.

When multiple networks are defined (or if the target is not defined in the project), the workbench uses the first default network. When one is not defined, the workbench uses the second default network. When neither default networks are defined, the first network defined for the target is used.

**To add a network**

You define network properties at the time of creation.

1. Right-click in the Deployment view, then point to **Add New Network**, and then click the required network.

   The network is added to the workspace.

2. In the **Properties** window for the network, define the required properties.

**To delete a network**

You can delete networks from the Deployment View.

1. In the **Deployment** view, click the network element.

2. From the **Edit** menu, click **Delete**.

# Connections

Connections between networks and devices enable communications to flow. You need to connect each device to one or more networks. Similarly, a network can be linked to many devices.

In the Deployment view, connections are displayed as vertical lines. After connecting devices to networks, you can move these by dragging.

**To connect a device to a network**

You can create connections between devices and networks.

1. In the **Deployment** view, click on the network and drag to the device.
2. In the **Properties** window for the connection, define the required IP address for the connection.

**To delete a connection between a device and network**

You can remove existing connections between devices and networks.

- In the **Deployment** view, click the connection line and press DELETE.

## Deployment View Keyboard Shortcuts

The following keyboard shortcuts are available for use with the Deployment view.

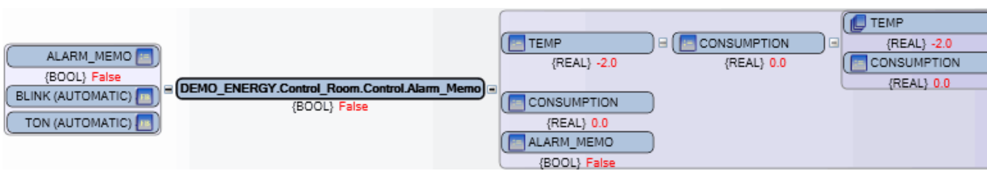
| | |
|---|---|
| Ctrl+= | Zoom in |
| Ctrl+- | Zoom out |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+Page Up | Jumps to the top of the language container |
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Ctrl+Home | Jumps to the top of the language container |
| Ctrl+End | Jumps to the bottom of the language container |
| Up Arrow | Scrolls up |
| Down Arrow | Scrolls down |
| Left Arrow | Scrolls left |
| Right Arrow | Scrolls right |
| Ctrl+Up Arrow | Scrolls up |
| Ctrl+Down Arrow | Scrolls down |
| Ctrl+Left Arrow | Scrolls left |
| Ctrl+Right Arrow | Scrolls right |

# Variable Dependencies

You can view the dependencies of a variable in both ascending and descending directions. Ascending dependencies display the variables affecting the variable while descending dependencies display the variables affected by the variable. These dependencies are displayed as structures leading to the right for ascending dependencies and to the left for descending dependencies. When viewing the dependencies of a variable, the variable identification indicates its source such as the program name, device name, and ending with the variable name.

You can view dependencies of variables while editing, debugging, or running online. While online, you can monitor and force the values of variables from the dependencies.

The following example shows the dependencies of the Alarm_Memo variable where the variables on the right, ascending, affect its value while the variable affects the values of the variables to the left, descending.




Descending Dependencies Ascending Dependencies

When viewing the dependencies of a variable, you can perform the following tasks:

- Display the dependencies of variables within the dependencies structures
- Add variables to a Spy list
- Access dictionary instances containing selected variables
- Access the POU where a variable is used
- Adjust the zoom factor for individual dependencies windows

Furthermore, you can also force the values of variables while debugging.

**To display the dependencies of a variable**

You can access the dependencies of variables from graphic programs or dictionary instances.

**Note:** Before accessing the dependencies of variables, you need to generate the cross references for a project.

- From a graphic program or a dictionary instance, right-click the variable, then click **Dependencies**.

The dependencies structure is displayed for the variable.

**To display the dependencies of a variable within the dependencies structures**

You can display the dependencies of variables from the ascending or descending structures.

- From the dependencies structure for a variable, double-click the variable from the ascending or descending structure for which to display the dependencies.

The dependencies structure for the selected variable is displayed in another window.

**To add a variable to a Spy List**

You can add variables from dependencies structures to a Spy List.

- From the dependencies structure for a variable, select the variable to add to the Spy List, right-click, and then click **Add to Spy List**.

The variable is added to the Spy List window.

**To force the value of a variable**

You can force, i.e., write, the value of a variable from the ascending or descending structures.

1. From the dependencies structure for a variable, select the variable for which to force the value, right-click, and then click **Write Variable**.
2. In the Write Logical Value dialog box, write the value for the variable.
3. To lock the value for the variable, click **Lock**.

**4.** Click **Write**.

The variable displays the written value within the dependencies structure.

**To access the dictionary instance containing a variable**

You can access the dictionary instance containing a variable for variables displayed in dependencies structures.

- From the dependencies structure for a variable, select the variable for which to access the dictionary instance, right-click, and then click **Variables**.

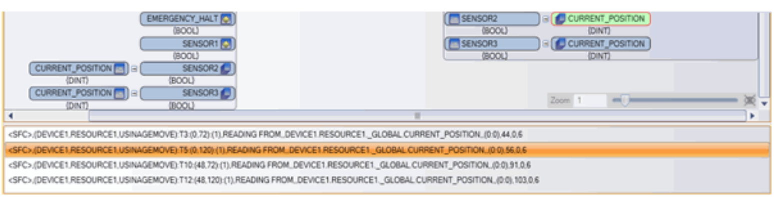
The dictionary instance having the variable is displayed.

**To access the POUs where a variable is used**

For any variable selected in the dependencies structure, you can access all occurrences where the variable is used.

**1.** From the dependencies structure for a variable, select the variable for which to access the POUs where it is used.

**2.** From the list of the variable usage occurrences below the dependencies structure, double-click the required occurrence to open its usage location.




**To set the zoom of a dependencies window**

You can adjust the magnification factor for individual dependencies windows.

- From the dependencies window for a variable, slide the zoom scale to the required magnification factor.

# Properties Window

The Properties window enables viewing and editing the properties of items selected within language containers, ISaVIEW instances, the Solution Explorer, and the Deployment View. You can also use the Properties window to view and edit file, project, and solution properties. You can view the common properties for multiple objects and elements. When selecting multiple objects, the Properties window displays only the properties that are common to all the objects and elements.

In the Properties window, properties are organized into categories displayed alphabetically. You can expand the categories to view the property information including property names and values. For ISavIEW objects, you can also choose to display either basic or extended (all) properties. Note that properties displayed in gray are read-only.

You edit property values using the plain text fields and drop-down combo-boxes provided. Where required, links to custom editors or dialogs are displayed in the property value fields.

The Properties window toolbar containing the following:

| | |
|---|---|
|  | Displays the name of the item or group of items selected. |
|  | Displays the property names and values organized into categories. You click ⊞ to expand categories and ⊟ to collapse categories. |
|  | Displays the properties sorted in alphabetical order |
|  | Displays the basic (subset of extended) properties for a selected ISaVIEW object. You can choose to include individual properties as basic properties. |
|  | Displays the extended (all) properties for a selected ISaVIEW object |
|  | Displays the Properties Pages for the Solution |

**To access the Properties window**

- From the View menu, click **Properties Window**. The **F4** or **Alt+Enter** keyboard shortcuts are also available.

# Collection Editor

The collection editor enables creating and editing individual members of collections. The properties available for editing depend on the collection. The collection editor is made up of a members list and a properties grid. You can perform the following tasks in the collection editor:

- Add members to the list. You add members by selecting a member, then clicking Add. Clicking the first time adds an initial member.
- Remove members from the list. You remove members by selecting a member, then clicking Remove.
- Reorder members in the list. You reorder members by selecting the member, then clicking the up or down arrows.
- Edit the properties of a member. You edit properties by selecting the member, then editing its properties in the grid.

**To access a Collection Editor**

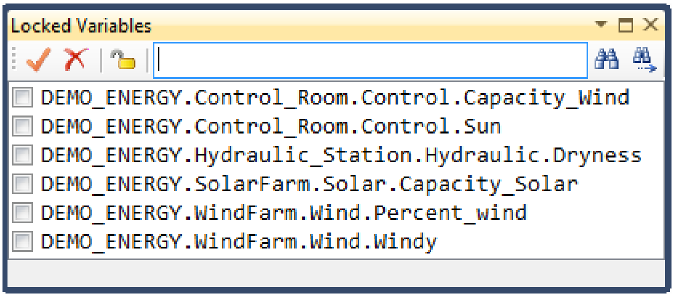
- From the Properties window, in the *Fill Color Phase* property for an ISaVIEW object, click .

The Collection Editor is displayed.

### See Also
Properties Window

# Locked Variables Viewer

The Locked Variables window enables unlocking locked variables while debugging, running online, and simulating an application. This window lists all locked variables and their source throughout an application. When viewing locked variables, the variable identification indicates its source such as the program name, device name, and ending with the variable name.




From the Locked Variables window, you can perform the following tasks:

- Find variables within the list of locked variables based on any part of their source name
- Unlock variables from the list of locked variables

**To access the list of locked variables**

You can only access the Locked Variables window while debugging, running online, or simulating an application. The window lists all locked variables throughout the application.

- From the Debug menu, click **Locked Variables**.

**To find variables from the list of locked variables**

You can perform searches based on any part of the variable identification displayed in the viewer such as the complete or partial variable, resource (if supported by the CAM), device, or project name. For example, to locate the DEMO_ENERGY.SolarFarm.Solar.Consumption variable, the following are some possible search entries: Solar, Farm, Consumption.

- In the search field, enter text contained in the identification of the required variable, then do one of the following:

  - To find the first instance of the variable, click  .

  - To find the next instance of the variable, click  .

**To unlock variables from the list of locked variables**

You can unlock one or more variables from the list of locked variables.

- In the list of locked variables, select the variables to unlock or click  to select all the variables in the list, then click  to unlock them.

# ISaVIEW

You can create graphical interfaces, i.e., ISaVIEW screens, within the workbench. From these screens, you can monitor or run control processes on local computers or remote locations using internet or network connections. You can add ISaVIEW screens to Solutions at the device, resource (if supported by the CAM), and program level.

You create and develop ISaVIEW screens in the Workbench while editing a project or running online (simulation or debugging). Developing ISaVIEW screens consists of inserting graphic objects and defining animation behaviors for execution at run-time.

While running online, you can switch between design mode and animation mode. Design mode enables editing objects contained in screens. Animation mode launches the execution of animation effects defined for objects contained in screens.

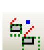
From the ISaVIEW toolbar, you can perform the following operations:

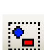
 Design Mode, enables editing objects contained in screens while running online (simulation or debugging)

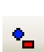
 Animation Mode, launches the execution of animation effects defined for objects contained in screens while running online (simulation or debugging)

 No Preview, enables graphically editing objects contained in a screen without displaying any animation effects. You can modify, add, delete, move, group, or ungroup objects.

 Preview Selection, enables visualizing some animation effects defined for selected objects in a screen

 Animation Preview (Editable), enables visualizing and graphically modifying some animation effects defined for all objects contained in a screen. You cannot modify, add, delete, move, group, or ungroup objects.

 Group Selection, enables grouping selected objects

 Ungroup Selection, enables dissociating a selected group of objects

### See Also

Creating ISaVIEW Screens

Inserting Objects
Defining Animation Effects for Objects

# Creating ISaVIEW Screens

While editing a Workbench project or running online (simulating or debugging), you can create and develop ISaVIEW screens. While simulating or debugging, you need to switch the ISaVIEW screen to design mode.

You can create ISaVIEW screens from blank documents or from a template. Developing screens consists of inserting objects available from the Toolbox. You can define animation effects for objects by modifying their properties. You can also group objects together, then define animation effects for the group.

**To create an ISaVIEW screen from a blank document**

1. From the Solution Explorer, right-click the required device, resource (if supported by the CAM), or program element, point to **Add**, then click **New ISaVIEW**.

   An ISaVIEW screen is added in the Solution Explorer.

2. Open the screen by double-clicking in the Solution Explorer.
3. Proceed to inserting objects and defining animation effects.

**To create an ISaVIEW screen from a template**

ISaVIEW templates have the *.hmi extension.

1. From the Solution Explorer, right-click the required device, resource (if supported by the CAM), or program element, point to **Add**, then click **ISaVIEW from Template**.
2. In the Select the ISaVIEW Template dialog box, browse to locate the required template, then click **Open**.
3. In the Import ISaVIEW Template dialog box, specify a screen name and associate the required variables where required, then click **OK**.

   An ISaVIEW screen is added in the Solution Explorer.

4. Open the screen by double-clicking in the Solution Explorer.
5. Proceed to inserting objects and defining animation effects.

### See Also
ISaVIEW
Exporting ISaVIEW Screens as Templates

## Exporting ISaVIEW Screens as Templates

ISaVIEW templates are screens that you export as templates. These templates are assigned the *.hmi extension. When creating a template, you develop an ISaVIEW screen then export the screen to template. The default names of templates are the same as the screen names. When adding an ISaVIEW screen from a template, you are automatically prompted to choose from the available templates.

ISaVIEW templates are stored in the following location:

%PROGRAMFILES(X86)%\ISaGRAF\6.x\ACP\Templates\ItemTemplates

**To export an ISaVIEW screen as template**

1. Develop an ISaVIEW screen.
2. From the Solution Explorer, right-click the ISaVIEW item, then click **Export as Template**.

## Inserting Objects

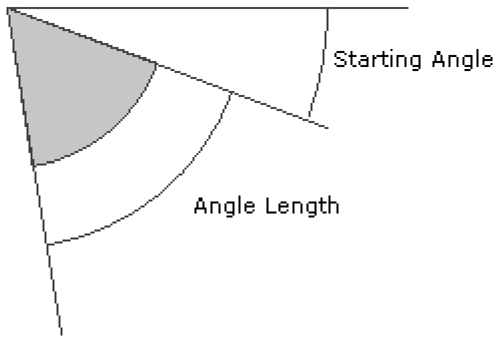
You insert objects into an ISaVIEW screen from the Toolbox. The available objects are the following:

- Arc
- Arrow
- Ellipse
- Rectangle
- Rounded Rectangle
- Triangle
- Image
- Web Container
- Button
- Edit Box
- Gauge
- Slider
- Line
- Bar Meter
- Polygon

You can overlap or superimpose objects and groups of objects. Using the contextual menu options, you can group objects and move objects to the front or back. Note that web containers always remain on top of other objects.

# Arc

An arc is any unbroken part of the circumference of a circle. An arc can represent, for example, a container displaying a changing quantity of liquid as it flows to or from another object. An arc object is made up of a starting angle and an angle length:




You define the properties for the arc object using the Properties Window. For the arc object, you can define properties for Action, Color, Displacement, Size, Text, and Visibility. The arc object also has the following specific properties:

| | |
|---|---|
| **Angle Length** | Length of the arc in degrees |
| **Starting Angle** | Size of the angle prior to the start of the arc |

**To insert an arc**

- From the Toolbox, drag the arc object into the workspace.

### See Also

Defining Animation Effects for Objects

## Arrow

The arrow object is a directional shape having a rectangular shaft and triangular head. You define the properties for the arrow object using the Properties Window. For the arrow object, you can define properties for Action, Color, Displacement, Rotation, Size, Text, and Visibility. The arrow also has frame color and width properties.

| | |
|---|---|
| **Frame Color** | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| **Frame Width** | Width of the frame for the object. Possible values are literal values. |

You define fill color for objects in the Color properties.

**To insert an arrow**

- From the Toolbox, drag the arrow object into the workspace.

### See Also

Defining Animation Effects for Objects

# Ellipse

An ellipse can represent items such as a container displaying a changing quantity of liquid as it flows to or from another object. You define the properties for the ellipse object using the Properties Window. For the ellipse object, you can define properties for Action, Color, Displacement, Rotation, Size, Text, and Visibility. The ellipse also has frame color and width properties.

| | |
|---|---|
| **Frame Color** | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| **Frame Width** | Width of the frame for the object. Possible values are literal values. |

You define fill color for objects in the Color properties.

**To insert an ellipse**

- From the Toolbox, drag the ellipse object into the workspace.

### See Also

Defining Animation Effects for Objects

## Rectangle

A rectangle can represent, for example, pipes indicating a flow from one object to another with a change of color or a container displaying a changing quantity of liquid as it flows to or from another object. You define the properties for the rectangle object using the Properties Window. For the rectangle object, you can define properties for Action, Color, Displacement, Rotation, Size, Text, and Visibility. The rectangle also has frame color and width properties.

| | |
|---|---|
| **Frame Color** | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| **Frame Width** | Width of the frame for the object. Possible values are literal values. |

You define fill color for objects in the Color properties.

**To insert a rectangle**

- From the Toolbox, drag the rectangle object into the workspace.

### See Also

Defining Animation Effects for Objects

## Rounded Rectangle

A rounded rectangle is a rectangular shape having its corners rounded. You define the properties for the rounded rectangle object using the Properties Window. For the rounded rectangle object, you can define properties for Action, Color, Displacement, Size, and Text, and Visibility. The rounded rectangle also has frame color and width properties.

| | |
|---|---|
| **Frame Color** | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| **Frame Width** | Width of the frame for the object. Possible values are literal values. |
| **Corner Radius** | Radius of the corners for a rounded rectangle. Possible values are literal values. |

You define fill color for objects in the Color properties.

**To insert a rounded rectangle**

- From the Toolbox, drag the rounded rectangle object into the workspace.

### See Also

Defining Animation Effects for Objects

## Triangle

A triangle object is a triangular shape. You define the properties for the triangle object using the Properties Window. For the triangle object, you can define properties for Action, Color, Displacement, Rotation, Size, Text, and Visibility. The triangle also has frame color and width properties.

| | |
|---|---|
| **Frame Color** | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| **Frame Width** | Width of the frame for the object. Possible values are literal values. |

You define fill color for objects in the Color properties.

**To insert a triangle**

- From the Toolbox, drag the triangle object into the workspace.

### See Also

Defining Animation Effects for Objects

## Image

The image object can hold file formats such as GIF, JPEG, BMP, PNG, and TIFF. You cannot rotate image objects.

You define the properties for the image object using the Properties Window. For the image object, you can define properties for Action, Displacement, Size, Text, and Visibility. The image object also has the Image Path property.

| | |
|---|---|
| **Image Path** | Path to the image to display |

**To insert an image**

- From the Toolbox, drag the image object into the workspace.

### See Also

Defining Animation Effects for Objects

## Web Container

The web container object has a rectangular shape. You define the properties for the web container object using the Properties Window. For the web container object, you can define properties for Action, Displacement, Size, Text, and Visibility. The web container object also has frame and object-specific properties:

| | |
|---|---|
| **Link Page** | Target URI to display in the object |
| **Frame Color** | Color for the frame of the object. Possible colors are custom, web, and system colors. |

Within a screen, the web container always remains on top of other objects.

**To insert a web container**

- From the Toolbox, drag the web container object into the workspace.

### See Also

Defining Animation Effects for Objects

## Button

The button object displays text and has a rectangular shape. You define the properties for the button object using the Properties Window. For the button object, you can define properties for Action, Color, Displacement, Size, Text, and Visibility.

**To insert a button**

- From the Toolbox, drag the button object into the workspace.

### See Also

Defining Animation Effects for Objects

# Edit Box

The edit box object enables displaying and entering text and has a rectangular shape. You define the properties for the edit box object using the Properties Window. For the edit box object, you can define properties for Action, Color, Displacement, Size, Text, and Visibility. You can also choose to display a border outlining the edit box:

**Border** Indication of whether the object has a border. Possible values are True or False.

**To insert an edit box**

- From the Toolbox, drag the edit box object into the workspace.

## See Also

Defining Animation Effects for Objects

## Gauge

The gauge object is a circular dial having a needle and range of values representing a traditional meter or dial. The gauge's needle moves around the dial indicating the changing value.

You define the properties for the gauge object using the Properties Window. For the gauge object, you can define properties for Action, Color, Displacement, Size, Text, and Visibility. The gauge object also has frame and object-specific properties:

| | |
|---|---|
| **Frame Color** | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| **Frame Width** | Width of the frame for the object. Possible values are literal values. |
| **Background Shape** | Shape of the measuring object. Possible shapes are rectangle, ellipse, hexagon, and octagon. |
| **Indicator Value Variable** | Variable controlling the indicator of the measuring object. Possible variable data types are SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, and LREAL. Clicking  enables selecting a variable. |
| **Indicator Color** | Color for the interior of the indicator. Possible colors are custom, web, and system colors. |
| **Indicator Constant Length** | Indication of whether the indicator maintains the same length when traveling along the scale of the measuring object. Possible values are True or False. |
| **Indicator Frame Color** | Color for the outline of the indicator. Possible colors are custom, web, and system colors. |
| **Indicator Frame Width** | Width of the outline for the indicator. Possible values are literal values. |
| **Indicator Thickness** | Width of the indicator. Possible values are literal values. |
| **Indicator Value** | Initial value of the measuring object. Possible values are literal values. |
| **Margin Bottom Right** | Margin from the gauge dial to the bottom and right sides of the object perimeter. Possible values are literal values. |

| | |
|---|---|
| **Margin Top Left** | Margin from the gauge dial to the top and left sides of the object perimeter. Possible values are literal values. |
| **Maximum** | Maximum value of the scale on the measuring object. Possible values are literal values. |
| **Minimum** | Minimum value of the scale on the measuring object. Possible values are literal values. |
| **Scale Label Distance** | Distance between the scale on the measuring object and the displayed range values, in pixels. Possible values are literal values. |
| **Scale Label Frequency** | Frequency of labeling of major divisions on the scale of the measuring object. For example, a value of two (2) results in labeling every second major division. Possible values are literal values. |
| **Scale Label Style** | Location of the displayed labels in reference to the circular scale. Possible values are Left, Right, AlternateStartLeft, and AlternateStartRight. Setting labels on the left places these on the outside of the scale while labels on the right places these on the inside of the scale. Setting alternate starts places the lowest range label respectively then every other label on alternating sides of the scale. |
| **Scale Label Text Bold** | Indication of whether the bold style is applied to the label text. Possible values are True or False. |
| **Scale Label Text Color** | Color of the label text. Possible colors are custom, web, and system colors. |
| **Scale Label Text Size** | Size of the label text. Possible values are literal values. |
| **Scale Frame Color** | Color of the scale on the measuring object. Possible colors are custom, web, and system colors. |
| **Scale Frame Width** | Width of the scale on the measuring object. Possible values are literal values. |
| **Scale Start Angle** | Angle at which the circular scale starts in reference to the x-axis. For example, a start angle of 0° places the beginning of the scale on the positive x-axis. Possible values are 0 to 360. |
| **Scale Sweep Angle** | Span of the circular scale. For example, a sweep angle of 180° indicates a semicircular scale. Possible values are 0 to 360. |

| | |
|---|---|
| **Scale Tick Major Frequency** | Frequency of major ticks in reference to minor ticks on the scale. For example, on a scale ranging from 1 to 100 having a Tick Unit value of 5, a major tick frequency setting of 5 sets a major division at every 5th minor division, i.e., at each increment of 25. Possible values are literal values. |
| **Scale Tick Major Width** | Width of the major tick marks dividing the scale. Possible values are literal values. |
| **Scale Tick Width** | Width of the minor tick marks dividing the scale. Possible values are literal values. |
| **Tick Color** | Color of the ticks dividing the scale. Possible colors are custom, web, and system colors. |
| **Tick Unit** | Value associated to individual tick divisions on the measuring scale. Possible values are literal values. |

**To insert a gauge**

- From the Toolbox, drag the gauge object into the workspace.

## See Also

Defining Animation Effects for Objects

## Slider

The slider object reads the position of the indicator within its perimeter then sends a value associated to the position to mapped variables. You can define the accuracy of position readings by increasing or decreasing the number of horizontal and vertical divisions within the slider.

You define the properties for the slider object using the Properties Window. For the slider object, you can define properties for Action, Color, Displacement, Size, Text, and Visibility. The slider also has the following object-specific properties:

| | |
|---|---|
| **Frame Color** | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| **Frame Width** | Width of the frame for the object. Possible values are literal values. |
| **Indicator Value Variable** | Variable controlling the indicator of the measuring object. Possible variable data types are SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, and LREAL. Clicking  enables selecting a variable. |
| **Indicator Color** | Color for the interior of the indicator. Possible colors are custom, web, and system colors. |
| **Indicator Dimensions** | The length and width of the indicator, in pixels. Possible values are literal values. |
| **Indicator Frame Color** | Color for the outline of the indicator. Possible colors are custom, web, and system colors. |
| **Indicator Style** | Shape of the indicator. Possible shapes are bar and triangles. |
| **Indicator Value** | Initial value of the measuring object. Possible values are literal values. |
| **Maximum** | Maximum value of the scale on the measuring object. Possible values are literal values. |
| **Minimum** | Minimum value of the scale on the measuring object. Possible values are literal values. |
| **Orientation** | Indication of whether the orientation of the measuring object is horizontal or vertical |

| | |
|---|---|
| **Scale Label Distance** | Distance between the scale on the measuring object and the displayed range values, in pixels. Possible values are literal values. |
| **Scale Label Frequency** | Frequency of labeling of major divisions on the scale of the measuring object. For example, a value of two (2) results in labeling every second major division. Possible values are literal values. |
| **Scale Label Text Bold** | Indication of whether the bold style is applied to the label text. Possible values are True or False. |
| **Scale Label Text Color** | Color of the label text. Possible colors are custom, web, and system colors. |
| **Scale Label Text Size** | Size of the label text. Possible values are literal values. |
| **Scale Frame Color** | Color of the scale on the measuring object. Possible colors are custom, web, and system colors. |
| **Scale Frame Width** | Width of the scale on the measuring object. Possible values are literal values. |
| **Scale Tick Major Frequency** | Frequency of major ticks in reference to minor ticks on the scale. For example, on a scale ranging from 1 to 100 having a Tick Unit value of 5, a major tick frequency setting of 5 sets a major division at every 5th minor division, i.e., at each increment of 25. Possible values are literal values. |
| **Scale Tick Major Width** | Width of the major ticks dividing the scale. Possible values are literal values. |
| **Scale Tick Width** | Width of the minor ticks dividing the scale. Possible values are literal values. |
| **Tick Color** | Color of the ticks dividing the scale. Possible colors are custom, web, and system colors. |
| **Tick Unit** | Value associated to individual tick divisions on the measuring scale. Possible values are literal values. |

**To insert a slider**

- From the Toolbox, drag the slider object into the workspace.

**See Also**

Defining Animation Effects for Objects

## Line

The line object is a unbroken linear shape. You define the properties for the line object using the Properties Window. For the line object, you can define properties for Action, Displacement, Rotation, Size, and Visibility. The line also has color and width properties.

| | |
|---|---|
| **Line Color** | Color for the line object. Possible colors are custom, web, and system colors. |
| **Line Width** | Width of the line object. Possible values are literal values. |

**To insert a line**

- From the Toolbox, drag the line object into the workspace.

**See Also**

Defining Animation Effects for Objects

## Bar Meter

The bar meter object reads the position of the indicator within its perimeter then sends a value associated to the position to mapped variables. You can define the accuracy of position readings by increasing or decreasing the number of divisions within the bar meter.

You define the properties for the bar meter object using the Properties Window. For the bar meter object, you can define properties for Action, Color, Displacement, Size, Text and Visibility. You can also define properties for the frame color and width. The bar meter object has the following specific properties:

**Frame Color** Color for the frame of the object. Possible colors are custom, web, and system colors.

**Frame Width** Width of the frame for the object. Possible values are literal values.

**Indicator Value Variable** Variable controlling the indicator of the measuring object. Possible variable data types are SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, and LREAL. Clicking  enables selecting a variable.

**Indicator Color** Color for the interior of the indicator. Possible colors are custom, web, and system colors.

**Indicator Frame Color** Color for the outline of the indicator. Possible colors are custom, web, and system colors.

**Indicator Value** Initial value of the measuring object. Possible values are literal values.

**Maximum** Maximum value of the scale on the measuring object. Possible values are literal values.

**Minimum** Minimum value of the scale on the measuring object. Possible values are literal values.

**Orientation** Indication of whether the orientation of the measuring object is horizontal or vertical

**Scale Label Distance** Distance between the scale on the measuring object and the displayed range values. Possible values are literal values.

| | |
|---|---|
| **Scale Label Frequency** | Frequency of labeling of major divisions on the scale of the measuring object, in pixels. For example, a value of two (2) results in labeling every second major division. Possible values are literal values. |
| **Scale Label Text Bold** | Indication of whether the bold style is applied to the label text. Possible values are True or False. |
| **Scale Label Text Color** | Color of the label text. Possible colors are custom, web, and system colors. |
| **Scale Label Text Size** | Size of the label text. Possible values are literal values. |
| **Scale Frame Color** | Color of the scale on the measuring object. Possible colors are custom, web, and system colors. |
| **Scale Frame Width** | Width of the scale on the measuring object. Possible values are literal values. |
| **Scale Tick Major Frequency** | Frequency of major ticks in reference to minor ticks on the scale. For example, on a scale ranging from 1 to 100 having a Tick Unit value of 5, a major tick frequency setting of 5 sets a major division at every 5th minor division, i.e., at each increment of 25. Possible values are literal values. |
| **Scale Tick Major Width** | Width of the major ticks dividing the scale. Possible values are literal values. |
| **Scale Tick Width** | Width of the minor ticks dividing the scale. Possible values are literal values. |
| **Tick Color** | Color of the ticks dividing the scale. Possible colors are custom, web, and system colors. |
| **Tick Unit** | Value associated to individual tick divisions on the measuring scale. Possible values are literal values. |

**To insert a bar meter**

- From the Toolbox, drag the bar meter object into the workspace.

## See Also
Defining Animation Effects for Objects

## Polygon

A polygon object is made up of three or more connected straight lines forming a closed figure. Each line is called a segment. You define the properties for the polygon object using the Properties Window. For the polygon object, you can define properties for Action, Color, Displacement, Size, Text, and Visibility. The polygon also has frame color and width properties.

| | |
|---|---|
| **Frame Color** | Color for the frame of the object. Possible colors are custom, web, and system colors. |
| **Frame Width** | Width of the frame for the object. Possible values are literal values. |

You define fill color for objects in the Color properties.

**To insert a polygon**

When creating polygon objects, you need to establish the end of each segment making up the shape, then establish the end of the shape when all segments are completed. You establish the end of a segment.

You can use keyboard commands when working with polygon objects: pressing Ctrl+Z undoes up to the first segment and pressing Escape deletes a polygon object in progress.

**1.** From the Toolbox, drag the polygon object into the workspace.

**2.** Click and drag to define each segment making up the shape, then press Enter to complete the shape.

**3.** To cancel the shape, press Escape.

### See Also

Defining Animation Effects for Objects

## Editing Objects

You can perform many editing tasks on objects:

- Selecting Objects
- Editing the Properties of Objects
- Cutting, Copying, and Pasting Objects
- Deleting Objects
- Moving Objects
- Resizing Objects
- Grouping Objects
- Aligning Objects
- Moving Objects to the Front and Back

While running online (simulation and debugging), you can edit objects by switching to design mode.

## Selecting Objects

Selecting objects is required as a first step for all editing functions. You can choose to select one or more objects at the same time. When objects are selected, you can move these by dragging with the mouse.

**To select a single object**

- In the workspace, click the desired object

**To select multiple objects**

You can select multiple objects either by dragging the cursor to enclose them or by selecting individual objects. When dragging, an invisible rectangle encloses the area.

- Position the cursor to the left and above the desired objects, then drag to enclose them
- Hold down the Ctrl key while clicking the desired objects one after the other. Clicking a selected object deselects the object while all others remain selected.
- Hold down the Shift key while clicking the desired objects one after the other

**Deselecting objects**

- Click on an empty space in the workspace
- Press the ESCAPE key

## Editing the Properties of Objects

You can change the properties of objects in the Properties window.

**To edit the properties of an object**

**1.** Select the object.

**2.** In the Properties window, enter the required information for the individual properties.

## Cutting, Copying, and Pasting Objects

You can cut, copy, and paste objects in screens using the commands from the contextual menu or from the Edit menu. To access the contextual menu, right-click a selected object.

**To cut an object**

**1.** Select the desired object.

**2.** Do one of the following:

- Right-click, then click **Cut**.
- From the Edit menu, click **Cut**.

The object is removed from the workspace and a copy is placed on the clipboard.

**To copy an object**

**1.** Select the desired object.

**2.** Do one of the following:

- Right-click, then click **Copy**.
- From the Edit menu, click **Copy**.
- Press the Ctrl key and drag the object.

A copy of the object is placed on the clipboard.

**To paste an object**

You can insert the contents of the clipboard into the workspace.

- In the workspace, click where you want to insert the object, then do one of the following:
  - Right-click, then click **Paste**.
  - From the Edit menu, click **Paste**.

The content of the clipboard is inserted in the workspace.

## Deleting Objects

Once you select an object, you can choose to delete it using the commands from the contextual menu or from the Edit menu. To access the contextual menu, right-click the selected object.

**To delete objects**

**1.** Select the desired object.

**2.** Do one of the following:

- Right-click, then click **Delete**.
- From the Edit menu, click **Delete**.

The object is removed from the workspace.

## Moving Objects

You can move objects within the screen.

**To move objects**

- Select one or more objects then drag to their new position

## Resizing Objects

You can resize objects in screens.

**To resize an object**

1. Select the desired object.
2. Click a handle (a small square on the outer edge of the selected object) then move it in the appropriate direction.

## Grouping Objects

You can group individual objects in a screen to form a unique object. You can also group individual groups of objects. When objects are grouped, you cannot resize, move, delete, or copy the individual objects contained in the group. You can change the properties of individual objects belonging to a group as well as those properties of grouped objects.

You can apply action, size, and visibility animation effects to grouped objects other than those effects attached to the grouped items.

Once you select objects, you can choose to group and ungroup these using commands from the contextual menu or icons from the toolbar.

**To group objects**

**1.** Select the required objects.

**2.** Do one of the following:

- Right-click, then click **Group Items**.
- From the ISaVIEW toolbar, click .

**To ungroup objects**

**1.** Select the grouped object.

**2.** Do one of the following:

- Right-click, then click **Ungroup Items**.
- From the ISaVIEW toolbar, click .

## Aligning Objects

You can align objects relative to their left side, right side, top edge, or bottom edge. Elements are aligned relative to the first element you select.

Once you select objects, you can choose to align them using the commands from the Layout menu available from the contextual menu or using the arrow keys.

**To align and position objects**

1. Select the objects to align starting with the element to use as reference for the alignment.
2. Right-click the elements, point to Layout, then click the required alignment command:
   - Align Left
   - Align Center
   - Align Right
   - Align Top
   - Align Middle
   - Align Bottom

The objects are aligned in the selected direction in reference to the first selected item.

## Moving Objects to the Front and Back

You can move objects to the front or to the back of each other. Once you select objects, you can choose to move these using the commands from the contextual menu.

**To bring an object to the front**

1. Select the object.
2. Right-click the element, then click **Bring to Front**.

**To send an object to the back**

1. Select the object.
2. Right-click the element, then click **Send to Back**.

## Defining Animation Effects for Objects

You can define animation effects for objects or groups of objects defined in ISaVIEW screens. The Workbench supports the following animation effects:

- Action
- Color
- Displacement
- Rotation
- Size
- Text
- Visibility

You define animation effects by setting their property values in the Properties window. You can also graphically modify the rotation, displacement, and size properties by switching to the animation preview mode.

When setting the properties in the Properties window, all global and local variables are available for use. The Collection Editor is available when defining the color property Fill Color Phase and the text property Text Color Phase. You use the Collection Editor to create and edit the members of a collection and to define the colors (PhaseColors) and numerical values (PhaseMaximum and PhaseMinimum) for each member. When online, the object or object text displays the color that corresponds to its current value as defined in the collection. For example, for the color black, assigning a value of 10 to PhaseMaximum and a value of 0 to PhaseMinimum enables the object or object text to display as black when its value is between 0 and 10.

**See Also**

ISaVIEW
Previewing ISaVIEW Screens

## Action

Any ISaVIEW object or group can act as a push button. The styles and variables of the action properties enable you to define a push button-like behavior for the object. You define the action properties using the Properties window. The following properties are available for defining the action of an object:

| Action Property | Description |
|---|---|
| **Action Event** | Operation to perform upon occurrence of Action Type event. Possible values are None, GoToHTML, GoToPage, IncrementValue, AutoIncrementValue, and ReverseValue. |
| **Action Link** | Destination address or path for GoToHTML or GoToPage Action Event operations. Possible values are ftp://, http://www, and \\. |
| **Action Type** | Mouse event triggering the Action Event operation. Possible values are None, MouseClick, MouseDoubleClick, and MouseAll. |
| **Action Variable** | Variable controlling the Action Event for IncrementValue, AutoIncrementValue, and ReverseValue operations. Depending on the Action Event Type, the expected variable types are the following:<br>- IncrementValue: any integer and any real<br>- AutoIncrementValue: any integer and any real<br>- ReverseValue: Boolean, any integer and any real<br><br>Clicking  enables selecting a variable. |
| **Increment Time** | Interval between increments, in seconds, of the Action Variable variable where the Action Event is AutoIncrementValue |
| **Increment Value** | Rate of increase of the Action Variable variable for each Action Type mouse event where the Action Event is either IncrementValue or AutoIncrementValue |

| ActionEvent Operations | Description |
|---|---|
| **None** | Disables Action Event |
| **GoToHTML** | Jumps to the HTML page defined in Action Link |
| **GoToPage** | Jumps to the ISaVIEW page defined in Action Link |
| **IncrementValue** | Increments once the value of the Action Variable variable by the value of Increment Value |
| **AutoIncrementValue** | Increments continuously the Action Variable variable by the Increment Value value using the Increment Time time lapse |
| **ReverseValue** | Reverses the value of the Action Variable variable |

| Mouse Event | Description |
|---|---|
| **None** | Disables Action Type |
| **MouseClick** | Sets a single mouse click to execute Action Event |
| **MouseDoubleClick** | Sets a double mouse click to execute Action Event |
| **MouseAll** | Sets any type of mouse click to execute Action Event |

**To define the action properties of an object**

You define action properties for an object from the Properties window while the ISaVIEW screen is in design mode.

1. Set the ISaVIEW screen to design mode by clicking , in the ISaVIEW toolbar.
2. In the ISaVIEW screen, select the required object or group of objects.
3. In the Properties window, define the required action properties.

## See Also
ISaVIEW
Defining Animation Effects for Objects

## Color

You can define color properties for the following objects: arcs, arrows, ellipses, rectangles, rounded rectangles, triangles, buttons, edit boxes, gauges, sliders, and polygons. You define the color properties using the Properties window. The following properties are available for defining the color of an object:

| | |
|---|---|
| **Color Variable** | Variable defining the phase value during animation mode. Possible variable data types are DINT and DWORD. Clicking enables selecting a variable. |
| **Initial Color** | *(Read only)* Initial color of the object, while in design mode |
| **Fill Color** | Actual color of the object. Equal to InitialColor while in design mode. Possible colors are custom, web, and system colors. |
| **Fill Color Phase** | List of colors to apply during phases while in animation mode. Clicking accesses the phase collection editor. |
| **Fill Foreground Color** | Contrast color used for Fill Style. Possible colors are custom, web, and system colors. Available for all objects except edit boxes. |
| **Fill Style** | Style applied to the coloring of an object such as a gradient, texture, or hatch line. Available for all objects except edit boxes. Fill Foreground Color provides the contrast color used in the style. Possible styles are available from a drop-down combo-box. |

**To define the color properties of an object**

You define color properties for an object from the Properties window while the ISaVIEW screen is in design mode.

1. Set the ISaVIEW screen to design mode by clicking , in the ISaVIEW toolbar.
2. Select the required object or group of objects.
3. In the Properties window, define the required color properties.

### See Also

ISaVIEW

## Displacement

You can define displacement properties for all ISaVIEW objects. Before displacement occurs, the starting position is defined by the coordinates of the upper left corner of the object. The displacement properties enable you to define the linear movement of the object when in animation mode. You define displacement properties in the Properties window. Also, you can define the AnimationPosition property within the workspace. The following properties are available for defining the displacement of an object:

| | |
|---|---|
| **Animation Position** | Destination coordinates after displacement during animation mode in reference to the top left corner of the object bounding box, in pixels (design mode displays InitialPosition coordinates) |
| **Displacement Variable** | Variable controlling the object displacement. Possible variable data types are DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT, and STACKINT. Clicking  enables selecting a variable. |
| **Initial Position** | *(Read only)* Coordinates of the object prior to displacement |
| **Location** | Actual coordinates of the object. Equal to Initial Position while in design mode. |
| **Maximum Displacement** | Maximum amount of displacement during animation mode. The default value is 100. |
| **Minimum Displacement** | Minimum amount of displacement during animation mode. The default value is 0. |

In animation mode, the final position of the object is defined as the following:

```
Initial Position + (Animation Position - Initial Position) *
[(Displacement Variable - Minimum Displacement) / (Maximum Displacement
- Minimum Displacement)]
```

**To define the displacement properties of an object**

You define displacement properties for an object from the Properties window while the ISaVIEW screen is in design mode.

**1.** Set the ISaVIEW screen to design mode by clicking , in the ISaVIEW toolbar.

2. Select the required object or group of objects.

3. In the Properties window, define the required displacement properties.

**See Also**
ISaVIEW
Defining Animation Effects for Objects

## Rotation

You can define the rotation properties for the following ISaVIEW objects: arrows, ellipses, rectangles, triangles, and lines. The rotation properties enable you to define the rotation of the object when in animation mode. You define rotation properties in the Properties window. Also, you can define the CenterOfRotation property within the workspace. The following properties define the rotation of an object:

| | |
|---|---|
| **Center of Rotation** | Coordinates of the center of rotation for the object in reference to the top left corner of the object bounding box |
| **Rotation Variable** | Variable controlling the object rotation. Possible variable data types are DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT, and STACKINT. Clicking  enables selecting a variable. |
| **Maximum Rotation** | Maximum range of rotation of the object, in degrees. Possible values are positive or negative; The default value is 360 degrees. |
| **Minimum Rotation** | Minimum range of rotation of the object, in degrees. Possible values are positive or negative; The default value is 0 degrees. |
| **Static Angle** | Initial angle in reference to the right side of the base of the object. Possible values are 0 to 360. |

The final rotation of an object is defined as the following:

```
{[(Rotation Variable - Minimum Rotation) * 360]/(Maximum Rotation -
Minimum Rotation)}%360
```

**To define the rotation properties of an object**

You define displacement properties for an object from the Properties window while the ISaVIEW screen is in design mode.

1. Set the ISaVIEW screen to design mode by clicking , in the ISaVIEW toolbar.
2. Select the required object or group of objects.
3. In the Properties window, define the required rotation properties.

**See Also**

ISaVIEW

Defining Animation Effects for Objects

## Size

You can modify the size of all ISaVIEW objects. You define size properties in the Properties window. Also, you can define the AnimationSize property within the workspace. The following properties are available for defining the size of an object:

| | |
|---|---|
| **Animation Size** | Maximum enlargement of the object in percentage (%). This value must be at least 100%. |
| **Size Variable** | Variable controlling the resizing of the object. Possible variable data types are DINT, INT, LINT, SINT, UDINT, UINT, ULINT, USINT, and STACKINT. Clicking ... enables selecting a variable. |
| **Initial Size** | *(Read only)* The width and height of the object before resizing occurs in animation mode |
| **Maximum Size** | Value used by Size Variable defining the maximum range of enlargement for the object. Possible values are positive or negative and must be greater than Minimum Size; The default value is 100. |
| **Minimum Size** | Value used by Size Variable defining the minimum range of enlargement for the object. Possible values are positive or negative and must be less than Maximum Size; The default value is 0. |
| **Size** | Actual size of the object whether in design or animation mode |

**To define the size properties of an object**

You define size properties for an object from the Properties window while the ISaVIEW screen is in design mode.

1. Set the ISaVIEW screen to design mode by clicking , in the ISaVIEW toolbar.
2. Select the required object or group of objects.
3. From the Properties window, define the required size properties.

**See Also**
ISaVIEW
Defining Animation Effects for Objects

# Text

You can define text properties for the following objects: arcs, arrows, ellipses, rectangles, rounded rectangles, triangles, buttons, edit boxes, web containers, gauges, sliders, bar meters, and polygons. However, for web containers, only the **Text** text property is available; All other text properties are unavailable for this object. You define text properties in the Properties window. The following properties define the appearance of the text associated with objects:

| | |
|---|---|
| **Text Variable** | Variable controlling the text displayed on the object. All variable data types are possible. Clicking ... enables selecting a variable. Available for all objects except web containers. |
| **Text Color Variable** | Variable controlling the text color. Possible variable data types are DINT and DWORD. Clicking ... enables selecting a variable. Available for all objects except web containers. |
| **Initial Text** | *(Read only)* The text prior to animation mode. Equal to Text while in design mode. Available for all objects except web containers. |
| **Initial Text Color** | *(Read only)* The text color prior to animation mode. Equal to Text Color while in design mode. Available for all objects except web containers. |
| **Text** | Actual text displayed on the object whether in design or animation mode |
| **Text Color** | Actual text color whether in design or animation mode. Possible colors are custom, web, and system colors. Available for all objects except web containers. |
| **Text Color Phase** | List of colors to apply to displayed text during phases while in animation mode. Clicking ... accesses the phase collection editor. Available for all objects except web containers. |
| **Text Size** | Size of the text displayed on the object. Possible values are literal values. Available for all objects except web containers. |

**To define the text properties of an object**

You define text properties for an object from the Properties window while the ISaVIEW screen is in design mode.

1. Set the ISaVIEW screen to design mode by clicking , in the ISaVIEW toolbar.
2. Select the required object or group of objects.
3. From the Properties window, define the required text properties.

## See Also

ISaVIEW
Defining Animation Effects for Objects

## Visibility

You can define the visibility property for individual ISaVIEW objects. You define the visibility property in the Properties window. The following property defines the visibility of an object:

| | |
|---|---|
| **Visibility Variable** | Variable controlling the visibility of the object. Possible variable data type is BOOL. Clicking enables selecting a variable. |

**To define the visibility property of an object**

You define the visibility property for an object from the Properties window while the ISaVIEW screen is in design mode.

**1.** Set the ISaVIEW screen to design mode by clicking , in the ISaVIEW toolbar.

**2.** Select the required object or group of objects.

**3.** From the Properties window, define the visibility property.

**See Also**
ISaVIEW
Defining Animation Effects for Objects

# Previewing ISaVIEW Screens

When visualizing ISaVIEW screens, you can choose to display different graphic views of objects and their properties:

- No preview, displaying the objects defined in a screen where selecting an object exposes bounding box and dimension lines for the object. You can modify, add, delete, move, group, or ungroup objects.
- Preview selections, displaying the objects defined in a screen where selecting an object exposes the bounding box and dimension lines as well as the rotation, displacement, and size animation effects for the object. You can modify, add, delete, move, group, or ungroup objects but only visualize animation effects.
- Previewing animation effects (editable), displaying the objects defined in a screen while exposing the rotation, displacement, and size animation effects for the object. You can modify animation effects but only visualize objects.

Previewing screens is available while debugging.

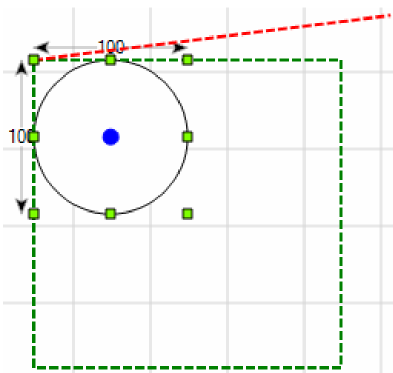
**To switch to no preview mode**

- From the ISaVIEW Toolbar, click .

## Previewing Selections

You can preview selections where you can modify objects and visualize the rotation, displacement, and size animation effects defined for selected individual and grouped objects. While previewing selections, you can modify objects and their properties; you cannot modify any animation effects. However, since the size animation effect is defined as a percentage, the boundaries outlining this effect change as you resize an object.

You visualize the animation effects defined for selected individual and grouped objects from the colored indicators as follows:




- The displacement for the Animation Position property where the broken red line indicates the end position and path of travel for the object.
- The rotation for the Center of Rotation property where the blue circle indicates the center of rotation for the object.
- The size for the Animation Size property where the green broken outline indicates the final size of the object.

Previewing screens is available while debugging.

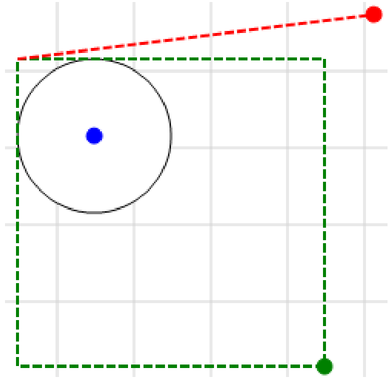
**To switch to preview selection mode**

**1.** From the ISaVIEW Toolbar, click .

**2.** In the screen workspace, select the required objects.

## Previewing Animation Effects (Editable)

You can graphically modify the rotation, displacement, and size properties for individual and grouped objects while in animation preview mode. While in animation preview mode, you cannot add, delete, move, group, or ungroup objects. You graphically modify animation effects properties by repositioning the displayed indicators as follows:



- The displacement indicator for the Animation Position property where the red dot and broken line indicate the end position and path of travel for the object.
- The rotation indicator for the Center of Rotation property where the blue dot indicates the center of rotation for the object.
- The size indicator for the Animation Size property green dot and broken lines indicate the final size of the object.

Previewing screens is available while debugging.

You access the animation preview mode from the ISaVIEW toolbar.

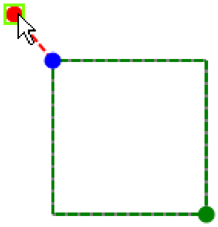
**To switch to animation preview mode**

The following procedure details the steps required to modify the properties of the Animation Position displacement property, the Center of Rotation rotation property, and the Animation Size size property. In the Properties window, you can view changes to the property values as you reposition the colored indicators.

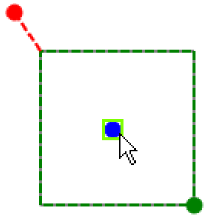
1. From the ISaVIEW Toolbar, click .

2. In the screen workspace, reposition the indicators as follows:

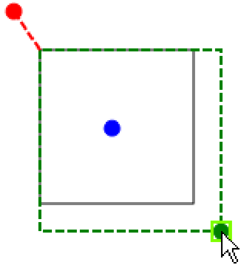
   - To modify the Animation Position property, drag the displacement indicator to the desired position.



   - To modify the Center of Rotation property, drag the rotation indicator to the desired position.



   - To modify the Animation Size property, drag the size indicator to the desired position.



In the Properties window, new values for the Animation Position, Center of Rotation, and Animation Size properties are displayed.

**See Also**

ISaVIEW

Defining Animation Effects for Objects

# Toolbox

You can expand the multiple segments or tabs of the Toolbox. You can also scroll though the entire tree within the Toolbox. To expand Toolbox tabs, click the blank right-pointing arrow next to the tab name. To collapse expanded Toolbox tabs, click the darkened down-pointing arrow next to the tab name.

The Toolbox displays icons for elements that you can add to programs. When shifting focus to a different program, the current selection in the Toolbox shifts to the tab for the corresponding programming language. You can manipulate the Toolbox in the following ways:

- Display the Toolbox
- Conceal the Toolbox
- Close the Toolbox automatically
- Dock the Toolbox
- Move the Toolbox
- Display using tabs
- Restore default Toolbox settings

You can customize the Toolbox by rearranging elements within a tab or adding custom tabs and elements. You can manipulate Toolbox tabs in the following ways:

- Expand tabs
- Collapse tabs
- Move tabs
- Rename tabs
- Add custom tabs
- Remove custom tabs
- Display all tabs
- Restore default tab settings

You can insert elements in language containers displayed in the integrated development environment (IDE). This action adds the fundamental code to create an instance of the Toolbox element in the active program file. You can manipulate Toolbox elements in the following ways:

- Rename elements
- Sort elements
- Conceal element names
- Rearrange elements
- Move elements between Toolbox tabs
- Remove elements
- Restore default elements settings

### To display the Toolbox

- From the View menu, click **Toolbox** (or press **Ctrl+Alt+X**).

### To hide the Toolbox

- From the Window menu, click **Hide**.

### To close the Toolbox automatically

The Toolbox must be docked to enable auto hide.

- From the Window menu, click **Auto Hide**.

### To dock the Toolbox

- From the Window menu, click **Dock**.

### To move the Toolbox to a different location

1. From the Window menu, click **Float**.
2. Drag the Toolbox to the desired location.

### To display the Toolbox as a tabbed document

1. From the Window menu, click **Dock as Tabbed Document**.
2. To restore the Toolbox to a docked window, from the Window menu, click **Dock**.

### To restore all default tabs and elements to the Toolbox

- Right-click the Toolbox, and then click **Reset Toolbox**.

### To expand a Toolbox tab

- Click the blank right-pointing arrow next to the name of the collapsed Toolbox tab.

**To collapse a Toolbox tab**

- Click the darkened down-pointing arrow next to the name of the expanded Toolbox tab.

**To move a Toolbox tab**

You can move Toolbox tabs within the Toolbox by performing one of the following:

- Right-click the name of the tab, and then click **Move Down** or **Move Up**.
- Drag the tab to the required position in the Toolbox, and release the mouse.

**To rename a Toolbox tab**

1. From the Toolbox, right-click the required tab, and then click **Rename Tab**.
2. In the space provided, type a name for the tab, then press ENTER.

**To add a custom Toolbox tab**

When adding tabs, these are displayed at the bottom of the Toolbox. You can reposition and add elements to tabs.

1. From the Toolbox, right-click any tab, and then click **Add Tab**.
2. On the blank tab, in the space provided, type a name for the tab, then press ENTER.

**To remove a custom Toolbox tab**

When removing custom tabs, move the elements to retain to other tabs before deleting the custom tabs.

1. From the Toolbox, right-click the tab to remove, then click **Delete Tab**.

   When elements remain on the tab, a message box informs you that those elements will be deleted.

2. To proceed with the deletion of the selected tab, click **OK**.

**To display all available Toolbox tabs**

- Right-click the Toolbox, and then click **Show All**.

**To insert an element in the workspace**

- From the Toolbox, drag the required element into the workspace.

The element is displayed in the workspace.

**To rename an element**

1. In the Toolbox, right-click the required element, then click **Rename Item**.
2. In the space provided, type a name for the element, then press ENTER.

**To sort the elements alphabetically**

- In the Toolbox, right-click the required tab, then click **Sort Items Alphabetically**.

**To hide element names**

- In the Toolbox, right-click the required tab, then click **List View**.

**To rearrange elements**

You can reposition elements displayed on Toolbox tabs.

- In the Toolbox, select the required element and perform one of the following:
  - Right-click the element, and then click **Move Down** or **Move Up**.
  - Drag the element to the required position.

**To move an element between tabs**

- In the Toolbox, select the required element and perform one of the following:

- Drag the required element onto another tab.
- Right-click the element and click **Cut** or **Copy**, then right-click the required tab and click **Paste**.

**To remove an element**

Note that certain elements cannot be removed, such as the Pointer element.

- In the Toolbox, right-click the required element, and then click **Delete**.

# Variable Selector

The Variable Selector displays the variables defined for an open program. From the Variable Selector, you can create, edit, and delete variables at the global level and local level.

When working in the Variable Selector, you can create variables or limit searches by entering data and choosing from the available options in the field.

- Name, enables defining variable names and literal values. You can filter the variables displayed by typing alpha-numeric characters in the field.
- Type, displays a list of the data types available for a project. You can view the variables having a specific data type by selecting individual data types in the list.
- Global Scope, displays a list of the resources or devices (depending on the CAM) in the project. You can view the variables defined for each by selecting individual items from the list.
- Local Scope, displays a list of the programs available for the item specified in the global scope. You can view the variables defined for each program by selecting individual programs from the list.

You can also navigate the different variables and defined words through the multiple tabs:

- Global Variables, displays the variables defined for the item selected in Global Scope
- Local Variables, displays the variables defined for the program selected in Local Scope
- System Variables, displays the system variables
- Directly Represented Variables, displays the directly represented variables defined for the solution
- Defined Words, displays the defined words specified for the solution

When working in the Variable Selector, you can navigate using keyboard and mouse controls.

| | |
|---|---|
| Arrow keys | Enable moving up, down, left, and right among the cells of the list of variables. Also enables moving left and right among the tabs. |
| Tab key | Enables moving from left to right between the fields, tab, and list of variables. Within the list, enables moving left to right between cells of a row. After exiting the list of variables, enables moving between the command buttons and back to the fields. |
| Esc key | Enables moving from the list of variables to the command buttons. |
| Ctrl + PLUS SIGN on the numeric keypad (+) | Expands the fields of complex data types |
| Ctrl + MINUS SIGN on the numeric keypad (-) | Collapses the fields of complex data types |
| Enter key | Enables closing the Variable Selector and displaying the selected variable in the workspace. |

You can customize the Variable Selector environment by arranging the columns to display and setting the display colors.

You can perform the following tasks from the Variable Selector:

- Creating Variables
- Editing Existing Variables
- Cutting, Copying, and Pasting Variables
- Deleting Variables
- Sorting Columns
- Filtering Variables

### To access the Variable Selector

The Variable Selector is available while editing language containers for POUs and displays only the variables available to the POU and the resource or device (depending on the CAM) containing the POU.

- From the language container of a graphical program, perform one of the following:
    - From the Toolbox, drag the variable element into the language container.
    - In the language container, double-click an existing variable.

The Variable Selector is displayed.

### To arrange the columns to display

1. To move a column, drag the column header to another location.
2. To hide a column, right-click a column header and then click **Hide Column**.
3. To show a column, right-click any column header, point to **Show Column**, then click the required column name.

### To set the display colors

To change the colors displayed in the Variable Selector you must apply the changes to the Dictionary Settings. You can customize the colors applied to the column headers, row headers, and rows. Note that the Variable Selector automatically alternates colored rows with white rows. Furthermore, you can adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row.

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand the **Grid Settings** node, then click **Variable Selector**.

3. In the *Misc* options, customize the required options:

   - To specify the number of consecutive rows for the alternating sequence, click **Alternate row value**, then type a value.

   - To change the colors applied to headers and rows, select the respective option, then select a color from the drop-down combo box.

# Creating Variables

Using the Variable Selector, you can create variables and insert variables into programs. You can also insert literal values into programs.

### To create a variable

You access the Variable Selector from language containers for opened programs.

1. In the Variable Selector, click the required tab, locate the empty row at the bottom of the grid.

   In the left-most column of the empty row, an asterisk (  ) is displayed.

2. In the cells of the empty row, enter the required information, then click **OK**.

The variable is displayed in the language container.

### To insert a literal value

You can insert literal values using Variable Selector. When inserting literal values that being with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'

- From the Variable Selector, in the *Name* field, type the literal value， then click **OK**.

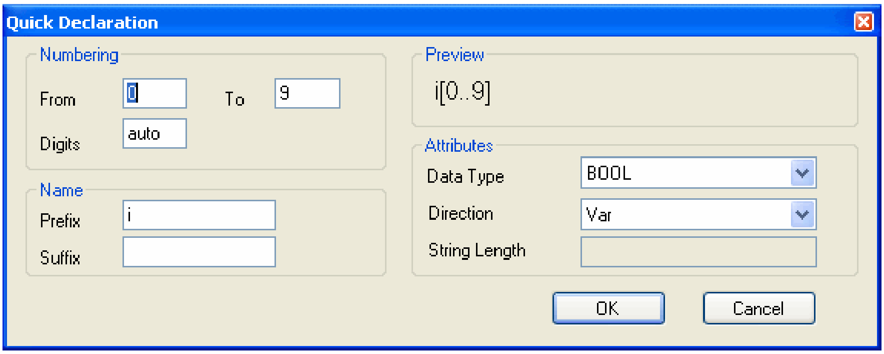
The literal value is displayed in the language container.

## See Also

Variable Selector

# Creating Multiple Variables Using Quick Declaration

Using the Quick Declaration dialog box, you can simultaneously create multiple local or global variables. Quick Declaration can be accessed using the Variable Selector. A preview of the variable is available on the top-right of the Quick Declaration dialog box.




The following attributes can be configured for variables in Quick Declaration:

| Property | Description |
|---|---|
| Numbering | The range of values for the variables. The digits option is set to auto by default and can be changed to alter the quantity of displayed digits. |

| Property | Description |
|---|---|
| Name | The variable name is separated into a prefix and suffix. The prefix appears before the number value and can contain letters, digits, and single underscores. The suffix appears after the number value and can contain letters, digits, and single underscores. Neither can contain two consecutive underscores. |
| Attributes | The following attributes are available:<br><br>Data type: Drop down combo box displaying the variable types. Possible values are elementary IEC 61131-3 types (BOOL, BYTE, DATE, DINT, DWORD, INT, LINT, LREAL, LWORD, REAL, SAFEBOOL, SINT, STRING, TIME, UDINT, UINT, ULINT, USINT, or WORD) or derived types (arrays, structures, or function blocks).<br><br>Direction: Indicates whether the variable is internal, input, or output. Possible values are Var, VarInput, or VarOutput.<br><br>String Length: Defined length only applying to the STRING variable. Possible values are 1 to 252. |

**To create multiple variables using Quick Declaration**

1. In the Variable Selector, select the tab in which you want to create variables.
2. Right click an empty row, and then click **Quick Declaration**.
3. Configure the variable attributes in the Quick Declaration dialog box.

# Editing Existing Variables

You can edit variables from the Variable Selector. The cells of the grid contain drop-down list boxes or editable text fields. To retain changes made to variables, you must save these changes.

**To edit a variable**

1. In the Variable Selector grid, locate the variable.
2. Select the grid cell to edit and make the necessary changes, then press ENTER.

**See Also**

Variable Selector

# Cutting, Copying, and Pasting Variables

You can cut, copy, and paste variables between the tabs of the Variable Selector as well as between the Variable Selector and instances of the Dictionary.

**To cut, copy, and paste variables**

When selecting variables, an indicator arrow (  ) is displayed in the leftmost column of the grid.

1. In the grid of the required Variable Selector tab or Dictionary instance, cut or copy the required variables.
   - To remove variables, select the required variable or variables, right-click the selection, then click **Cut**.
   - To copy variables, select the required variable or variables, right-click the selection, then click **Copy**.
2. In the grid of the required Variable Selector tab or Dictionary instance, right-click the desired location, then click **Paste**.

The variables are displayed at the desired location.

### See Also

Variable Selector

# Deleting Variables

You can delete variables from the Variable Selector. Deleting variables from the Variable Selector also removes these variables from the Dictionary.

**To delete variables**

- From the Variable Selector, in the grid, select the required variable or variables, right-click the selection, then click **Delete record(s)**.

**See Also**
Variable Selector

## Sorting Columns

You can sort the columns of the Variables selector in an ascending or descending order.

**To sort a column**

1. In the Variable Selector, select the required column header.
2. Toggle the column header to switch between ascending and descending order.

### See Also

Variable Selector

# Filtering Variables

You can filter the variables by their attributes in the Variable Selector. When filtering, you create a view displaying only the variables having specific attributes or containing specific characters.

You can filter the list of blocks by typing alphabetical and numerical characters in the *Name* field. The filter row is the top row of the grid. In the filter row, you can type alphabetical and numerical characters or select from the drop-down-combo-boxes. Variables containing matching characters are automatically displayed in the grid.

**To filter variables**

1. To filter using characters in variable names, in the *Name* field, type the characters to use in the filtering operation.

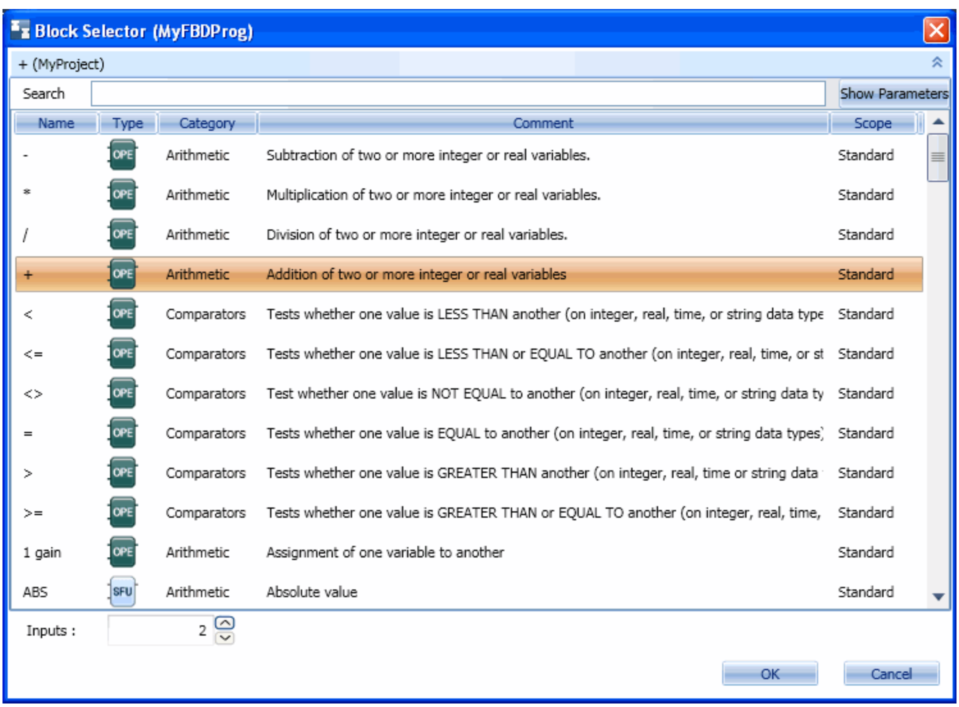
2. To filter using the variables attributes, in the filter row of the list of variables, click the required cell, then do one of the following:

   - Type the characters to use in the filtering operation
   - Select the required variable or filtering option from the drop-down combo-box

# Block Selector

The Block Selector enables the selection of operators, functions, and function blocks for use in block elements defined in programs. For FBD 61131 programs, you enter blocks and declared instances.

The Block Selector lists the available operators, functions, and function blocks for the program type: IEC 61131-3 or IEC 61499. For IEC 61131-3 programs, the available items are operators (OP), standard functions (SF), standard function blocks (SB), user IEC 61131-3 Functions (IFU), user IEC 61131-3 Function Blocks (IFB) and all "C" Functions (CFU) and “C” Function Blocks (CFB) supported by the target. For IEC 61499 programs, the displayed items are basic IEC 61499 Function Blocks (QFB) and composite IEC 61499 Function Blocks (QCF) for which instances are defined in the dictionary.




For the block list, the properties are the following:

| Column | Description |
|---|---|
| Name | Name of the function, function block, or operator |
| Type | Type of function, function block, or operator. Possible types are "C" function (CFU), "C" function block (CFB) , IEC 61131-3 function (IFU), IEC 61131-3 function block (IFB), operator (OPE), standard function block (SFB), and standard function (SFU). |
| Category | Category of function, function block, or operator. Possible categories vary depending on the target definition. |
| Comment | Comment for the function, function block, or operator. Free-format text. |
| Scope | Indicates where the POU is defined |

For IEC 61499 programs, after selecting a block, you need to indicate the instance of the IEC 61499 function block and the resource for which the instance is defined.

When selecting operators such as addition, multiplication, and AND, you need to specify the number of inputs. You can also force the inclusion of the EN and ENO parameters for blocks having either one Boolean input, one Boolean output, or no Boolean input and output. You activate the Enable EN/ENO option from the Ladder Diagram options.

Using the Block Selector, you can refine the list of available blocks by sorting the block list and limiting searches. You can also choose to display the parameters while viewing the blocks.

When working in the Block Selector, you can navigate using keyboard and mouse controls.

| | |
|---|---|
| Arrow keys | Enable moving up, down, left, and right within the cells of the blocks list |
| Tab key | Enables moving left and right within the cells of each row in the blocks list. After exiting the blocks list, enables moving from left to right between the fields, option, command buttons and back to the blocks list. |
| Esc key | Enables moving from the blocks list to the fields |
| Space bar | When the Show Parameters option is selected, opens the Parameters Display. |
| Enter key | Enables closing the Block Selector and displaying the chosen block in the workspace. |

From the Block Selector, you can access help for the displayed operators, functions, and function blocks.

### To access the Block Selector

The Block Selector is available while editing language containers for POUs and displays only the operators, functions, and function blocks available to the POU.

- From the language container of a graphical program, perform one of the following:
  - From the Toolbox, drag the block element into the language container.
  - In the language container, double-click an existing block.

The Block Selector is displayed.

### To create a declared instance of a function block

Declared instances are function blocks having assigned instances. For graphical and non-graphical programs, you declare such instances in the Block Selector. These instances are considered as variables.

1. From the list of available blocks, select the function block type.
2. In the *Instance* field, type a name for the instance, then click **OK**.

A declared instance of a block is displayed in the workspace.

### To sort the block list

You can sort the columns of the block list by setting these in ascending or descending order.

- Click the required column header to toggle the sort order between ascending and descending.

### To limit searches

As you type text in the Search field, the list displays only the blocks containing these characters.

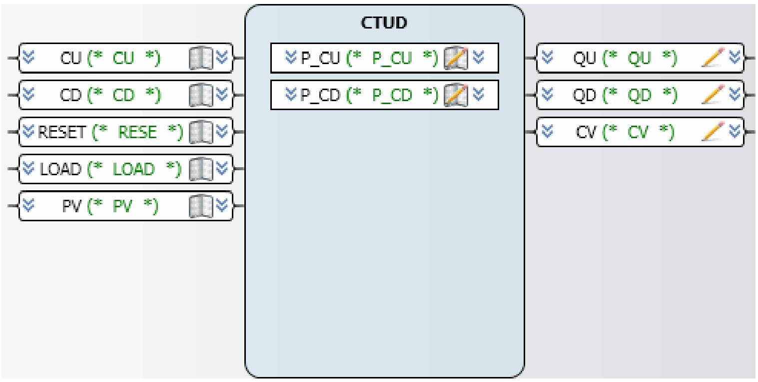
- In the Search field in of the Block Selector, type the required text.

### See Also

Parameters Display

## Parameters Display

The parameters display graphically presents the parameters for a POU selected in the Block Selector. When selecting a POU from the block list, the parameters display automatically shows the local, input, and output parameters. You can expand all or collapse all parameters for POUs.




**To access the parameter display**

- In the Block Selector, click **Show Parameters**.

The parameters display opens on the right.

**To expand or collapse all input and output parameters**

You can expand or collapse all input and output parameters for user-defined functions and function blocks.

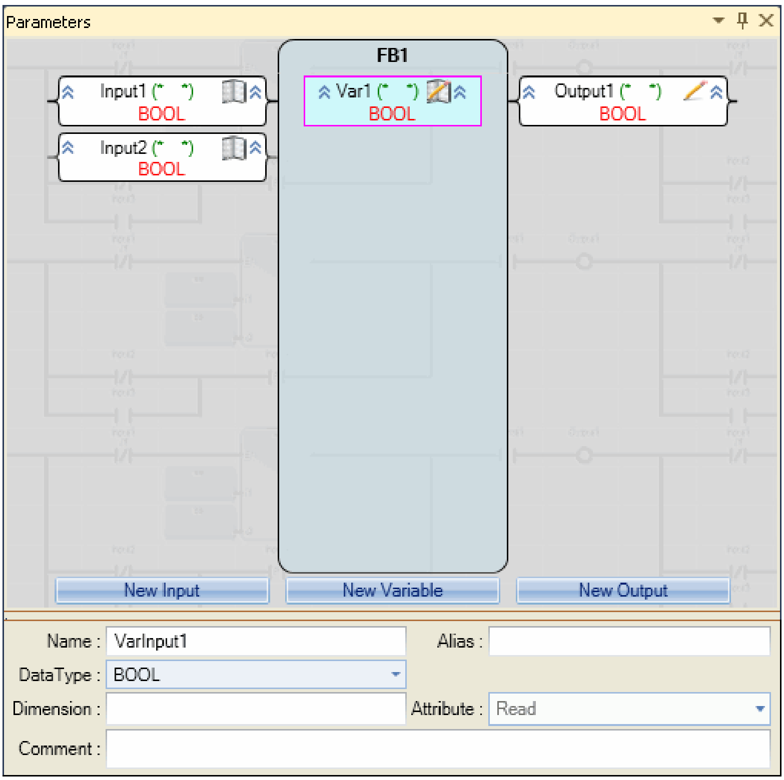
1. In the block list, select the required block for which to display the existing parameters.
2. To expand all parameters, right-click in the parameters display, then click **Expand All**.
3. To collapse all parameters, right-click in the parameters display, then click **Collapse All**.

### See Also

Block Selector

# Parameters View

The Parameters view enables managing parameter and local variables for user-defined POUs. When defining these POUs, the Parameters view provides a graphic view of the parameters and local variables. You can manage the parameters and local variables for user-defined POUs.




You can perform the following tasks from the Parameter view:

- Creating parameters or local variables
- Editing parameters or local variables
- Deleting parameters or local variables

- Cutting, copying, and pasting parameters and local variables
- Display data types for parameters or local variables

In the Parameters view, the properties of parameters and local variables varies for different CAMs (Concrete Automation Models):

| Column | Description |
|---|---|
| Name | Name of the parameter |
| Alias | The short name used in the graphical language editors for display only. Limited to four characters. |
| Data Type | Data type of the parameter |
| Dimension | For function blocks, dimension of the block. The dimension is defined as a positive double integer (DINT) value. |
| Attribute | Property of a parameter indicating its read and write access rights. Possible values are Read, Write, or ReadWrite. |
| Comment | Comment for the parameter. Free-format text. |

You can modify the parameters for functions and function blocks. User-defined functions are limited to one output parameter having modifiable data type.

**To access the parameter view for a user-defined function or function block**

You access the Parameters view when defining parameters for user-defined functions and function blocks.

1. From the Solution Explorer, create a user-defined function or function block in the Lib section.
2. Right-click the function or function block, then click **Parameters**.

**To create parameters and local variables**

You create parameters for a currently opened user-defined function or function block. Functions can only have one output.

1. In the Lib section of the Solution Explorer, right-click the required function or function block, then click **Parameters**.

   The Parameters view is displayed.

2. To add an input parameter, click **New Input**, then define the properties for the parameter.
3. To add an output parameter, click **New Output**, then define the properties for the parameter.
4. To add a local variable, click **New Variable**, then define the properties for the variable.

**To edit parameters and local variables**

You edit parameters and local variables for a currently opened user-defined function or function block.

1. In the Lib section of the Solution Explorer, right-click the required function or function block, then click **Parameters**.

   The Parameters view is displayed.

2. To edit a parameter, select the parameter, then modify its properties.
3. To edit a local variable, select the variable, then modify its properties.

**To delete parameters and local variables**

You delete parameters and local variables for a currently opened user-defined function or function block.

1. In the Lib section of the Solution Explorer, right-click the required function or function block, then click **Parameters**.

   The Parameters view is displayed.

2. Select the parameter or local variable to delete, right-click, then click **Delete**.

**To cut, copy, and paste parameters and local variables**

You can cut, copy, and paste parameters and local variables for a currently opened user-defined function or function block.

1. In the Parameters view for a user-defined function or function block, cut or copy the required parameter or local variable:
   - To remove the parameter or local variable, select the item, right-click and then click **Cut**.
   - To copy the parameter or local variable, select the item, right-click and then click **Copy**.
2. To paste a copied parameter or local variable, right-click in the Parameters view and then click **Paste**.

   Duplicated parameters or local variables are automatically placed in their respective area, i.e., input, output, or variable.

**To display data types for parameters and local variables**

You can expand and collapse the display of data types for all parameters and local variables of a currently opened user-defined function or function block.

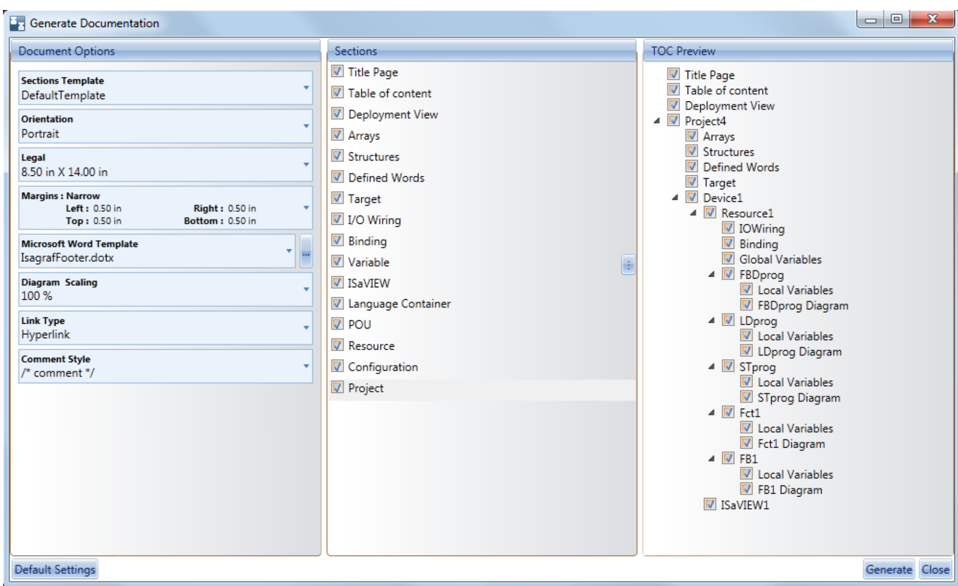
- In the Parameters view, right-click, and then click **Expand All**.

# Generating Documentation

While in design mode, you can generate documentation for projects, devices, resources (if supported by the CAM), POUs, variables, and library elements. The output format of the documentation is Microsoft Word® 2010 (*.docx). Generating documentation enables viewing the project information for a specific time. You can also search and edit the generated documentation.

**Note:** You need to have Microsoft Word® 2010 (or more recent) or another .docx application installed to properly view, search, and edit the generated documentation.

The Generate Documentation dialog box is separated into three panes: Document Options, Sections, and TOC Preview. Selections made in a pane affect what is displayed in the following pane (from left to right). Therefore, changes made in the Document Options pane affects the Sections pane and changes made in the Sections pane affects what is displayed in the TOC Preview pane.




In the Document Options pane you can set the following options:

| Option | Description | Possible Values |
|---|---|---|
| Sections Template | The template in XML format defining the Sections to be generated in the documentation as well as their hierarchy. The selected Sections Template affects the items displayed in the Sections and TOC Preview panes.<br><br>The templates are located in the following directory: %ALLUSERSPROFILE%\ISaGRAF\6.4\ACP\Templates | DefaultTemplate or a user-defined *.xml template. The default value is DefaultTemplate. |
| Orientation | The orientation of the page | Portrait or Landscape. The default value is Portrait. |
| Page size | The size of the page | Letter, Legal, Statement, Executive, A3, A4, A5, B4 (JIS), B5 (JIS), 11x17, Envelope #10, Envelope DL, Envelope C5, Envelope B5, Envelope Monarch, Japanese Postcard, A6, Double Japan Postcard Rotated, Executive (JIS), Oficio 8.5x13, 12x18, 8k 273x394 mm, 16k 197x273 mm, or Custom. The default value is Legal. |
| Margins | The left, right, top, and bottom margins for the page | Narrow, Normal, Moderate, or Custom. The custom margins range from 0 inches to the maximum size of the page. The default value is Narrow. |

| Option | Description | Possible Values |
|---|---|---|
| Microsoft Word® Template | The Microsoft Word® template in *.dotx format used to define the layout for the title page, table of contents, and tables.<br><br>The templates are located in the following directory: %ALLUSERSPROFILE%\ISaGRAF\6.4\ACP\Templates | IsagrafFooter.dotx or a user-defined *.dotx template. The default template is IsagrafFooter.dotx. |
| Diagram Scaling | The scaling for all diagrams displayed in the generated documentation. | 25%, 50%, 75%, 100%, 125%, 150%, 175%, 200%, 300%, 400%, 500%, Fit to Page, or Custom. When selecting the Custom scaling, a spin box appears enabling the user to select the scaling value. The default value is 100%. |
| Link Type | The type of links in the documentation. | None, Only Bookmarked, Cross Reference, or Hyperlink. The default value is Hyperlink. |
| Comment Style | How comments are displayed in the documentation. This option does not affect how comments are displayed in graphical POU diagrams. | // comment, /* comment */, or (* comment *). The default value is /* comment */. |

Selecting a Sections Template in the Document Options pane modifies the items listed in the Sections pane. The Title Page, Table of content, and Deployment View items are always available for selection in the Sections pane. The button displays the **Variable Settings** dialog box and is used to specify how you want the variables to be sorted in the generated documentation. You can sort variables by Name, Comment, Alias, Data Type, Wiring, Attribute, Dimension, Initial Value, Direction, or String Size in ascending or descending order.

The items selected in the Sections pane modifies the items displayed in the tree view of the TOC Preview pane. You can also select or clear items in the TOC Preview pane. The final selection in the TOC Preview pane displays what will be generated in the documentation.

The items displayed in the Sections and TOC Preview panes also depend on the element selected in the Workbench when using the Generate Documentation command. For example if a POU is selected, only associated sections (local variables and the POU diagram) are displayed in the Documentation Generator dialog box. When the project is selected in the Solution Explorer, all sections (project, global variables, defined words, structures, arrays, targets, etc) are displayed in the dialog box. If the Documentation Generator is unable to find an associated element, the Generate Documentation command does not appear in the File menu.

The Documentation Generator retains the selections made in the three panes for each element across project sessions. You can reset the pane selections by clicking **Default Settings**.

Users can create their own custom templates for the Sections Templates (*.xml) and Microsoft Word® templates (*.dotx). When creating a custom XML template, you must use the following syntax:

| Section | Description |
|---|---|
| TitlePageSection | The title page |
| TOCPageSection | The table of contents |
| SolutionSection | The title of the solution name |
| ProjectSection | The title of the project name |
| ArraysSection | The table displaying arrays |
| StructuresSection | The table displaying structures |
| DefinedWordsSection | The table displaying defined words |
| ConfigurationSection | The title of the controller name as well as the table displaying network links |
| ProgramSection | The title of the program name |
| POUContentSection | The POU diagrams |
| VariableSection | The tables for local and global variables. Also displays the extended attributes for global variables. |
| IOWiringSection | The I\O wiring table |
| TargetSection | The table displaying the targets |
| BindingSection | The table displaying the bindings |

When creating a custom Microsoft Word® template (*.dotx), you can modify how sections are displayed, but you must retain the following styles and table styles defined by Microsoft Word or the workbench:

| Style | Description |
|---|---|
| Heading 1 | How Header 1 is displayed in the documentation. |
| Heading 2 | How Header 2 is displayed in the documentation. |
| Heading 3 | How Header 3 is displayed in the documentation. |
| Heading 4 | How Header 4 is displayed in the documentation. |
| Heading 5 | How Header 5 is displayed in the documentation. |
| Heading 6 | How Header 6 is displayed in the documentation. |
| Heading 7 | How Header 7 is displayed in the documentation. |
| Heading 8 | How Header 8 is displayed in the documentation. |
| Heading 9 | How Header 9 is displayed in the documentation. |
| Alias | How the Alias section is displayed in the documentation. |
| Comment | How comments are displayed in the documentation. |

| Table Style | Description |
|---|---|
| IOWiring | The tables displaying information for I/O wiring and targets. |
| NormalStyle | The tables displaying information for bindings. |
| VariableTableStyle | The tables displaying information for variables, arrays, structures, and defined words. |

**To generate documentation**

You can only generate documentation while in design mode.

1. In the Solution Explorer, select the element (project, device, resource (if supported by the CAM), POU, library element, etc.) for which to generate documentation.
2. From the File menu, click **Generate Documentation**.
3. Specify the required options, then click **Generate**.

   The **Save As** dialog box is displayed.

4. In the **Save As** dialog box, specify the file name and save location, then click **Save**.
    - A progress bar shall appear over the Documentation Generator dialog box displaying the generation progress. The user can click **Cancel** to abort the documentation generation process. Any files created during the generation process are deleted.

Once generation is complete, the docx application displays the documentation.

# Find and Replace Utility

The Find and Replace utility enables performing the following operations:

- Quick Find
- Quick Replace

# Quick Find

You can find strings or expressions in files using the Quick Find utility. Quick Find steps from one search result to the next in sequence, either backwards or forwards from the insertion point. Upon reaching the end or beginning of a document, Quick Find automatically jumps to unsearched sections. When the search is complete, a message is displayed.

When all search options are defined, you can choose to find the next instances of the required string or expression within the specified scope.

- Find What, enables defining the string or expression to find within the open document. You can type the required string into the field, select one of the last twenty searches from the Find What drop-down combo-box, and use wildcards or regular expressions in searches. When using wildcards or regular expressions, the Expression Builder displays a list of available wildcards or expressions.

- Look in, enables defining the scope for the search. You can select the required scope from the Look in drop-down combo-box.

- Find Options, enables selecting options that refine the search. You can search for case sensitive matches using Match Case. You can disregard partial word matches by selecting Match whole word. You can search for matches from the insertion point to the top of the file by selecting Search up. You can search collapsed or concealed text by selecting Search hidden text. You can include special characters, such as wildcards or regular expressions, in the Find What field by selecting Use.

**To find a string or expression in a file**

You can perform searches using the Find and Replace utility or you can type the necessary text in the search field on the toolbar. You can place the cursor in the toolbar search field using the **Ctrl+D** keyboard shortcut.

1. From the Edit menu, point to **Find and Replace**, then click **Quick Find** (or press **Ctrl+F**).

2. In the Quick Find utility, enter the required information, then click **Find Next** (or press **F3**).

**To use wildcards or regular expressions**

1. From Quick Find, expand **Find Options**, then select **Use**.

2. From the *Use* drop-down combo-box, select the required option, either **Wildcards** or **Regular expressions**.

3. In the *Find What* field, type the required wildcard or regular expression, or click  to select from the list of available wildcards or regular expressions.

### See Also
Quick Replace

# Quick Replace

You can replace strings or expressions in files using the Quick Replace utility. Quick Replace steps from one search result to the next in sequence, either backwards or forwards from the insertion point. Upon reaching the end or beginning of a document, Quick Replace automatically jumps to unsearched sections. When the search is complete, a message is displayed.

When all search options are defined, you can choose to find the next instance of the required string or expression within the specified scope, then replace individual or all instances of searched items.

- Find What, enables defining the string or expression to find within the open document. You can type the required string or expression in the field, select one of the last twenty searches from the Find What drop-down combo-box, and use wildcards or regular expressions in searches. When using wildcards or regular expressions, the Expression Builder displays a list of available wildcards or expressions.
- Replace with, enables defining the string or expression that will replace each match found. You can type the required string or expression in the field provided, or select one of the last twenty items entered using the drop-down combo-box. You can delete matches found by leaving the Replace with field empty. You can use wildcards or regular expressions in the Replace with field.
- Look in, enables defining the scope for the search. You can select the required scope from the Look in drop-down combo-box.
- Find Options, enables selecting options that refine the search. You can search for case sensitive matches using Match Case. You can disregard partial word matches by selecting Match whole word. You can search for matches from the insertion point to the top of the file by selecting Search up. You can search collapsed or concealed text by selecting Search hidden text. You can include special characters, such as wildcards or regular expressions, in the Find What field by selecting Use.

**To replace a string or expression in a file**

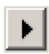
1. From the Edit menu, point to **Find and Replace**, then click **Quick Replace** (or press **Ctrl+H**).

2. In the Quick Replace utility, enter the required information, then click one of the following command buttons: **Find Next**, **Replace**, or **Replace All**.

**To use wildcards or regular expressions**

1. From Quick Replace, expand **Find Options**, then select **Use**.

2. From the *Use* drop-down combo-box, select the required option, either **Wildcards** or **Regular expressions**.

3. In the *Find What* or *Replace with* fields, type the required wildcard or regular expression, or click  to select from the list of available wildcards or regular expressions.

### See Also

Quick Find

# Spy Lists

You can choose to spy on selected variables and instances of function blocks, i.e., view changes in the values for these variables and function blocks. You spy on variables and instances of function blocks by adding these to spy lists. Before adding these, you need to create a spy list.

You view spy lists in the Spy List window.

When managing spy lists, you can perform the following tasks:

- Accessing existing spy lists
- Adding items to a list
- Removing items from a list
- Saving spy lists
- Cutting, copying, and pasting items between spy lists
- Dragging items between Spy Lists

For the Spy List, the properties are the following:

| Column | Description | Possible Values |
|---|---|---|
| Name | Name of the variable or function block instance | Limited to 128 characters beginning with a letter or underscore character followed by letters, digits, and single underscore characters. These names cannot have two consecutive underscore characters. |
| Alias | Any name (for use in LD POUs) | Limited to 128 characters beginning with a letter or underscore character followed by letters, digits, and single underscore characters. These cannot have two consecutive underscore characters. |

| Column | Description | Possible Values |
|---|---|---|
| Logical Value | Available when online. The displayed value differs depending on the direction of the variable or function block instance. | Input: Locked<br>Output: Updated by the running TIC code<br>Internal: Locked |
| Physical Value | Available when online. The displayed value differs depending on the direction of the variable or function block instance. | Input: Updated by the field value<br>Output: Locked<br>Internal: Updated by the running TIC code |
| Lock | Available when online. The indication of whether the value of the variable or function block instance is locked. Locking operates differently for simple variables, array and structure elements, and function block parameters. For simple variables, individual variables are locked directly. For structure and array elements, locking an element locks all the elements of the structure or array. | Yes or No |
| Comment | User-defined text | Free format |
| Access Path | The location of the variable or function block instance within the project. | Name of the project, device, resource (if supported by the CAM), and program is displayed, as well as the name of the variable or function block instance. |

You can also customize spy lists by arranging the columns to display and setting the display colors. In the Spy List, you can refine the contents of the grid by grouping items in a list, sorting items in a list, and filtering items in a list.

For spy lists, the properties are the following:

| | |
|---|---|
| Monitoring Refresh Rate | The rate at which the values of variables are refreshed in the spy list, in milliseconds. You can only change the refresh rate while in design mode. |
| Spy List Name | Name of the spy list displayed in the spy list title bar and the menu |

When working in the Spy List, you can navigate using the mouse controls and arrow keys to move up and down the list.

| | |
|---|---|
| Arrow keys | Enable moving up or down in the list |
| Enter key | When selecting variables using the Name field, enables saving the selected variable to the grid. |

**To create a spy list**

- From the Debug menu, point to **Spy List**, then click **Create Spy List**.

A spy list having an empty grid is displayed.

**To access an existing spy list**

- From the Debug menu, point to **Spy List**, then click the required list from the available spy lists.

**To add items to a list**

In a spy list, you add variables and instances of function blocks to the list individually.

- In the name column of the list, double-click the available record row, then select a variable or function block instance from the drop-down menu.

**To remove items from a list**

You can delete one or more variables and instances of function blocks from a spy list. When selecting an item, an indicator arrow is displayed in the left-most column of the list.

- In the list, select the item or items to delete, right-click the selection, then click **Delete**.

The items are removed from the list.

**To save a spy list**

Changes to spy lists are saved automatically upon closing.

- From the required spy list, click the Close button at the top-left corner of the Spy List window.

**To cut, copy, and paste items between spy lists**

You can cut, copy, and paste variables and instances of function blocks between spy lists. When selecting these items, an indicator arrow is displayed in the leftmost column of the list.

**1.** In the grid of the required spy list, cut or copy the required items.

   - To remove variables, select the required item or items, right-click the selection, then click **Cut**.
   - To copy variables, select the required item or items, right-click the selection, then click **Copy**.

**2.** In the grid of the required spy list, right-click the required location, then click **Paste**.

The items are displayed at the desired location.

**To drag items between spy lists**

You can drag variables and instances of function blocks from one spy list to another.

**1.** Access the spy lists containing the required items and their destination.

**2.** From the spy list containing the required items, select the items.

   The selection indicator is displayed in the leftmost column.

**3.** Drag to the destination, placing it at the required location within the list.

The items are displayed at the destination.

**To arrange the columns to display**

1. To move a column, drag the column header to another location.

   When dragging a column header, arrows indicate the current position of the header.

2. To show or hide a column, right-click on a column header, then click the column name.

**To sort items in a spy list**

You can sort items in a spy list according to the ascending or descending order for the different columns.

- Click the required column header to toggle the sort order between ascending and descending.

**To filter items in the grid**

You can filter variables and function block instances displayed in a list. When filtering, you create a view displaying only the entries containing specified characters.

The filter row is the top row of the grid. You can filter variables and function block instances by typing alphabetical and numerical characters in the cells of the filter row. You can also select from the drop-down-combo box. Matching variables and function block instances are automatically displayed.

- In the filter row of the Spy List, click the required cell, then do one of the following:
  - Type the characters to use in the filtering operation
  - Select the required structure from the drop-down combo-box

**To group items in a spy list**

You can group items contained in a spy list according to columns.

- Drag the required column header to toggle the sort order between ascending and descending.

# Add-in Manager

The Add-in manager enables specifying the loading method of available, i.e., registered, add-ins. The Add-in manager dialog box lists the available add-ins for which you specify whether to load at startup or using a command line. The dialog box also displays descriptions defined for add-ins.

At startup or build time, when add-ins are set to load using command line switches, those having user interfaces are automatically displayed. Add-ins displaying as toolbar icons or menu commands are also displayed within the toolbars and menus. When add-ins are set to load at startup time, you can stop the add-in from loading by pressing and holding SHIFT during startup. Add-ins having user interfaces remain accessible from toolbars and menus.

For projects containing add-ins, you can avoid errors when moving a project to another location by updating its paths in the following tag of the respective **.Addin* XML file:

```
<Assembly>C:\MyAddin1.dll</Assembly>
```

When working in the Add-in Manager dialog box, you can toggle the selection of the loading options using keyboard shortcuts for a selected add-in: Startup option using ALT+S and Command Line option using ALT+C.

**To access the Add-in Manager**

- From the Tools menu, click **Add-In Manager**.

**To set the loading behavior for an add-in**

1. From the Add-In Manager, in the *Available Add-ins* column, click the check-box next to the add-in name, then perform the following as required:
   - To load the add-in at startup, click the check-box in the *Startup* column.
   - To load the add-in using a command line, click the check-box in the *Command Line* column.
2. Click **OK**.

# External Tools

You can launch external tools and applications by adding items to the Tools menu. You can also create keyboard shortcuts for external tools added to the Tools menu. Supported file types include .exe, .bat, .com, .cmd, and .pif.

From the External Tools dialog box, you can perform the following tasks:

- Adding an external tool
- Specifying a tool for handling arguments
- Defining a working directory

When specifying a tool for handling arguments, the required argument is immediately transferred to the tool when the external tool is launched. At this time, you can also choose to edit required arguments. Upon subsequent startups of the external tool from the Tools menu, selected arguments are automatically passed to the tool. When Prompt for Arguments is selected, the Arguments dialog box is displayed.

You can define a working directory for tools or commands. You can also specify additional arguments when the command is launched.

**To add an external tool**

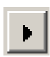
1. From the Tools menu, click **External Tools**.
2. In the External Tools dialog box, in the *Title* field, type a name for the menu option. To include a keyboard shortcut, type an ampersand (&) before the letter in the title to use as shortcut. For example: "My External Tool", the letter "x" is the keyboard shortcut.
3. In the *Command* field, type the path to the file, or browse for the file by clicking .
4. Select the **Use Output window** and **Close on exit** check boxes (optional).

   The *Use Output window* option is only available for .bat and .com files.
5. Click **Add**, then click **OK**.

The external tool is available from the Tools menu.

**To specify a tool for handling arguments**

When the specified tool is launched, the required argument is immediately transferred to the tool. Selecting the Prompt for Arguments option enables editing the argument at launch time.

1. From the Tools menu, click **External Tools**.
2. In the External Tools dialog box, in the *Menu contents* list, select the required tool.
3. In the *Arguments* field, type the required arguments, or select a predefined argument by clicking .
4. Select **Prompt for arguments** (optional), click **Apply**, then click **OK**.

**To define a working directory**

Selecting the Prompt for Arguments option enables adding additional arguments at launch time.

1. From the Tools menu, click **External Tools**.
2. In the External Tools dialog box, in the *Menu contents* list, select the required tool.
3. In the *Initial directory* field, enter the working directory for the tool, or select a predefined directory path by clicking .
4. Select **Prompt for arguments** (optional), click **Apply**, then click **OK**.

# Working in the Development Environment

When working in the development environment, you can use keyboard shortcut combinations to perform multiple tasks. These tasks include customizing, creating, and renaming toolbars. You can also customize commands and edit buttons. Navigating in the development environment is simplified with the use of the Integrated Development Environment (IDE) Navigator.

# Displaying the Output Window

You can review messages generated by various features of the Workbench by accessing the Output window. From the Output window, you can perform the following tasks:

- Reviewing status messages
- Managing the contents of the window

The Output window toolbar contains the following commands:

| | |
|---|---|
| Show output from: | Enables selecting individual features for which to view generated status messages |
|  Go to Previous Message | In the Output window, jumps to the previous build error message. In the code editor, locates the build error and automatically moves the insertion point to the error. |
|  Go to Next Message | In the Output window, jumps to the next build error message. In the code editor, locates the build error and automatically moves the insertion point to the error. |
|  Clear all | In the Output window, deletes all displayed messages. |
|  Toggle Word Wrap | Wraps text to continue on the next line for messages extending beyond the viewing area |

**To access the Output window**

**1.** From the View menu, click **Output** (or press **Ctrl+Alt+O**).

The Output window is displayed.

**To review the generated status messages**

**1.** In the Output window, from the *Show output from* drop-down combo-box, click the required feature.

The status messages are displayed.

**To manage the contents of the Output window**

You can manage the word wrapping and clear the contents of the window.

**1.** To wrap text to continue on the next line, click .

**2.** To delete the contents of the window, click .

# Using the Error List

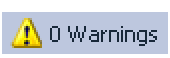
You can view the errors, warnings, and messages produced when you edit programs and perform build operations by accessing the Error List window.

From the Error List window, you can navigate from one error to the next using the contextual menu options. You can also navigate between errors using the keyboard arrows.

| Column | Description |
|---|---|
| Category | Displays an icon identifying the type of error |
| Default Order | Displays an integer indicating the order in which the error occurred relative to the other errors |
| Description | Displays the error message text |
| File | Displays the program name or the program location and program name |
| Line | Displays the line number |
| Column | Displays the column number |
| Project | Displays the name of the project |

The Error List toolbar contains the following commands:


Displays the number of generated errors. Click to toggle between displaying and hiding the errors in the list.




Displays the number of generated warnings. Click to toggle between displaying and hiding the warnings in the list.


Displays the number of generated messages. Click to toggle between displaying and hiding the messages in the list.

You can sort the contents of the Error List. You can customize the Error List by hiding columns, resizing columns, and arranging the columns to display.

**To display the Error List window**

- From the View menu, click **Error List** (or press **Ctrl+\, Ctrl+E**).

The error list is displayed.

**To sort the errors**

You can sort the list of displayed errors.

- In the Error List window, click the required column heading for which to sort. To further sort the list, click another column heading while pressing SHIFT.

**To customize the Error List window**

1. To move a column, drag the column heading to the required location.

2. To modify the width of columns, drag the column dividers to the required location.

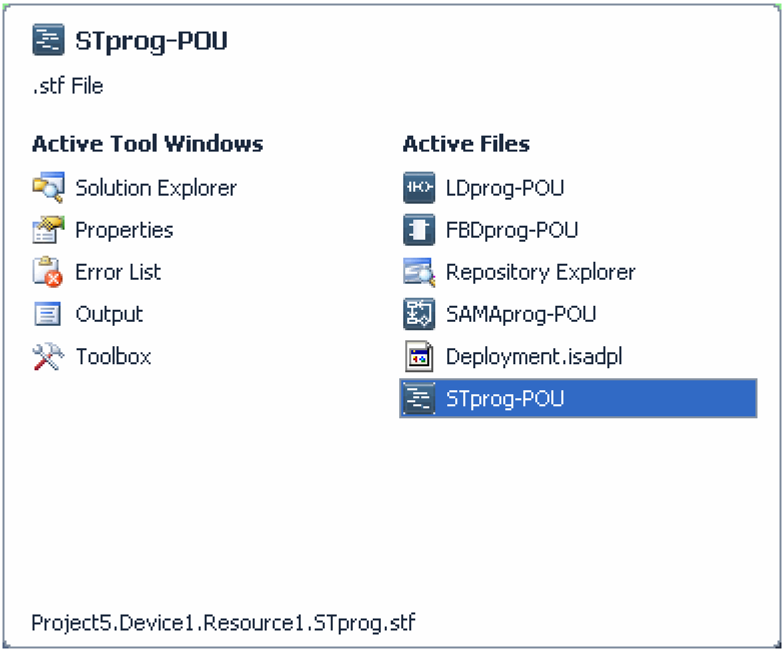
# Navigating in the Development Environment

Navigating in the development environment is simplified with the use of the following utilities:

- Integrated Development Environment (IDE) Navigator
- Windows Dialog Box

## Integrated Development Environment (IDE) Navigator

The IDE Navigator lists all Active Files and Tool Windows open in the current project. The navigator enables navigation between Active Files and navigation between Active Tool Windows. You can only access the IDE Navigator using keyboard shortcuts.




When using the IDE Navigator, the currently selected file is displayed on the top right of the navigator. The file type is displayed under the file name when applicable. The full path of the selected window or file is located at the bottom of the navigator.

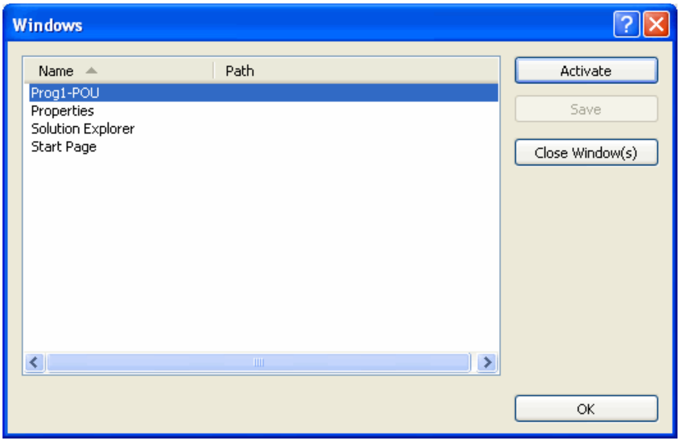
Active Tool Windows consist of windows docked around the workspace or undocked windows. Active Files consist of language containers, the deployment view, and other windows docked in the workspace. You navigate between the different files using keyboard shortcuts or the arrow keys.

The order in which the Active Files and Active Tool Windows are displayed depends on activation. The first file is the most recently used/selected while the last file is the least recently used.

**Note:** Using a different set of keyboard shortcuts, you can navigate between Active Files and navigate between Active Tool Windows without displaying the IDE Navigator.

### Windows Dialog Box

The Windows dialog box displays the active files open in the current project. Active files consist of language containers, the deployment view, and other windows docked in the workspace.




From the Windows dialog box, you can perform the following management tasks for active files:

- Switch between active files
- Save changes to one or more active files

- Close active files

**To navigate using the Windows dialog box**

1. From the Window menu, click **Windows**.

   The Windows dialog box displays the list of active files.

2. To switch to another active file in the list, select the required file, then click **Activate**.

3. To save changes to active files, select the required files from the list, then click **Save**.

4. To close active files, select the required files from the list, then click **Close Window(s)**.

**See Also**
Development Environment Keyboard Shortcuts

# Customizing Toolbars

For toolbars provided with **ISaGRAF 6**, you can modify docking locations. For custom toolbars, you can modify docking locations, rename toolbars, and delete toolbars.

**To customize a toolbar**

The Customize dialog box lists the provided toolbars as well as any custom user toolbars.

1. From the Tools menu, click **Customize**.
2. From the Customize dialog box, click the **Toolbars** tab, make the required changes, then click **Close**.
   - To modify the docking location for a toolbar, select the required toolbar from the *Toolbars* list, click Modify Selection, then click the preferred location for docking the toolbar. Available docking locations are top, left, right, and bottom.
   - To rename a custom toolbar, select the required toolbar from the *Toolbars* list, click **Modify Selection**, then type the required name in the text field.
   - To delete a custom toolbar, select the required toolbar from the *Toolbars* list, then click **Delete**.

     The toolbar is removed from the *Toolbars* list.

### See Also
Creating Toolbars

# Creating Toolbars

You can create custom toolbars for use in the workbench.

**To create a custom toolbar**

1. From the Tools menu, click **Customize**.
2. From the Customize dialog box, click the **Toolbars** tab, then click **New**.
3. In the New Toolbar dialog box, type a name for the custom toolbar, then click **OK**.

The custom toolbar name is added to the *Toolbars* list.

### See Also

Customizing Toolbars

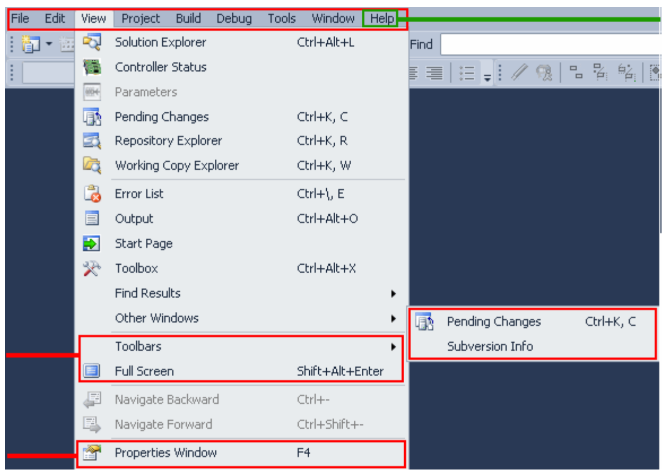
# Customizing Commands

You can customize menu bar, toolbar, and contextual menu commands by selecting a set of commands, then choosing an individual command to modify using the available options. You can add, rename, reset, delete, and rearrange the order of commands in the menus. You can also delimit groups of commands in menus and specify display options.

When customizing menus, the following image shows the different levels and options for menu items.

Menu Bar

Menu Category




Menu Items

Submenu

Commands

**To add a menu category to the menu bar**

1. From the Tools menu, click **Customize**.
2. From the Customize dialog box, click the **Commands** tab.
3. Select *Menu Bar* from the Menu bar drop-down combo-box.

4. To add a menu category to the menu bar, click **Add New Menu**.

   The menu category is added to the menu bar.

5. Rename the menu item by clicking **Modify Selection**, then typing the required name in the text field.

**To add a menu item to an existing menu category, toolbar, or contextual menu.**

Menu items are either commands or subcategories leading to submenus. Before adding a menu item, you need to arrange the required order by selecting the menu item following the location of the new item in the list or rearranging the menu items after insertion.

1. From the Tools menu, click **Customize**.
2. From the Customize dialog box, click the **Commands** tab.
3. Select the required menu from the Menu bar, Toolbar, or Context menu drop-down combo-boxes.
4. Perform one of the following operations:
   - To add a menu item to an existing menu category, toolbar, or contextual menu, select the item following the location for the new item, then click **Add New Menu**.
   - To add a command to an existing menu category, toolbar, or contextual menu, select the item following the location for the new item, click **Add Command**, then select the category and choose from the available commands in the Commands list.

   The menu item is added to the existing menu category, toolbar, or contextual menu.

5. To rename the menu or command, click **Modify Selection**, then type the required name in the text field.

**To reset menu bars, toolbars, or contextual menus**

1. From the Tools menu, click **Customize**.
2. From the Customize dialog box, click the **Commands** tab.
3. Perform the required reset operation:

- To reset a command or menu item, select the item from the respective drop-down combo-box, select the command or menu item from the Controls list, click **Modify Selection**, then click **Reset**.
- To reset a menu, toolbar, or contextual menu, select the item from the respective drop-down combo-box, then click **Reset All.**

**To delete a menu item, toolbar, or contextual menu**

1. From the Tools menu, click **Customize**.
2. From the Customize dialog box, click the **Commands** tab.
3. Select the required menu from the Menu bar, Toolbar, or Context menu drop-down combo-boxes.
4. In the Controls list, select the item to delete, then click **Delete**.

**To create a group of commands**

You can create groups of commands by inserting separator bars.

1. From the Tools menu, click **Customize**.
2. From the Customize dialog box, click the **Commands** tab.
3. Select the required menu from the Menu bar, Toolbar, or Context menu drop-down combo-boxes.
4. From the Controls list, select the menu item starting the group, click **Modify Selection**, then click **Begin a Group**.

   A separator bar is inserted before the selected menu item.

**To rearrange menu items**

1. From the Tools menu, click **Customize**.
2. From the Customize dialog box, click the **Commands** tab.

3. Select the required menu from the Menu bar, Toolbar, or Context menu drop-down combo-boxes.

4. To place the menu item at a different location in the selected menu or toolbar, select the menu item in the Controls list, then click **Move Up** or **Move Down** to move across the existing menu items.

**To specify the display options for a command**

Initially, the display options for commands are set to default. In menus, the default display option is Image and Text, while in toolbars it is Text Only (in Menus). The Text Only (in Menus) option displays an image in a toolbar or text in a menu. The Text Only (Always) option displays text in a menu or toolbar. The Image and Text option displays both image and text in a menu or toolbar. A command may not have an associated image.

1. From the Tools menu, click **Customize**.

2. From the Customize dialog box, click the **Commands** tab

3. Select the menu to modify from the Menu bar, Toolbar, or Context menu drop-down combo-boxes.

4. To specify the display options, select the required command in the Controls list, click **Modify Selection**, then click one of the following:

   - Default style
   - Text Only (Always)
   - Text Only (in Menus)
   - Image and Text

# Importing and Exporting Settings

You can import or export specific categories of settings, or reset the environment to one of the default collections of settings. The environment settings include the settings for the various development views, editors, and tools.

- Export Selected Environment Settings
- Import Selected Environment Settings
- Reset all Settings

**To import, export, or reset environment settings**

1. From the Tools menu, click **Import and Export Settings...**
2. Select the required option, then follow the on-screen instructions.

## Export Selected Environment Settings

When exporting selected environment settings, you need to choose the settings to export from the list of available environment settings. Environment settings identified with a warning symbol are not selected by default since these may contain intellectual property or sensitive information. Some categories may have sub-categories visible upon expanding the arrows to the left of the category item.

The settings exportation process requires the following operations:

**1.** Choosing the environment settings to export.

**2.** Naming a settings file.

During the environment settings export process, a window indicates the progress of the operation. Upon completion of the environment settings export process, a summary page indicates the results of the operation.

**See Also**
Import Selected Environment Settings
Reset all Settings

## Naming a Settings File

When exporting selected environment settings, you need to specify a settings file in which to store the exported settings. The default location of this settings file is the following:

%USERPROFILE%\documents\isagraf 6.4\Settings\ISaGRAF

**See Also**

Export Selected Environment Settings

## Settings Export in Progress

During the environment settings export process, a window indicates the progress of the operation.

## Import Selected Environment Settings

When importing selected environment settings, you need to choose a file containing the settings to import, then select the required settings to import from the list of available environment settings in the file. Environment settings identified with a warning symbol are not selected by default since these may contain intellectual property or sensitive information. Some categories may have sub-categories visible upon expanding the arrows to the left of the category item.

The settings importation process requires the following operations:

1. Choosing whether to save the current environmental settings or overwriting the current setting with the settings to import.

2. Choosing a file containing the collection of environmental settings to import.

3. Selecting the individual settings to import from the list of available environment settings in the settings file.

During the environment settings import process, a dialogue indicates the progress of the operation. Upon completion of the environment settings import process, a summary page indicates the results of the operation.

**See Also**

Export Selected Environment Settings
Reset all Settings

## Choosing a Collection of Settings to Import

When importing selected environment settings, you need to choose a file containing the settings to import.

**See Also**
Choosing Settings to Import
Import Selected Environment Settings

## Choosing Settings to Import

When importing selected environment settings, you can select the required settings to import from the list of available environment settings contained in the settings file. Environment settings identified with a warning symbol are not selected by default since these may contain intellectual property or sensitive information. Some categories may have sub-categories visible upon expanding the arrows to the left of the category item.

### See Also

Choosing a Collection of Settings to Import
Import Selected Environment Settings

## Settings Import in Progress

During the environment settings import process, a window indicates the progress of the operation.

## Reset all Settings

You can revert the environment settings to the initial settings. When resetting the environment settings, you can choose whether to save the current environment settings to a file.

**See Also**
Import Selected Environment Settings

## Settings Reset in Progress

During the environment settings reset process, a window indicates the progress of the operation.

## Operations Summary

When performing one of the following tasks regarding the environment settings, the wizard informs you of the results (whether successful or unsuccessful) for the operation.

- Export Selected Environment Settings
- Import Selected Environment Settings
- Reset all Settings

# Development Environment Keyboard Shortcuts

When working in the development environment, keyboard shortcuts are available for the following tasks:

- AccessingWindows
- Debugging
- Dictionary
- Getting Help
- Saving and Closing
- Working with the Cross Reference Browser and Find utility
- Navigating in the Development Environment

Some keyboard shortcuts do not apply or may differ while debugging.

**Note:** Keyboard shortcuts specific to the programming languages, deployment view, and version source control are indicated on their respective keyboard shortcut pages.

**Accessing Windows**

| | |
|---|---|
| Ctrl+Alt+T | Accesses the Block Library |
| Ctrl+W, Ctrl+C | Accesses the Cross Reference Browser |
| Ctrl+\, Ctrl+E | Accesses the Error List window |
| Ctrl+Shift+N | Accesses the New Project dialog box (not available while debugging) |
| Ctrl+Shift+O | Accesses the Open Project dialog box (not available while debugging) |
| Ctrl+Alt+O | Accesses the Output window |
| Ctrl+K, C | Accesses the Pending Changes window |
| F4 | Accesses the Properties window |
| Alt+Enter | Accesses the Properties window |
| Ctrl+F | Accesses the Quick Find utility |
| Ctrl+H | Accesses the Quick Replace utility |

| | |
|---|---|
| Ctrl+K, R | Accesses the Repository Explorer |
| Ctrl+Alt+L | Accesses the Solution Explorer |
| Ctrl+Alt+X | Accesses the Toolbox |
| Ctrl+K, W | Accesses the Working Copy Explorer |

**Debugging**

| | |
|---|---|
| Ctrl+Shift+B | Builds the solution (not available while debugging) |
| F5 | Starts debugging |
| F10 | While debugging, steps over the next rung or line of code |
| F11 | While debugging, steps into the next rung or line of code |
| Shift+F5 | Stops debugging |
| Ctrl+D | Only available in debug mode for the date data type. When the Write Logical Value dialog box is open, enters the current date. |

**Dictionary**

| | |
|---|---|
| Up Arrow | Moves up the grid between cells |
| Down Arrow | Moves down the grid between cells |
| Left Arrow | Moves left across the grid between cells |
| Right Arrow | Moves right across the grid between cells |
| Ctrl+PLUS SIGN on numeric keypad (+) | Expands the fields of complex data types |
| Ctrl+MINUS SIGN on numeric keypad (-) | Collapses the fields of complex data types |

**Getting Help**

| | |
|---|---|
| Ctrl+F1 | Accesses the Help Viewer |
| Shift+F1 | Accesses help for the selected window |
| F1 | Accesses help for the selected element |

**Saving and Closing**

| | |
|---|---|
| Ctrl+S | Saves the selected elements (not available while debugging) |
| Ctrl+Shift+S | Saves all files making up a solution (not available while debugging) |
| Alt+F4 | Exits ISaGRAF |
| Ctrl+F4 | Closes files and windows located in the workspace |
| Shift+Esc | Closes selected windows except for programs and the deployment view |

**Working with the Cross Reference Browser and Find utility**

| | |
|---|---|
| Ctrl+T, Ctrl+R | Refreshes the Cross Reference Browser data |
| F8 | Jumps to the selected instance of an element |
| Shift+F8 | Jumps to the selected instance of an element |
| F3 | Finds next text in a selected window |
| Ctrl+D | Goes to the find toolbar. Current text is highlighted/selected. |

**Navigating in the Development Environment**

| | |
|---|---|
| Alt+- | For language containers and the deployment view, displays various menu options including saving, docking, and tiling.<br>For other windows, displays docking options. |
| Ctrl+Alt+Down Arrow | Displays a drop down list on the top right corner of the workspace listing all active files tabbed in the workspace. |
| Ctrl+Tab | Displays the IDE Navigator. You can navigate to the next Active File by holding the Ctrl key and pressing Tab. Releasing the keys selects the current file. |
| Ctrl+Shift+Tab | Displays the IDE Navigator. You can navigate to the previous Active File by holding Ctrl+Shift and pressing Tab. Releasing the keys selects the current file. |
| Alt+F7 | Displays the IDE Navigator. You can navigate to the next Active Tool Window by holding Alt and pressing F7. Releasing the keys selects the current window. |

| | |
|---|---|
| Alt+Shift+F7 | Displays the IDE Navigator. You can navigate to the previous Active File by holding Alt+Shift and pressing F7. Releasing the keys selects the current window. |
| Ctrl+F6 | Navigates to the next Active File |
| Ctrl+Shift+F6 | Navigates to the previous Active File |
| Alt+F6 | Navigates to the next Active Tool Window |
| Alt+Shift+F6 | Navigates to the previous Active Tool Window |

# Options for the Development Environment

When setting the development options, you can customize the following aspects of the development environment:

- Setting Environment Options
- Specifying Project Options
- Specifying Source Control Settings
- Specifying Block Library Settings
- Specifying CAM3 Settings
- Specifying Deployment View Settings
- Specifying Device View Options
- Specifying Documentation Generator Options
- Setting Grid Options
- Defining CAM 3 I/O Device Settings
- Defining CAM 5 I/O Device Settings
- Setting IEC Language Options
- Setting ISaVIEW Options
- Defining Spy List Settings

## Setting Environment Options

You can define the environment options for the following:

- Find and Replace
- Fonts and Colors
- Import and Export Settings
- International Settings
- Keyboard
- Startup

You can modify the general settings for the workbench by accessing the general environment options. Some changes to the general settings take effect after restarting the workbench.

- Recent files, enables defining the number of recently used files displayed in menus. The items shown in Window menu field defines the number of windows (ranging from 1 to 24) displayed in the Windows list of the Window menu. For the number of items shown in the Window menu, the default is 10. The items shown in recently used lists field defines the number of recent projects and files (ranging from 1 to 24) displayed in the File menu. For the number of items shown in recently used lists, the default is 6.

- Visual experience, specifies whether the visual experience is set automatically or explicitly. This adjustment may change the display of colors from gradients to flat colors, or it may restrict the use of animations in menus or popup windows. Enabling the full visual experience includes gradients and animations. Clear this option when using remote desktop connections or older graphics adapters because these features may have poor performance in such cases. Use hardware graphics acceleration if available rather than software acceleration.

- Show status bar, enables displaying the status bar. The status bar displays progress information for ongoing operations.

- Close button affects active tool window only, enables the Close button to shut down the active window only. This option is selected by default.

- Auto Hide button affects active tool window only, enables Auto Hide to hide the active window only.

- Restore File Associations, registers file types that are normally associated with the workbench. When uninstalling or reinstalling versions of the workbench, Restore File Associations enables Microsoft® Windows to display the correct icons in Windows Explorer, and to recognize the workbench as the default application for editing workbench related files.

**To access the general environment options**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Environment**, then click **General**.

The general environment options are displayed in the Options dialog box

## Find and Replace

You can define the display settings for the Find and Replace dialog box. You can choose to display informational messages and warnings as well as populate the Find What field with text from an open editor. You can also choose to hide the Find and Replace dialog box once a match is found.

**To define the display settings for the Find and Replace dialog box**

1. From the Tools menu, click **Options**.

2. In the Options dialog box, expand **Environment**, then click **Find and Replace**.

3. In the Options dialog box, select the required options, then click **OK**.

## Fonts and Colors

You can define font and color schemes for the various interface items in the workbench. Scheme changes take effect after restarting the workbench.

- Show settings for, lists all interface elements having items with modifiable fonts and colors schemes. You can customize the color settings for an item selected from the Display items list. Clicking Use Defaults resets the font and color settings for the selected item.
- Font (bold type indicates fixed-width fonts), lists all installed fonts. The current font for a selected interface element is displayed in the Font field. You can change the font size using the Size drop-down combo-box.
- Display items, lists items belonging to a selected interface element having modifiable fonts and color schemes. The Item foreground and Item background drop-down combo-boxes automatically display the current color settings for the selected item. You can modify the color setting for the selected item using the drop-down combo-boxes or by clicking Custom. You can apply bold font to the selected item by clicking the Bold checkbox.

**To define custom fonts and colors for interface items**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Environment**, then click **Fonts and Colors**.
3. In the Options dialog box, define the font and color settings for the required interface items, then click **OK**.

## Import and Export Settings

You can define options for saving settings files. You can choose to save your settings to a .vssetting file located on your system or to a shared settings files. When saving settings to a shared .vssettings file, you must provide a UNC path or local path to the shared file.

- Automatically save my settings to this file, displays the name and path to the .vssettings file currently in use. You can change the setting file used by typing a different path or browsing to locate the required settings file on your system.

- Use team settings file, enables navigating to a shared .vssettings file. You can browse to locate the required settings file. This vssettings file is automatically re-applied to the workbench following each modification.

**To define the options for saving the settings file**

1. From the Tools menu, click **Options**.

2. In the Options dialog box, expand **Environment**, then click **Import and Export Settings**.

3. In the Options dialog box, define the required name and location of the settings file, then click **OK**.

## International Settings

When more than one language version of the workbench is installed on a computer, you can change the default language setting for the workbench. Changes to the default language take effect after restarting the workbench.

**To change the default language setting**

1. From the **Tools** menu, click **Options**.
2. In the **Options** dialog box, expand **Environment**, then click **International Settings**.
3. In the **Options** dialog box, select the required language from the **Language** drop-down combo-box, then click **OK**.

The required language is displayed after restarting the workbench.

## Shortcut Keyboard Combinations

The keyboard options enable you to perform many tasks regarding the keyboard shortcuts for the various commands available in the ISaGRAF environment. You can perform the following tasks:

- Viewing defined keyboard shortcuts
- Defining keyboard shortcuts
- Removing defined keyboard shortcuts

Keyboard shortcuts enable quicker operation of the ISaGRAF environment. The keyboard options enable viewing the defined keyboard shortcuts mapping schemes available for commands. In ISaGRAF, only the default keyboard shortcut mapping scheme is available. You view commands in the *Show commands containing* section listing all available commands and their respective keyboard shortcuts. In the text field, you can also type text to find a specific command. By default, only some commands have pre-defined shortcuts. Users can define (add) a shortcut to a command or modify an existing shortcut by adding a new shortcut and removing an unwanted shortcut. You manage keyboard combinations from the following options:.

- *Apply the following additional keyboard mapping scheme*, only the default mapping scheme is available for ISaGRAF.
- *Show commands containing*, displays all commands available in the ISaGRAF environment. When typing characters into the textbox, the list displays all entries containing the specified characters.
- *Shortcuts for selected command*, lists mapped keyboard shortcuts for the command selected in the *Show commands containing* list.
- *Use new shortcut in*, specifies the scope of the keyboard shortcut. You can use the shortcut globally in the ISaGRAF environment or only within a specific context (or window). The default setting is global, meaning the shortcut key works in any active window. If a global keyboard shortcut and context specific shortcut are identical, the context specific shortcut takes precedence. For example, commands having the MLGE editor scope have precedence over commands having the global scope. A context specific keyboard shortcut remains in effect only while the context (or window) is active.

- *Press shortcut keys*, enables pressing a key combination to be used for the currently selected command. You must use one or more modifier keys such as CTRL, ALT, or SHIFT combined with various keys. SHIFT cannot be combined with letters or numbers. The F1-F12 keys can be used with or without a modifier. You can enter one or two key combinations to use as a shortcut. For example, you can enter CTRL+Y, or enter F6, CTRL+Y. Regardless of their scope, shortcut key combinations cannot contain the following keys:

| | |
|---|---|
| PRT SCR/SYS RQ | Application key |
| SCRLK | NUM LOCK |
| CAPS LOCK | CTRL+ALT+DELETE key combination |
| ESCAPE | |

- *Shortcut currently used by*, displays the command assigned to the current keyboard shortcut combination. The textbox is only activated when you assign a key combination that is already assigned to another command. To replace the current shortcut keyboard combination with a custom one you must define a new keyboard shortcut mapping scheme.

**To view existing commands and keyboard shortcuts**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Environment**, then click **Keyboard**.

   The keyboard options are displayed in the Options dialog box.
3. In the *Show commands containing* field, scroll to find a command or type the required command name without spaces. For example, ShowNextStatement.
4. In the *Show commands containing* list, select the required command. For example, Debug.ShowNextStatement.

The drop-down combo-box displays the shortcut key combinations for the selected shortcut.

**To define keyboard shortcuts**

Clicking Assign permanently saves changes for a selected command.

1. From the keyboard options, in the *Show commands containing* field, type the required command name without spaces. For example, ShowNextStatement.

2. In the *Show commands containing* list, select the required command. For example, Debug.ShowNextStatement.

3. In the *Use new shortcut in drop-down* combo-box, select the scope. For example, MLGE.

4. In the *Press shortcut keys* field, type the new key combination.

5. Click **Assign**, then click **OK**.

The shortcut key combination is saved for the required command.

**To remove keyboard shortcuts**

1. From the keyboard options, in the *Show commands containing* field, type the required command name without spaces. For example, ShowNextStatement.

2. In the *Show commands containing* list, select the required command. For example, Debug.ShowNextStatement.

3. In the *Shortcuts for selected command field*, select the keyboard shortcut to be removed.

4. Click **Remove**.

The keyboard shortcut is no longer assigned to the command.

## Startup

The startup options enable you to specify the Workbench behavior when launching **ISaGRAF**:

- Open Home Page, where the Workbench automatically displays the ISaGRAF home page
- Load last loaded solution, where the Workbench opens the last opened project
- Show Open Project dialog box, where the Workbench automatically displays the Open Project dialog box
- Show New Project dialog box, where the Workbench automatically displays the New Project dialog box
- Show empty environment, where the Workbench opens without displaying any project or dialog box

## Specifying Project Options

You can specify the default locations and behavior of project components. You can set default paths for projects and templates. For the Output window, and Solution Explorer, you can set the default behavior during project creation and building. You can also set the options for building and running projects.

- Project location, User project template location, and User item template location, enable defining the default path to project folders used in workbench dialog boxes. The Project location path is used in the Open Project dialog box to define the My Projects location. The User project template location is used in the New Project dialog box to define the My templates list. The User item template location is used in the Add New Item dialog box to define the My Templates list. When defining these default paths, you can type directly in the field or browse for the required location.

- Always show Error List if build finishes with errors, enables opening the Error list window when errors occur during a build operation. When the build operation is complete, the Error List is displayed containing the errors generated by the build operation.

- Track Active Items in Solution Explorer, enables the Solution Explorer to scroll to the node containing the active item, open the folder containing the active item, and select the name of the active item

- Show advanced build configurations, *not implemented*

- Always show solution, enables displaying the solution element and commands that act on the solution within the Solution Explorer. When this option is cleared, new projects are created as stand-alone projects.

- Save new projects when created, enables defining the location of projects in the New Project dialog box. When this option is cleared, new projects are created as temporary projects.

- Warn user when the project location is not trusted, displays a warning message when opening projects from an untrusted location

- Show Output window when build starts, enables displaying the Output window when starting build operations

- Prompt for symbolic renaming when renaming files, enables displaying a message prompting you to select whether to rename all references in the project or just the selected file

**To specify the default locations and behavior of project components**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Projects**, then click **General**.
3. In the Options dialog box, type the required paths or browse for their locations, select the required options, then click **OK**.

## Build Options

You can specify whether a message is displayed before cleaning operations are executed. After performing cleaning operations, online changes are unavailable.

- Proceed to cleaning without asking, enables the display of message indicating that online updates become unavailable after performing a cleaning operation.

**To enable the display of messages prior to cleaning operations**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Projects**, then click **Build**.
3. In the *Proceed to cleaning without asking* drop-down combo-box, select **False**, then click **OK**.

## Interrupts Options

When adding or moving programs to the interrupts section of the Solution Explorer, you can choose to associate the program with an interrupt instance.

- Prompt for interrupt association, enables the display of a message prompting users to choose whether to associate selected programs with interrupt instances when adding or moving programs.

**To set the option for program interrupts**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Projects**, then click **Interrupts**.
3. In the *Prompt for interrupt association* drop-down combo-box, select **True**, then click **OK**.

## Online Settings

When monitoring applications, you can choose to display messages prompting you to confirm the locking or unlocking of variables. You can also specify the number system and number of significant digits used for displaying of numerical values of the different data types categories.

- Prompt for Lock or Unlock, enables the display of messages prompting users to confirm the locking or unlocking of selected variables.
- Bool display format, indication of whether to display boolean values in bool (TRUE/FALSE), bit (1/0), or mixed (TRUE (1)/FALSE (0)) format.
- Integer, indication of whether to display integer values in decimal, hexadecimal, octal, or binary format.
- REAL, indication of whether to display REAL values using scientific notation or a specific number of significant digits after the decimal.
- LREAL, indication of whether to display LREAL values using scientific notation or a specific number of significant digits after the decimal.

**To enable the display of message prompts when locking and unlocking variables**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Projects**, then click **Online**.
3. Select the *Prompt for Lock or Unlock* option.

**To specify the options for displaying numerical values**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Projects**, then click **Online**.
3. In the *Numerical Display* section, set the required values for the different data type categories, then click **OK**.

# Specifying Source Control Settings

**ISaGRAF** includes software developed by * CollabNet (http://www.Collab.Net/) based on the Subversion AnkhSVN source control plug-in for Visual Studio.

**Note:** Source control settings are only available for use with the ISaGRAF 5 CAM.

For source control usage, you can specify options for the following aspects:

- Plug-in Selection
- Subversion Environment
- Subversion User Tools

## Plug-in Selection

For source control, you choose the plug-in to use with the **Automation Collaborative Platform**. The following options are available:

- None, source control is deactivated. The Subversion Environment and Subversion User Tools options for source control are unavailable.
- AnkhSVN - Subversion Support for Visual Studio, source control is activated. The Subversion Environment and Subversion User Tools options for source control are available for use.

**To specify the source control plug-in**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Source Control**, then click **Plug-in Selection**.
3. Specify the required options and click **OK**.

## Subversion Environment

The Subversion environment enables specifying the use of the following options:

- Directly add new files to subversion, indicates whether files are added to subversion without displaying messages prompting users to confirm this action
- Automatically lock files on change without user confirmation, indicates whether files are automatically locked when making a change without displaying messages prompting users to confirm this action

**Note:** To prevent mistakenly stealing locks from other users, avoid automatically locking files on change without user confirmation.

- Flash title bar when a lengthy operation completes, indicates whether to inform users of completion by flashing the title bar of a lengthy operation
- When double clicking items in the Pending Changes, enables selecting the environment in which to view the contents of the file

You can also specify Subversion user settings:

- Proxy settings, enabling and identifying the server and port details for the proxy
- Authentication cache, enabling the storage of the repository logon details and other authentication settings
- Enable client-side hooks, enabling the use of client-side hook scripts

**To specify source control environment settings**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Source Control**, then click **Subversion Environment**.
3. Specify the required options and click **OK**.

**See also**

Subversion User Tools

## Subversion User Tools

For source control, you can specify the following tools for use in various operations:

- External Diff Tool, specifies the comparison utility to use when performing comparison operations for files
- External Merge Tool, specifies the merging utility to use when performing merging operations for files

**To specify source control user tools**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Source Control**, then click **Subversion User Tools**.
3. Specify the tools to use for the required options and click **OK**.

### See also

Specifying Source Control Settings

## Specifying Block Library Settings

You can specify the display mode for the Block Library on startup. The block library can be displayed using expanders or tabs.

**To adjust the display mode for the block library**

1. From the Tools menu, click **Options**.

2. In the Options dialog box, expand **Block Library Settings**, then click **General**.

3. In the Options dialog box, select the required display mode from the drop-down menu, then click **OK**.

# Specifying CAM3 Settings

For CAM3 variables, you can specify the display format for the modbus address. The modbus address can be displayed in hexadecimal or decimal format.

**To adjust the modbus address format for variables**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **CAM3 Settings**, then click **General**.
3. In the Options dialog box, select the required display type from the drop-down menu, then click **OK**.

## Specifying Deployment View Settings

You can adjust colors for various aspects of the deployment view. You can also adjust layout aspects such as the number of devices displayed per row and the horizontal offset (in pixels) between rows of devices.

**To adjust settings for aspects of the deployment view**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Deployment View Settings**, then click **General**.
3. In the Options dialog box, adjust the color or layout settings for the view, then click **OK**.

# Specifying Device View Options

You can specify whether to display the navigation window when opening the device view.

**To specify displaying the navigation window when opening the device view**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Device View**, then click **General**.
3. the option.
4. In the Options dialog box, select *Display the navigation window when opening the device view,* then click **OK**.

## Specifying Documentation Generator Options

You can specify the default Sections Template for the generated documentation. The selected Sections Template modifies the items listed in the Sections pane of the Documentation Generator.

**To specify the default Sections Template**

1. From the Tools menu, click **Options...**

2. In the Options dialog box, expand **Document Generator**, then click **General**.

3. Select the required default Sections Template, then click **OK**.

## Word Settings

You can specify the default Microsoft Word® 2010 (or more recent) settings for the generated documentation. You can specify the following default settings for the generated documentation: orientation, page size, margins, Microsoft Word® template, diagram scaling, link type, and comment style.

**To specify the default Word settings**

1. From the Tools menu, click **Options...**
2. In the Options dialog box, expand **Document Generator**, then click **Word**.
3. Specify the required settings, then click **OK**.

## Setting Grid Options

You can customize the colors displayed in the various workbench grids. You can access the grid options for the following grids:

- Arrays View
- Defined Words View
- Dictionary View
- Parameters Grid
- Structures View
- Variable Groups View
- Variable Selector

**To access the grid options**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Grid Settings**, then click the required grid type.

The grid options for the selected grid type are displayed in the Options dialog box.

## Arrays View

You can customize the colors displayed in the Arrays grid including column headers and rows. The Arrays grid automatically alternates colored rows with white rows. You can adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, you can define the colors displayed. You can also define the color used to indicate disabled rows. You can choose whether to display the filter bar in the Arrays grid.

**To customize the colors displayed in the Arrays grid**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **Grid Settings**, then click **Arrays**.
3. Customize the required options, then click **OK**.
    - To specify the number of consecutive rows for the alternating sequence, for **Consecutive Rows**, indicate the required value.
    - To change the colors applied to headers, alternate rows, and disabled rows, for the respective option, then select a color from the drop-down combo-box.

## Defined Words View

You can customize the colors displayed in the Defined Words grid including column headers and rows. The Defined Words grid automatically alternates colored rows with white rows. You can adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, you can define the colors displayed. You can also define the color used to indicate disabled rows. You can choose whether to display the filter bar in the Defined Words grid.

**To customize the colors displayed in the Defined Words grid**

1. From the Tools menu, click **Options**.

2. From the Options dialog box, expand **Grid Settings**, then click **Defined Words**.

3. Customize the required options, then click **OK**.

   - To specify the number of consecutive rows for the alternating sequence, for **Consecutive Rows**, indicate the required value.

   - To change the colors applied to headers, alternate rows, and disabled rows, for the respective option, then select a color from the drop-down combo-box.

## Dictionary View

You can customize the colors displayed in Dictionary instances including column headers and rows. The Dictionary grid automatically alternates colored rows with white rows. You can adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, you can define the colors displayed. You can also define the color used to indicate disabled rows. You can choose whether to display the filter bar in the Dictionary.

**To customize the colors displayed in the Dictionary**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **Grid Settings**, then click **Dictionary**.
3. Customize the required options, then click **OK**.
   - To specify the number of consecutive rows for the alternating sequence, for **Consecutive Rows**, indicate the required value.
   - To change the colors applied to headers, alternate rows, and disabled rows, for the respective option, then select a color from the drop-down combo-box.

## Parameters Grid

You can customize the colors displayed in Parameters grid including column headers and rows. The Parameters grid automatically alternates colored rows with white rows. You can adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, you can define the colors displayed. You can also define the color used to indicate disabled rows. You can choose whether to display the filter bar in the Parameters grid.

**To customize the colors displayed in the Parameters grid**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **Grid Settings**, then click **Parameters**.
3. Customize the required options, then click **OK**.
    - To specify the number of consecutive rows for the alternating sequence, for **Consecutive Rows**, indicate the required value.
    - To change the colors applied to headers, alternate rows, and disabled rows, for the respective option, then select a color from the drop-down combo-box.

## Structures View

You can customize the colors displayed in Structures grid including column headers and rows. The Structures grid automatically alternates colored rows with white rows. You can adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, you can define the colors displayed. You can also define the color used to indicate disabled rows. You can choose whether to display the filter bar in the Structures grid.

**To customize the colors displayed in the Structures grid**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **Grid Settings**, then click **Structures**.
3. Customize the required options, then click **OK**.
   - To specify the number of consecutive rows for the alternating sequence, for **Consecutive Rows**, indicate the required value.
   - To change the colors applied to headers, alternate rows, and disabled rows, for the respective option, then select a color from the drop-down combo-box.

## Variable Groups View

You can customize the colors displayed in the Variable Groups view including column headers and rows. The Variable Groups view automatically alternates colored rows with white rows. You can adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, you can define the colors displayed. You can also define the color used to indicate disabled rows. You can choose whether to display the filter bar in the Variable Groups view.

**To customize the colors displayed in the Variable Groups view**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **Grid Settings**, then click **Variable Groups**.
3. Customize the required options, then click **OK**.
   - To specify the number of consecutive rows for the alternating sequence, for **Consecutive Rows**, indicate the required value.
   - To change the colors applied to headers, alternate rows, and disabled rows, for the respective option, then select a color from the drop-down combo-box.

## Variable Selector

You can customize the colors displayed in Variable Selector including column headers and rows. The Variable Selector automatically alternates colored rows with white rows. You can adjust the number of consecutive rows used for the alternating sequence. The default row coloring scheme is one colored row followed by one white row. For colored rows, you can define the colors displayed. You can also define the color used to indicate disabled rows. You can choose whether to display the filter bar in the Variable Selector. You can also specify whether the Variable Selector opens displaying the local or global variables tab.

**To customize the colors displayed in the Variable Selector**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **Grid Settings**, then click **Variable Selector**.
3. Customize the required options, then click **OK**.
   - To specify the number of consecutive rows for the alternating sequence, for **Consecutive Rows**, indicate the required value.
   - To change the colors applied to headers, alternate rows, and disabled rows, for the respective option, then select a color from the drop-down combo-box.

# Defining CAM 3 I/O Device Settings

For CAM 3, you can specify settings for I/O devices.

- Always keep devices expanded, specifies whether devices are expanded to display information such as the slot order, number of channels, data type, and description
- Show empty device slots, specifies whether empty device slots are displayed when viewing I/O wiring
- Show full device names, specifies whether I/O devices are displayed with their full names beside the slot number
- Prompt on device removal, specifies whether to prompt users before removing devices
- Prompt when freeing wired variables, specifies whether to prompt users before freeing wired variables

**To define settings for CAM 3 I/O devices**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **I/O Device Settings CAM 3**, then click **General**.
3. In the Options dialog box, define the required settings, then click **OK**.

## Defining CAM 5 I/O Device Settings

For CAM 5 I/O devices, you can specify whether to display alias names for I/O devices. The device alias is defined when creating or editing an I/O device.

**To specify displaying alias names for CAM 5 I/O devices**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **I/O Device Settings CAM 5**, then click **General**.
3. In the Options dialog box, select *Show alias names for I/O devices,* then click **OK**.

## Setting IEC Language Options

You can customize the display settings for programs built in different IEC languages:

- Function Block Diagram
- IEC 61499
- Ladder Diagram
- SAMA
- Sequential Function Chart
- Structured Text

## Function Block Diagram

You can customize the displayed settings for FBD diagrams. You can choose to display grids and instance names. You can choose the comment position for variables and literals. You can define the colors used when displaying FBD elements and text as well as define which variable information is displayed in FBD diagrams. You can choose the width for FBD elements in the language container. You can also choose whether to display grid lines inside FBD language containers. For links, you can choose to display as arrows, present solid, dashed, dotted, dashed-dotted, dashed-dotted-dotted, or custom line styles, and apply a normal, rounded, or rounded with jump line types.

The following options are available for customization:

**Block Style**

| | |
|---|---|
| Background Color | The function and function block background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color | The function and function block background gradient color. The possible colors are custom, web, and system colors. |
| Cell Width | The width for a function or function block, in number of grid cells. |
| Display Instance Names | The indication of whether to display instance names for function blocks. |
| In design mode, go to definition on double click | While in design mode, enables going to the definition on double click. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |

**Comment Style**

| | |
|---|---|
| Background Color | The comment background color. The possible colors are custom, web, and system colors. |

**Constant Style**

| | |
|---|---|
| Background Color - Events | The constant background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color - Events | The constant background gradient color. The possible colors are custom, web, and system colors. |

| | |
|---|---|
| Cell Width | The width for a constant, in number of grid cells. |
| Comment Position | The position of the comment in reference to the constant shape. The possible positions are top, bottom, left, and right. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Variable Information | The information to display for variables. The possible values are name, alias, name and alias, or name and wiring. |
| **Container Settings** | |
| Auto Resize Elements when Modifying | When modifying, automatically resize blocks and variables to accomodate length of text. |
| Automatically Invoke Variable/Block Selector | Controls whether the Variable or Block Selector is automatically displayed when inserting a variable or block in the language container. |
| Display Grid | The indication of whether to display the grid in the language container. |
| **Jump** | |
| Cell Width | The width for a jump, in number of grid cells. |
| **Label** | |
| Cell Width | The width for a label, in number of grid cells. |
| **Left Power Rail** | |
| Background Color | The left power rail background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color | The left power rail background gradient color. The possible colors are custom, web, and system colors. |
| **Link Style** | |
| Is Arrow | The indication of whether to display an arrow at the end of the link. |
| Line Style | The style of the line. The possible values are solid, dash, dot, dash-dot, dash-dot-dot, and custom. |

| | |
|---|---|
| Line Type | The type of line. The normal line type has squared corners and overlapping link intersections. The rounded line type has rounded corners and overlapping link intersections. The rounded with jump line type has rounded corners and link intersections are jumped over. |
| Link Color | The color of links. The possible colors are custom, web, and system colors. |
| **Operator Style** | |
| Background Color | The operator background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color | The operator background gradient color. The possible colors are custom, web, and system colors. |
| Cell Width | The width for an operator, in number of grid cells. |
| Display Instance Names | The indication of whether to display instance names for operators. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| **Region Style** | |
| Background Color | The region background color. The possible colors are custom, web, and system colors. |
| Header Color | The header color of a region. The possible colors are custom, web, and system colors. |
| Header Transparency | The level of transparency of the header section of a region. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| **Right Power Rail** | |
| Background Color | The right power rail background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color | The right power rail background gradient color. The possible colors are custom, web, and system colors. |

**Variable Style**

| | |
|---|---|
| Background Color | The variable background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color | The variable background gradient color. The possible colors are custom, web, and system colors. |
| Cell Width | The width for a variable, in number of grid cells. |
| Comment Position | The position of the comment in reference to the variable shape. The possible positions are none, top, bottom, left, and right. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Variable Information | The information displayed for variables. The possible values are name, alias, name and alias, or name and wiring. |

**To customize the display settings for FBD Diagrams**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **IEC Languages**, then click **Function Block Diagram (FBD)**.
3. Using the available options, customize the required settings, then click **OK**.

## IEC 61499

You can customize the displayed settings for IEC 61499 diagrams. You can choose to display grids and block instance names. You can define background and gradient colors for IEC 61499 elements. You can choose to display names and aliases for literals, variables, and event variables. For links, you can choose to display as arrows, present solid, dashed, dotted, dashed-dotted, dashed-dotted-dotted, or custom line styles, and apply a normal, rounded, or rounded with jump line types.

The following options are available for customization:

**Block Style**

| | |
|---|---|
| Background Color | The function and function block background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color | The function and function block background gradient color. The possible colors are custom, web, and system colors. |
| Cell Width | The width for a function or function block, in number of grid cells. |
| Display Instance Names | The indication of whether to display instance names for function blocks. |
| In design mode, go to definition on double click | While in design mode, enables going to the definition on double click. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |

**Comment Style**

| | |
|---|---|
| Background Color | The comment background color. The possible colors are custom, web, and system colors. |

**Constant Style**

| | |
|---|---|
| Background Color - Events | The constant background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color - Events | The constant background gradient color. The possible colors are custom, web, and system colors. |
| Cell Width | The width for a constant, in number of grid cells. |

| | |
|---|---|
| Comment Position | The position of the comment in reference to the constant shape. The possible positions are top, bottom, left, and right. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Variable Information | The information to display for variables. The possible values are name, alias, name and alias, or name and wiring. |
| **Container Settings** | |
| Auto Resize Elements when Modifying | When modifying, automatically resize blocks and variables to accommodate length of text. |
| Automatically invoke Variable/Block Selector | Controls whether the Variable or Block Selector is automatically displayed when inserting a variable or block in the language container. |
| Display Grid | The indication of whether to display the grid in the language container. |
| **Event Link Style** | |
| Is Arrow | The indication of whether to display an arrow at the end of the link. |
| Line Style | The style of the line. The possible values are solid, dash, dot, dash-dot, dash-dot-dot, and custom. |
| Line Type | The type of line. The normal line type has squared corners and overlapping link intersections. The rounded line type has rounded corners and overlapping link intersections. The rounded with jump line type has rounded corners and link intersections are jumped over. |
| Link Event Color | The color of event links. The possible colors are custom, web, and system colors. |
| **Event Variable Style** | |
| Background Color - Events | The event variable background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color - Events | The event variable background gradient color. The possible colors are custom, web, and system colors. |
| Cell Width | The width for an event variable, in number of grid cells. |

| | |
|---|---|
| Comment Position | The position of the comment in reference to the event variable shape. The possible positions are top, bottom, left, and right. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Variable Information | The information displayed for variables. The possible values are name, alias, name and alias, or name and wiring. |
| **Link Style** | |
| Is Arrow | The indication of whether to display an arrow at the end of the link. |
| Line Style | The style of the line. The possible values are solid, dash, dot, dash-dot, dash-dot-dot, and custom. |
| Line Type | The type of line. The normal line type has squared corners and overlapping link intersections. The rounded line type has rounded corners and overlapping link intersections. The rounded with jump line type has rounded corners and link intersections are jumped over. |
| Link Color | The color of links. The possible colors are custom, web, and system colors. |
| **Region Style** | |
| Background Color | The region background color. The possible colors are custom, web, and system colors. |
| Header Color | The header color of a region. The possible colors are custom, web, and system colors. |
| Header Transparency | The level of transparency of the header section of a region. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| **Variable Style** | |
| Background Color | The variable background color. The possible colors are custom, web, and system colors. |
| Background Gradient Color | The variable background gradient color. The possible colors are custom, web, and system colors. |

| | |
|---|---|
| Cell Width | The width for a variable, in number of grid cells. |
| Comment Position | The position of the comment in reference to the variable shape. The possible positions are none, top, bottom, left, and right. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Variable Information | The information displayed for variables. The possible values are name, alias, name and alias, or name and wiring. |

**To customize the display settings for IEC 61499 Diagrams**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **IEC Languages**, then click **IEC 61499**.
3. Using the available options, customize the required settings, then click **OK**.

## Ladder Diagram

You can customize the displayed settings for LD diagrams. You can choose to display grids and instance names. You can define the colors used when displaying LD elements and text as well as define which variable information is displayed in LD diagrams. You can choose the width and height for LD elements in the language container.

The following options are available for customization:

**Block Settings**

| | |
|---|---|
| Display Image | The indication of whether to display block images. |
| Display Instance Names | The indication of whether to display instance names for function blocks. |
| Enable EN/ENO | Forces EN and ENO parameters onto all operators, functions, and function blocks. |
| Function Blocks Background Color | The function block background color. The possible colors are custom, web, and system colors. |
| Function Blocks Background Gradient Color | The function block background gradient color. The possible colors are custom, web, and system colors. |
| Functions Background Color | The function background color. The possible colors are custom, web, and system colors. |
| Functions Background Gradient Color | The function background gradient color. The possible colors are custom, web, and system colors. |
| Go to Definition on Double-click | While in design mode, enables going to the definition on double click. |
| Operators Background Color | The operator background color. The possible colors are custom, web, and system colors. |
| Operators Background Gradient Color | The operator background gradient color. The possible colors are custom, web, and system colors. |

**Container Settings**

| | |
|---|---|
| Cell Height | The height of individual cells making up the grid, in pixels. |
| Cell Width | The width of individual cells making up the grid, in pixels. |
| Display Grid | The indication of whether to display the grid. |

| | |
|---|---|
| Element Height | The height of elements, in grid cells. Basic elements are blocks without inputs or outputs, coils, and contacts. For blocks, each input and output adds a basic element dimension. |
| Element Width | The width of elements, in grid cells. Basic elements are blocks without inputs or outputs, coils, and contacts. For blocks, each input and output adds a basic element dimension. |
| Font | The type of font. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDICharSet and GDIVerticalFont properties are not editable. |
| Rung Line Thickness | The thickness of the rung line. The possible values range from 1.0 to 3.0. |
| **Editor Settings** | |
| Automatically Invoke Variable/Block Selector | Controls whether the Variable or Block Selector is automatically displayed when inserting a variable or block in the language container. |
| **Rung Settings** | |
| Coil Alignment | Indicates whether to align all coils on the rightmost section of the rung. |
| Comment Background Color | The comment background color. The possible colors are custom, web, and system colors. |
| Comment text color | The text color for comments. The possible colors are custom, web, and system colors. |
| Display Comment | The indication of whether to display comments for rungs. |
| Display Label | The indication of whether to display labels for rungs. When not displaying labels, an arrow appears in the leftmost section of the rung indicating the existence of a label. |
| Label Color | The color for rung labels. The possible colors are custom, web, and system colors. |
| Power Flow False Color | The color displayed when power flow monitoring is false. The possible colors are custom, web, and system colors. |
| Power Flow True Color | The color displayed when power flow monitoring is true.The possible colors are custom, web, and system colors. |

| | |
|---|---|
| Power Rail Color | The color for power rails. The possible colors are custom, web, and system colors. |
| Rung Header Color | The color for rung headers. The possible colors are custom, web, and system colors. |
| **Variables Settings** | |
| Text Color - Design | The color of text displayed while in design mode. The possible colors are custom, web, and system colors. |
| Text Color - Online | The color of text displayed while running online. The possible colors are custom, web, and system colors. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Variable Background Color | The variable background color. The possible colors are custom, web, and system colors. |
| Variable Background Gradient Color | The variable background gradient color. The possible colors are custom, web, and system colors. |
| Variable Information | The indication of whether to display the variable name only, alias only, name and alias, or name and wiring. |

**To customize the display settings for LD Diagrams**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **IEC Languages**, then click **Ladder Diagram (LD)**.
3. Using the available options, customize the required settings, then click **OK**.

## SAMA

You can customize the displayed settings for SAMA diagrams. You can choose to display grids and instance names. You can choose the width of variable elements as well as the variable comment position. You can define the colors used when displaying SAMA elements and text. You can also choose which variable information is displayed in SAMA diagrams.

The following options are available for customization:

**Block Settings**

| | |
|---|---|
| Display Instance Names | The indication of whether to display instance names for function blocks. |
| Go to Definition on Double-click | While in design mode, enables going to the definition on double click. |

**Container Settings**

| | |
|---|---|
| Display Grid | The indication of whether to display the grid. |
| Force Normal Line Type | The indication of whether all links use the normal line type. |

**Editor Settings**

| | |
|---|---|
| Automatically invoke Variable/Block Selector | Controls whether the Variable or Block Selector is automatically displayed when inserting a variable or block in the language container. |

**Variables Settings**

| | |
|---|---|
| Comment Position | The position of the comment in reference to the variable shape. The possible positions are none, top, bottom, left, and right. |
| Text Color - Design | The color of text displayed while in design mode. The possible colors are custom, web, and system colors. |
| Text Color - Online | The color of text displayed while running online. The possible colors are custom, web, and system colors. |
| Transparency | The level of transparency. The possible values range from 0 to 255 where 0 indicates complete transparency. |
| Variable Background Color | The variable background color. The possible colors are custom, web, and system colors. |

| | |
|---|---|
| Variable Background Gradient Color | The variable background gradient color. The possible colors are custom, web, and system colors. |
| Variable Information | The indication of whether to display the variable name only, alias only, name and alias, or name and wiring. |
| Width | The variable width, in number of grid cells. |

**To customize the display setting for SAMA Diagrams**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **IEC Languages**, then click **SAMA**.
3. Using the available options, customize the required settings, then click **OK**.

## Sequential Function Chart

You can customize the displayed settings for SFC diagrams. You can choose the orientation of the pane splitting when displaying SFC diagram and actions/conditions programming simultaneously in the language container. You can choose to display grids and sequence control types as well as diagram background and grid colors for design and online modes. For action blocks, jumps, and transitions, you can define the background, gradient, and font colors as well as the font style. For steps, you can define the active and inactive step and step gradient colors, the font color and style as well as the action list and list gradient colors.

The following options are available for customization:

**Action Block Settings**

| | |
|---|---|
| Action Block Color | The background color of action blocks. The possible colors are custom, web, and system colors. |
| Action Block Font | The font definition used for the text displayed in an action block. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Action Block Font Color | The color of the font for action blocks. The possible colors are custom, web, and system colors. |
| Action Block Gradient Color | The background gradient color of action blocks. The possible colors are custom, web, and system colors. |

**Container Settings**

| | |
|---|---|
| Background Color - Design | The background color for SFC diagrams while is design mode. The possible colors are custom, web, and system colors. |
| Background Color - Online | The background color for SFC diagrams while online. The possible colors are custom, web, and system colors. |
| Container Split Orientation | Controls the orientation for the splitting of the container between the SFC diagram and Actions/Conditions views. The possible values are vertical or horizontal. |
| Display Grid | The indication of whether to display the grid. |
| Display Sequence Control Type | The indication of whether to display the sequence controls type. |

| | |
|---|---|
| Display Transition Priority | The indication of whether to display the transition priority. |
| Grid Color - Design | The color of the grid while in design mode. The possible colors are custom, web, and system colors. |
| Grid Color - Online | The color of the grid while running online. The possible colors are custom, web, and system colors. |
| **Jump Settings** | |
| Jump Color | The background color of jumps. The possible colors are custom, web, and system colors. |
| Jump Font | The font definition used for the text displayed in a jump. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Jump Font Color | The color of the font for jumps. The possible colors are custom, web, and system colors. |
| Jump Gradient Color | The background gradient color of jumps. The possible colors are custom, web, and system colors. |
| **Macro Call Settings** | |
| Macro Call Color | The background color of macro calls. The possible colors are custom, web, and system colors. |
| Macro Call Font | The font definition used for the text displayed in a macro call. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Macro Call Font Color | The color of the font for macro calls. The possible colors are custom, web, and system colors. |
| Macro Call Gradient Color | The background gradient color of macro calls. The possible colors are custom, web, and system colors. |
| **Step Settings** | |
| Action List Color | The background color of action lists. The possible colors are custom, web, and system colors. |

| | |
|---|---|
| Action List Gradient Color | The background gradient color of action lists. The possible colors are custom, web, and system colors. |
| Step Color | The background color of steps. The possible colors are custom, web, and system colors. |
| Step Color - Active | The background color of active steps while online. The possible colors are custom, web, and system colors. |
| Step Font | The font definition used for the text displayed in a step. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Step Font Color | The color of the font for steps. The possible colors are custom, web, and system colors. |
| Step Gradient Color | The background gradient color of steps. The possible colors are custom, web, and system colors. |
| Step Gradient Color - Active | The background gradient color of active steps while online. The possible colors are custom, web, and system colors. |
| **Transition Settings** | |
| Transition Color | The background color of transitions. The possible colors are custom, web, and system colors. |
| Transition Font | The font definition used for the text displayed in a transition. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Transition Font Color | The color of the font for transitions. The possible colors are custom, web, and system colors. |
| Transition Gradient Color | The background gradient color of transitions. The possible colors are custom, web, and system colors. |

**To customize the display setting for SFC diagrams**

1. From the Tools menu, click **Options**.

2. From the Options dialog box, expand **IEC Languages**, then click **Sequential Function Chart.**

3. Using the available options, customize the required settings, then click **OK**.

## Structured Text

You can define the default display setting for ST elements and text displayed in ST language containers. You can choose the font used when displaying comments, editor text, identifiers, numbers, operators, POUs, punctuation, reserved words, and strings. You can choose to display these in bold, italic, strike-through, or underlined text as well as define their text color and size.

The following options are available for customization:

**Comment**

| | |
|---|---|
| Comment Font | The font definition used for comment text. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Comment Text Color | The color of the font for comments. The possible colors are custom, web, and system colors. |

**Editor**

| | |
|---|---|
| Editor Font | The font definition used for the ST editor. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Editor Text Area Background Color | The color of the ST editor background. The possible colors are custom, web, and system colors. |

**Identifier**

| | |
|---|---|
| Identifier Font | The font definition used for identifiers. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Identifier Text Color | The color of the font for identifiers. The possible colors are custom, web, and system colors. |

**Number**

| | |
|---|---|
| Number Font | The font definition used for numbers. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Number Text Color | The color of the font for numbers. The possible colors are custom, web, and system colors. |
| **Operator** | |
| Operator Font | The font definition used for operators. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Operator Text Color | The color of the font for operators. The possible colors are custom, web, and system colors. |
| **POU** | |
| POU Font | The font definition used for POUs. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| POU Text Color | The color of the font for POUs. The possible colors are custom, web, and system colors. |
| **Punctuation** | |
| Punctuation Font | The font definition used for punctuation. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Punctuation Text Color | The color of the font for punctuation. The possible colors are custom, web, and system colors. |
| **Reserved Word** | |

| | |
|---|---|
| Reserved Word Font | The font definition used for reserved words. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| Reserved Word Text Color | The color of the font for reserved words. The possible colors are custom, web, and system colors. |
| **String** | |
| String Font | The font definition used for strings. The definition includes the font name, size, unit of measure, as well as the indication of whether to apply bold, italic, strikeout, and underline styles. The GDI Character Set and GDI Vertical Font properties are not editable. |
| String Text Color | The color of the font for strings. The possible colors are custom, web, and system colors. |

**To customize the display setting for ST programs**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **IEC Languages**, then click **Structured Text (ST)**.
3. Expand the respective category, customize the required settings, then click **OK**.

# Setting ISaVIEW Options

You can customize the default settings and behavior of various facets of ISaVIEW screens and objects:

- ISaVIEW Animation Settings
- ISaVIEW Objects Settings
- ISaVIEW Edition Settings

## ISaVIEW Animation Settings

The animation settings enable customizing the animation settings for ISaVIEW screens including action, displacement, rotation, and size. You can also define the refresh rate of ISaVIEW screens as well as their default background color.

**To customize the animation settings for ISaVIEW screens**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then click **ISaVIEW Animation Settings**.
3. Customize the required settings, then click **OK**.

## ISaVIEW Edition Settings

The edition settings enable defining the default settings for ISaVIEW screens and generic object properties.

**To define the default edition settings for ISaVIEW screens and objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then click **ISaVIEW Edition Settings**.
3. Define default display settings, then click **OK**.

## ISaVIEW Objects Settings

The object settings enable specifying default values for the individual object properties and grouping properties.

- Arc Settings
- Arrow Settings
- Bar Meter Settings
- Button Settings
- Edit Box Settings
- Ellipse Settings
- Gauge Settings
- Group Settings
- Image Settings
- Line Settings
- Polygon Settings
- Rectangle Settings
- Rounded Rectangle Settings
- Slider Settings
- Triangle Settings
- Web Container Settings

## Arc Settings

The arc settings enable specifying default values for the individual object properties.

**To specify the default settings for arc objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Arc Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Arrow Settings

The arrow settings enable specifying default values for the individual object properties.

**To specify the default settings for arrow objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Arrow Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Bar Meter Settings

The bar meter settings enable specifying default values for the individual object properties.

**To specify the default settings for bar meter objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Bar Meter Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Button Settings

The button settings enable specifying default values for the individual object properties.

**To specify the default settings for button objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Button Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Edit Box Settings

The edit box settings enable specifying default values for the individual object properties.

**To specify the default settings for edit box objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Edit Box Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Ellipse Settings

The ellipse settings enable specifying default values for the individual object properties.

**To specify the default settings for ellipse objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Ellipse Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also
ISaVIEW Objects Settings

## Gauge Settings

The gauge settings enable specifying default values for the individual object properties.

**To specify the default settings for gauge objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Gauge Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also
ISaVIEW Objects Settings

## Group Settings

The group settings enable specifying default values for the individual object properties.

**To specify the default settings for grouped objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Group Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Image Settings

The image settings enable specifying default values for the individual object properties.

**To specify the default settings for image objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Image Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Line Settings

The line settings enable specifying default values for the individual object properties.

**To specify the default settings for line objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Line Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Polygon Settings

The polygon settings enable specifying default values for the individual object properties.

**To specify the default settings for polygon objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Polygon Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Rectangle Settings

The rectangle settings enable specifying default values for the individual object properties.

**To specify the default settings for rectangle objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Rectangle Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Rounded Rectangle Settings

The rounded rectangle settings enable specifying default values for the individual object properties.

**To specify the default settings for rounded rectangle objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Rounded Rectangle Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Slider Settings

The slider settings enable specifying default values for the individual object properties.

**To specify the default settings for slider objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Slider Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Triangle Settings

The triangle settings enable specifying default values for the individual object properties.

**To specify the default settings for triangle objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Triangle Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

## Web Container Settings

The web container settings enable specifying default values for the individual object properties.

**To specify the default settings for web container objects**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **ISaVIEW Settings**, then expand **ISaVIEW Object Settings**, and then click **Web Container Settings**.
3. Specify the default values for the required properties, then click **OK**.

### See also

ISaVIEW Objects Settings

# Defining Spy List Settings

You can customize the offline and online behavior options and look and feel of spy lists. The available behavior options are the following:

- Offline Grid Settings
- Online Grid Settings

## Offline Grid Settings

You can customize the offline behavior options and look and feel of spy lists. The available behavior options are the following:

- Filter row, displaying a row below the column heading enabling the filtering of items in the list
- Grouping drop area, displaying an area at the top of spy lists enabling the grouping of items in a list according to column types
- Indent sub-items, indenting sub-items of arrays, structures, and function blocks
- Item count rows, displaying rows indicating the item count for complete spy lists as well as individual arrays, structures, and function block instances

The available look and feel options enable customizing the colors used for the headers, various rows, and borders as well the text colors.

**To customize spy lists for offline usage**

You can define different settings for the offline and online options.

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **Spy List Settings**, then click **Offline Grid Settings** and make the required changes.

## Online Grid Settings

You can customize the online behavior options and look and feel of spy lists. The available behavior options are the following:

- Filter row, displaying a row below the column heading enabling the filtering of items in the list
- Grouping drop area, displaying an area at the top of spy lists enabling the grouping of items in a list according to column types
- Indent sub-items, indenting sub-items of arrays, structures, and function blocks
- Item count rows, displaying rows indicating the item count for complete spy lists as well as individual arrays, structures, and function block instances

The available look and feel options enable customizing the colors used for the headers, various rows, and borders as well the text colors.

**To customize spy lists**

You can define different settings for the offline and online options.

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **Spy List Settings**, then click **Online Grid Settings** and make the required changes.

# Description Window

The Description window enables adding descriptions to projects, devices, resources (if supported by the CAM), and POUs. These descriptions are free-formatted text using rich text format (RTF). When adding a description, all content is automatically saved. When editing descriptions, a text editor toolbar provides the means for performing basic formatting operations such as selecting a font, size, style, and color.

The Description window is dockable and scalable. When clicking the different items in the Solution Explorer, the contents of the Description window automatically displays the description for the selected item.

While in debug mode, the content displayed in the Description window is read-only.

**To access the description window**

You can access the description window from the menus or from the properties for items.

- To access the Description Window, from the **View** menu, click **Description Window**.

# ISaGRAF 3 Concrete Automation Model

The **ISaGRAF 3** Concrete Automation Model enables the creation of **ISaGRAF 3** applications supporting multi-process control. Applications consist of virtual machines running on hardware components, called target platforms. The development process consists of creating projects made up of a device that is downloaded to a target platform. At runtime, the device becomes a virtual machine running on the target platform.

Projects can be developed using different programming languages including some from the IEC 61131-3 standard. When building, a device is compiled to produce very fast "target independent code" (TIC) or "C" code.

Within devices, you can declare variables using standard IEC 61131-3 data types (i.e., BOOL, DINT, REAL, MESSAGE, and TIME) or user-defined types such as one-dimensional arrays.

You develop projects on a Windows development platform. The **Automation Collaborative Platform** graphically represents and organizes the device, POUs, and networks within a project from many views.

- Deployment
- Dictionary
- I/O wiring
- Bindings

You can choose to simulate the running of a project, after building a project, using high-level debugging tools, before actually downloading the device to the target platform.

# Creating a Project

You can create projects as part of new or existing solutions in the Automation Collaborative Platform. A solution can hold multiple projects. You can import existing projects created using previous versions of **ISaGRAF 3**.

The **ISaGRAF 3** Project template is available for **ISaGRAF 3** projects. This template enables creating a project without attaching it to a new or existing solution. Empty projects contain no device files.

Since the **ISaGRAF 3** run-time is a 16-bit application, the quantity of POUs, variables, and I/O devices are directly dependent upon that environment.

For projects, the following properties are defined:

**CAM**

| | |
|---|---|
| CAM Project | The device name for the project |
| Documentation | Free-form text describing the project |

**Info**

| | |
|---|---|
| Name | Name of the project. Project names are recommended to have up to 32 characters |
| Path | Complete path where the Automation Collaborative Platform (ACP) project file is stored on the computer. The path is automatically assigned:<br>%USERPROFILE%\My Documents\ISaGRAF 6.x\Projects*SolutionName**ProjectName* |

Projects are stored in the Projects directory, as MS-Access database (.MDB) files:

%USERPROFILE%\My Documents\ISaGRAF 6.x\Projects

**To create a project**

1. From the File menu, point to **New**, then click **Project** (or press **Ctrl+Shift+N**).
2. In the Installed Templates list, expand the *CAM Projects* option, then expand **ISaGRAF 3**, and click **Empty**.

3. From the list of available project templates, click the **ISaGRAF 3 Project** template.

4. Specify a name and location for the project, indicate whether to add the project to an existing solution or create a new solution by defining a solution name, then click **OK**. For new solutions, you can choose to create a directory.

### See Also

Importing an ISaGRAF 3 Project
Creating a Library
Importing an ISaGRAF 3 Library

# Devices

A device corresponds to a programmable logic controller. For devices, you can specify the following properties:

**Application Run-time Options**

| | |
|---|---|
| Cycle Timing (ms) | The amount of time given to each cycle. If a cycle is completed within the cycle timing period, the system waits until this period has elapsed before starting a new cycle. The cycle consists of scanning the physical inputs of the process to drive, executing the POUs of the device, then updating physical outputs. The virtual machine executes the device code according to the execution rules. |
| Memory | The size of the memory space reserved for storing the values of retained variables. The values of these variables are stored in this memory at the end of each cycle for use if the target is restarted. |
| Nb Stored Errors | Number of entries, i.e., the size of the queue (FIFO) in which detected errors are stored |
| Starting Mode | Indication of whether a device executes in real time or cycle-to-cycle. RealTime mode is the run time normal execution mode where target cycles are triggered by the cycle timing. In cycle-to-cycle mode, the virtual machine loads the resource code but does not execute it until you execute one cycle or activate real-time mode. |

**Compiler Options**

| | |
|---|---|
| Build Binary Decision Diagrams (BDDs) | Indication of whether the optimizer replaces Boolean equations (mixing AND, OR, XOR and NOT operators) with a reduced list of conditional jump operations. The translation is performed only when the expected execution time for the jump sequence is less than the one expected for the original expression. |
| Evaluate Constant Expressions | Indication of whether the compiler evaluates constant expressions. For example, the numerical expression "2 + 3" is replaced by "5" in the target code. When this option is not set, constant expressions are calculated at run-time. |
| Generate Debug Information | Indication of whether to generate information required for debugging using step-by-step execution |

| | |
|---|---|
| Optimize Arithmetic Operations | Indication of whether the optimizer simplifies arithmetic operations according to special operands. For example, the expression "A + 0" is replaced with "A". |
| Optimize Booleans | Indication of whether the optimizer simplifies Boolean operations according to special operands. For example, the Boolean expression "A & A" is replaced with "A". |
| Optimize Expressions | Indication of whether the optimizer re-uses the results of expressions and sub-expressions which are used more than once in the program |
| Optimize Variable Copying | Indication of whether to optimize the use of temporary variables (used to store intermediate results). This option is commonly used with the Optimize expressions option. |
| Run Two Optimizer Passes | Indication of whether the code optimizer runs twice. Optimizations performed during the second pass are generally less significant than those performed during the first pass. |
| Suppress Unused Code | Indication of whether the optimizer suppresses insignificant code. For example, if the following statements are programmed: "var := 1; var := X;", the corresponding generated code is: "var := X;". |
| Suppress Unused Labels | Indication of whether the optimizer simplifies the system of jumps and labels defined in programs in order to suppress unused target labels or null jumps. |
| Use Embedded SFC Engine | Indication of whether to use the **ISaGRAF** SFC engine. This mode leads to higher run-time performance. However, the target engine may be missing some particular **ISaGRAF** target implementations such as those more commonly found on customized targets based on **ISaGRAF** code post-processing. In such a case, you may need to remove this option and let the **ISaGRAF** compiler translate SFC charts into low-level instructions. For more information regarding using the embedded SFC engine, refer to your hardware documentation. |

**Serial Port Connection Information**

| | |
|---|---|
| Baud Rate | The data transmission speed, defined in bits per second. Possible values are 0, 600, 1200, 2400, 4800, 9600, and 19200; the default value is 19200. A value of 0 indicates no change. |

| | |
|---|---|
| Data Bits | The number of data bits in a byte. Possible values are 0, 7, or 8; the default value is 8. A value of 0 indicates no change. |
| Hardware Flow Control | Indication of whether the workbench controls the CTS and DSR lines to enable hardware handshaking during exchanges |
| Parity | The parity type. The value of this property is either None or Odd; the default value is None. |
| Stop Bits | Length of the stop bit. Possible values are 1 or 2. |

**Shared Connection Information**

| | |
|---|---|
| Network | The network used for communication. Possible values are Serial and TCP/IP. The default value is TCP/IP. |
| Retry | The number of tries for communication using a serial connection |
| Serial Port or Port Category | The serial port used for the workbench. Possible values are:<br>COM1<br>COM2<br>COM3<br>COM4<br>User (ISDK)<br>IP |
| Slave | Number identifying the target **ISaGRAF** task (Isaker or Wisaker). The value of this property ranges from 1 to 255. Refer to the target supplier manual for the slave number of the target system used. |
| Targets | Name of the target |
| Timeout | Communication timeout, expressed in seconds |

**TCP/IP Connection Information**

| | |
|---|---|
| Host Address | For TCP/IP connections, specifies the IP address of the host. |
| Host Socket Port | The Internet port number for a TCP-IP connection. The Workbench uses the WINSOCK.DLL Version 1.1 library for TCP-IP communications. This file must be correctly installed on the hard disk. If not specified, the default port number is 1100 when running the **ISaGRAF** target. |

### See Also

Creating a Project

Importing an ISaGRAF 3 Project
Creating a Library
Importing an ISaGRAF 3 Library

# Programs

You define programs in the Programs section of a device in the Solution Explorer. Within the program hierarchy, sequential programs must be adjacent where their execution is not interrupted by non-sequential programs; non-sequential programs can be placed before or after but not between sequential programs. Programs belonging to a same section must have different names.

**Miscellaneous**

| | |
|---|---|
| Comment | Text displayed next to the program name in the Solution Explorer |
| Description | Free-form text describing a program |
| Language | Programming language of the POU |
| Name | Name of the program. Program names can have up to eight (8) characters and must begin with a letter followed by letters, digits, and single underscores. |
| Type | Type of POU. Possible values are program, user-defined function, or user-defined function block |

**To add a program**

You define programs for a device.

- In the Solution Explorer, right-click the program element for a device, point to **Add**, then click the required programming language.

**To rename a program**

- In the Solution Explorer, right-click the program, click **Rename**, and then type a name for the program.

**To delete a program**

- In the Solution Explorer, right-click the program, and then click **Delete**.

# Functions

You define functions in the Functions section of a device in the Solution Explorer.

For functions, you can specify the following properties:

| | |
|---|---|
| Comment | Text displayed next to the function name in the Solution Explorer |
| Description | Free-form text describing a function |
| Language | Programming language of the POU |
| Name | Name of the function. Function names can have up to eight (8) characters. Function names must begin with a letter followed by letters, digits, and single underscores. |
| Type | Type of POU. Possible values are program, user-defined function, or user-defined function block |

When adding functions, you also need to define parameters. Functions can have a maximum of 32 parameters (31 inputs and one output). When defining parameters, consider the following limitations:

- Parameter names are limited to 32 characters and must begin with a letter followed by letters, digits, and single underscores
- Possible data types for parameters are BOOL, DINT, REAL, TIME, MESSAGE
- For Message type variables, string capacity is limited to 255 characters
- For user defined addresses, the format is hexadecimal and the value ranges from 1 to FFFF

**To add a function**

1. In the Solution Explorer, right-click the Functions element, point to **Add**, then click the required programming language for the function.
2. To define the parameters for the function, right-click the function, and then click **Parameters**.

   The Block Selector displays the Parameters section where you define the parameters for the function.

**To rename a function**

- In the Solution Explorer, right-click the function, click **Rename**, and then type a name for the function.

**To delete a function**

1. In the Solution Explorer, right-click the function, and then click **Delete**.

# Function Blocks

You define function blocks in the Function Blocks section of a device in the Solution Explorer.

For function blocks, you can specify the following properties:

| | |
|---|---|
| Comment | Text displayed next to the function block name in the Solution Explorer |
| Description | Free-form text describing a function block |
| Language | Programming language of the POU |
| Name | Name of the function block. Function block names can have up to eight (8) characters. Function block names must begin with a letter followed by letters, digits, and single underscores. |
| Type | Type of POU. Possible values are program, user-defined function, or user-defined function block |

When adding function blocks, you also need to define parameters. Function blocks can have a maximum of 32 parameters (inputs and outputs). When defining parameters, consider the following limitations:

- Parameter names are limited to 32 characters and must begin with a letter followed by letters, digits, and single underscores
- Possible data types for parameters are BOOL, DINT, REAL, TIME, MESSAGE
- For Message type variables, string capacity is limited to 255 characters
- For user defined addresses, the format is hexadecimal and the value ranges from 1 to FFFF

**To add a function block**

1. In the Solution Explorer, right-click the Function Blocks element, point to **Add**, and then click the required programming language for the function block.

2. To define the parameters for the function block, right-click the function block, and then click **Parameters**.

   The Block Selector displays the Parameters section where you define the parameters for the function block.

**To rename a function block**

- In the Solution Explorer, right-click the function block, click **Rename**, and then type a name for the function block.

**To delete a function block**

- In the Solution Explorer, right-click the function block, and then click **Delete**.

# Variables

You define variables for their scope. For instance, global variables are available for use throughout the programs, functions, and functions blocks of a device. Whereas, variables defined for a program, a function, or a function block are local to that element. You define variables in the Variables grid. For individual variable scopes, you can import and export variables data having the Microsoft Excel (*.xls) format.

When defining variables data using a spreadsheet you enter each piece of information in a separate cell, leave cells empty if items are to be omitted, and save the file in XLS format. These requirements are automatically followed by the export utility; you must respect these when building a file to be imported.

When defining complex variables such as arrays and structures, the syntax for the variable name is as follows:

- For arrays: arrayname[index]

  ```
  Name,Alias,Data Type,StringSize,InitValue,Direction,Wiring,Attribute ...
  array1,,BOOL,0,,, ...
  "array1[1,1]",,BOOL,0,,, ...
  "array1[1,2]",,BOOL,0,,, ...
  "array1[1,3]",,BOOL,0,,, ...
  "array1[1,4]",,BOOL,0,,, ...
  "array1[1,5]",,BOOL,0,,, ...
  ```

When managing variables data, you can import and export variables data.

# Targets

ISaGRAF 3 projects support compiling up to three different target codes during the same build operation. The following are the standard ISaGRAF 3 targets:

- SIMULATE, used when simulating the application. The compiler generates different code for simulation than online.
- ISA68M, when selected the compiler produces TIC code for use with Motorola-based processors. The processor type concerns byte ordering in the generated code.
- ISA86M, when selected the compiler produces TIC code for use with Intel-based processors. The processor type concerns byte ordering in the generated code.
- CC86M, when selected the compiler produces non-structured "C" source code to be compiled and linked with ISaGRAF target libraries producing an embedded executable code. The CC86M target is compatible with ISaGRAF 3.23 and earlier projects not supporting structured "C" source code.
- SCC, when selected the compiler produces structured "C" source code to be compiled and linked with ISaGRAF target libraries producing an embedded executable code.

# Networks and Connections

ISaGRAF 3 targets support the following networks:

- TCP/IP
- Serial

## TCP/IP

The TCP/IP protocol is the network driver used for communication with ISaGRAF on Ethernet. The following connection properties are available for the TCP/IP network driver:

**Shared Connection Information**

| | |
|---|---|
| Retry | The quantity of retries attempted when a timeout occurs during reading. The default value is 1. |
| Serial Port or Port Category | The **ISaGRAF 3** communication port. For the TCP/IP network, select **Ip**. |
| Slave | The slave number. The default value is 1. |
| Time out (s) | The time delay before a timeout occurs, in milliseconds. The default value is 2. |

**TCP/IP Connection Information**

| | |
|---|---|
| Host Address | The socket host name or IP address. The default value is 127.0.0.1 |
| Host Socket port | The socket port number. The default value is 1100. |

**To specify TCP/IP network connection properties**

**1.** From the View menu, click Deployment View.

The Deployment View is displayed in the workspace.

**2.** Select the network connection, then from the Properties window specify the required connection properties.

## Serial

The network driver used when developing an IXL client using serial communication with **ISaGRAF**. The following connection properties are available for the Serial network driver:

**Shared Connection Information**

| | |
|---|---|
| Retry | The quantity of retries attempted when a timeout occurs during reading. The default value is 1. |
| Serial Port or Port Category | The **ISaGRAF 3** communication port. For the Serial network, select Com1, Com2, Com3, Com4, Com5, Com6, Com7, Com8, Com9, Com10, or Com11. |
| Slave | The slave number. The default value is 1. |
| Time out (s) | The time delay before a timeout occurs, in milliseconds. The default value is 2. |

**Serial Port Connection Information**

| | |
|---|---|
| Baud Rate | The baud data transfer rate. The default value is 0. |
| Data Bits | The quantity of bits used for the smallest unit of data. The default value is 0. |
| Hardware Flow Control | The control of the flow of data transmission between the network hardware. Possible values are True or False. The default value is False. |
| Parity | The type of parity used. Possible values are None, Odd, Even, Mark, or Space. The default value is None. |
| Stop Bits | The number of stop bits used to indicate the end of a transmission. Possible values are None, One, Two, or OnePointFive. The default value is One. |

**To specify Serial network connection properties**

1. From the View menu, click Deployment View.

   The Deployment View is displayed in the workspace.

2. Select the network connection, then from the Properties window specify the required connection properties.

# Importing an ISaGRAF 3 Project

You can import projects from previous versions of **ISaGRAF 3** as part of new or existing solutions in the Automation Collaborative Platform. A solution can hold multiple projects.

The Import **ISaGRAF 3** Project template is available for **ISaGRAF 3** projects. This template enables importing an **ISaGRAF 3** project into the **ISaGRAF 6.x** workbench.

For projects, the following properties are defined:

| | |
|---|---|
| **CAM** | |
| CAM Project | The device name for the project |
| Documentation | Free-formatted text |
| **Info** | |
| Name | Name of the project. Project names are recommended to have up to 32 characters |
| Path | Complete path where the Automation Collaborative Platform (ACP) project file is stored on the computer. The path is automatically assigned: %USERPROFILE%\My Documents\ ISaGRAF 6.x\Projects*SolutionName**ProjectName* |

Projects are stored in the Projects directory, as MS-Access database (.MDB) files:

%USERPROFILE%\My Documents\ISaGRAF 6.x\Projects

**To import an ISaGRAF 3 project**

You can import projects created using **ISaGRAF 3**.

1. From the File menu, point to **New**, then click **Project** (or press **Ctrl+Shift+N**).
2. In the Installed Templates list, expand the *CAM Projects* option, then expand **ISaGRAF 3**, and click **Import**.
3. From the list of available project templates, select the **Import ISaGRAF 3 Project** template.

4. Specify a name and location for the project, indicate whether to add the project to an existing solution or create a new solution by defining a solution name, then click **OK**. For new solutions, you can choose to create a directory.

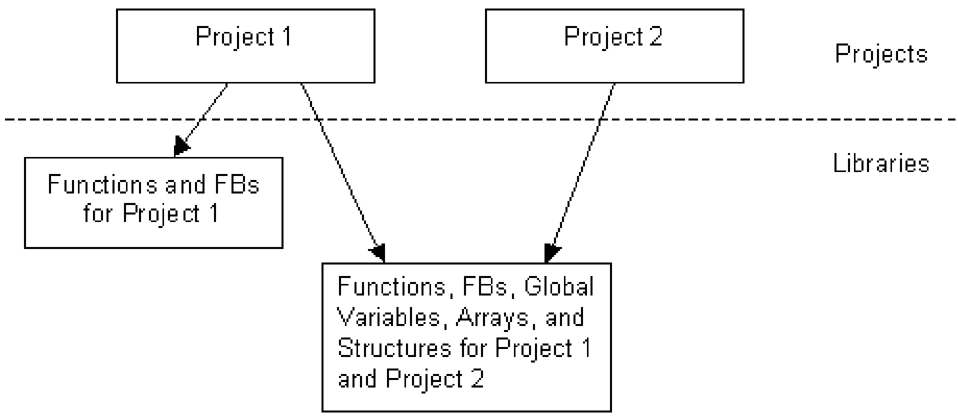
5. In the Selecting an *.hie File dialog box, locate and select the **ISaGRAF 3** project database (*.hie) file from the previous **ISaGRAF** version, then click **Open**.

### See Also

Creating a Project
Creating a Library
Importing an ISaGRAF 3 Library

# Creating a Library

Libraries are special projects containing elements, i.e., functions, function blocks, conversion functions, I/O boards, I/O configurations, and I/O equipment. However, when creating **ISaGRAF** CAM 3 libraries, you can only create functions and function blocks for reuse throughout **ISaGRAF** CAM 3 projects. Libraries are available for use in a project after creating a dependency.




The **ISaGRAF 3** Library template is available for **ISaGRAF 3** projects. This template enables creating a library without attaching it to a new or existing solution. Empty libraries contain no device files.

A project can depend on one library and different projects can call the same library. When creating a library, it can only contain functions and function blocks. These library elements can be called from a project once the library is added as a dependency. Functions and function blocks can be written using the IEC 61131-3 languages (FBD, LD, or ST).

When a library is included in a solution with a project, the library elements are available for modification.

You create libraries as part of a solution in the **Automation Collaborative Platform**. A solution can hold multiple projects and libraries.

You base a library on a library template then develop its functions and function blocks. Libraries are stored in the same location as projects.

Library functions and function blocks must have unique names; these must have different names from those in the project in which they are used. You do not need to compile functions and function blocks in the library before using them in projects. These are compiled in the calling project space, in order to take care of the compiling options defined for the project.

When building solutions or projects, libraries included as dependencies are automatically compiled upon detecting a modification whether the library is part of the solution or external.

**To create an empty ISaGRAF 3 library**

You can create a library without attaching it to a new or existing solution. Empty libraries contain no device files.

1. From the File menu, point to **New**, then click **Project** (or press **Ctrl+Shift+N**).
2. In the Installed Templates list, expand the *CAM Projects* option, then expand **ISaGRAF 3**, and click **Empty**.
3. From the list of available project templates, select the **ISaGRAF 3 Library** template.
4. Specify a name and location for the project, indicate whether to add the project to an existing solution or create a new solution by defining a solution name, then click **OK**. For new solutions, you can choose to create a directory.
5. To define the compiler properties for the library, right-click the Functions or Function Blocks elements in the Solution Explorer, and then click Properties.

## See Also

Using a Library in a Project
Importing an ISaGRAF 3 Library
Creating a Project

# Importing an **ISaGRAF** 3 Library

You can import libraries from **ISaGRAF 3.x** for use in CAM3 projects or imported **ISaGRAF 3.x** projects. When importing libraries, all elements from the initial **ISaGRAF 3.x** library are available for reuse in projects. These elements include functions, function blocks, conversion functions, I/O boards, I/O configurations, and I/O equipment.

The Import **ISaGRAF 3** Library template is available for **ISaGRAF 3** projects. This template enables importing an **ISaGRAF 3** library into the **ISaGRAF 6.x** workbench.

A project can depend on one library and different projects can call the same library. When importing a library, it contains all elements defined in the original version. Library elements can be called from a project once the library is added as a dependency.

You can only add functions and function blocks to imported libraries; you cannot add conversion functions, I/O boards, I/O configurations, and I/O equipment. Functions and function blocks can be written using the IEC 61131-3 languages (FBD, LD, or ST).

You import libraries as part of a solution in the **Automation Collaborative Platform**. A solution can hold multiple projects and libraries.

Libraries are stored in the same location as projects.

Library functions and function blocks must have unique names; these must have different names from those in the project in which they are used. You do not need to compile functions and function blocks in the library before using them in projects. These are compiled in the calling project space, in order to take care of the compiling options defined for the project.

When building solutions or projects, libraries included as dependencies are automatically compiled upon detecting a modification whether the library is part of the solution or external.

**To import an ISaGRAF 3 library**

You can import libraries created using **ISaGRAF 3**.

1. From the File menu, point to **New**, then click **Project** (or press **Ctrl+Shift+N**).

2. In the Installed Templates list, expand the *CAM Projects* option, then expand **ISaGRAF 3**, and click **Import**.

3. From the list of available templates, select the **Import ISaGRAF 3 Library** template.

4. Specify a name and location for the library, indicate whether to add the library to an existing solution or create a new solution by defining a solution name, then click **OK**. For new solutions, you can choose to create a directory.

5. In the Selecting a *NUMS File dialog box, locate and select the **ISaGRAF 3** library database (*NUMS) file, then click **Open**.

### See Also

Using a Library in a Project
Creating a Library
Creating a Project

# Using a Library in a Project

Projects can use elements, i.e., functions, function blocks, conversion functions, I/O boards, I/O configurations, and I/O equipment, from a library. However, when using libraries created in **ISaGRAF** CAM 3, only functions and function blocks are available. You need to create libraries or import libraries before using them. Furthermore, you need to define a project's dependencies, i.e., the library the project will use, before using a library's defined elements. Multiple projects can depend on a library.

A library cannot use elements from another library. In other words, you cannot define external dependencies for a library. However, a function or function block from a library can call other functions or function blocks from the same library. Furthermore, functions or function blocks from libraries can call 'C' written functions and function blocks defined for the corresponding target.

All functions and function blocks within a project, including those coming from libraries, must have unique names.

You add a dependency onto a library from the Dependencies dialog box. In this dialog box, the Libraries list displays the library on which a project has a dependency.

**Note:** When redefining the location of a library dependency you can modify the path in the library properties; removing the library will result in a loss of all project references.

When building solutions or projects, libraries included as dependencies are automatically compiled upon detecting a modification whether the library is part of the solution or external.

**To use a library in a project**

1. In the Solution Explorer, expand the project for which to add a dependency.
2. Right-click the Dependency element, point to **Add**, and then click **Add Dependency...**.
3. In the Dependencies dialog box, click Browse to locate the library on which to create the dependency.

The library is displayed in the Libraries list.

**See Also**

Creating a Library
Importing an ISaGRAF 3 Library
Creating a Project

# Importing and Exporting Variables Data

You can import variables that were previously exported and saved as Microsoft Excel spreadsheets (.xls). Exporting variables enables management of variables data in Excel, including adding, removing, and modifying variables. You can import previously exported Excel files into other programs in the same project or in other projects.

When importing variables, you import the complete contents of the *.xls file. For previously exported Excel files containing modified content, any additional columns of data using proper syntax will be imported. The Output window details the progress of import operations, including the names and location of the variables added.

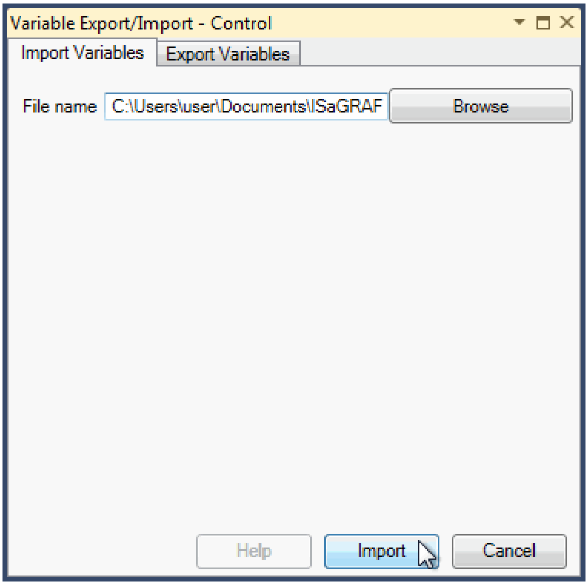
When exporting variables, you can select the fields of the variables to export. You also specify the location in which to save the exported files.

You can also import files containing manually defined variables for use in devices and programs. When importing files created manually, you must include a header row containing the same syntax used in files exported from **ISaGRAF**. The Excel file syntax uses the internal names for the columns of data instead of those displayed in the Variable Export/Import dialog box. Any rows of data using improper syntax will not be imported.

**To import variables**

You can only import variables having been previously exported and stored as Excel (.xls) files.

1. In the Solution Explorer, right-click the device or POU, point to Import, then click **Variables from Excel...**.
2. In the Variable Export/Import dialog box, on the *Import Variables* tab, click browse to select the Excel file to import.
    - In the **Import/Export File** dialog box, select the Excel file to import, then click **Open**.
3. In the Variable Export/Import dialog box, click **Import**.




4. When the import process is complete, in the Variable Export/Import dialog box, click .

The imported variables are available for use.

**To export variables**

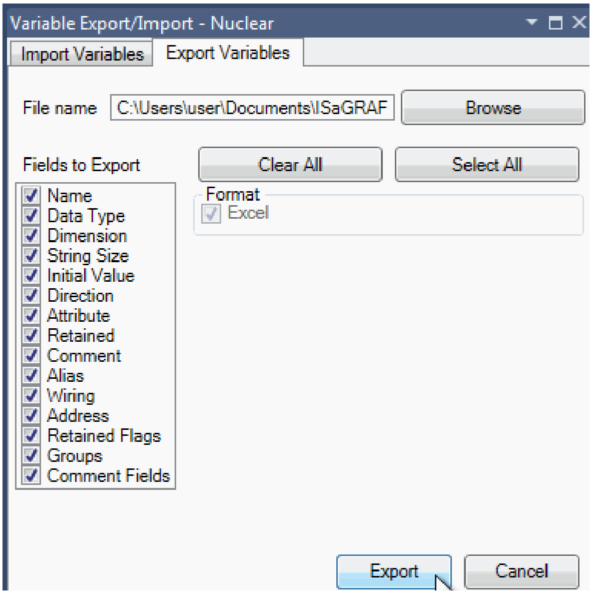
You can export selected fields of variables data in Excel (.xls) format.

1. In the Solution Explorer, right-click the device or POU containing the variables to export, point to Export, then click **Variables to Excel...**.

2. In the Variable Export/Import dialog box, on the *Export Variables* tab, click browse to select the destination for the exported variables.

3. In the **Import/Export File** dialog box, specify the name of the Excel file, then click **Save**.

**4.** From the *Fields to Export* check box list, select the variables data to export, then click **Export**.




Using the *Select All* option, you can select all the fields displayed. The *Clear All* option enable you to clear all fields, then reselect only those required.

**5.** When the export process is complete, click  .

The variables are exported to the specified file.

# Generating Code

Before downloading code onto your target systems, you need to build the code for the whole solution. This operation builds the code for all projects within the solution, and builds information used to recognize your systems on networks. When a solution contains more than one project, you can build the code for individual projects within the solution. Once a solution or project has been built, subsequent build operations only regenerate the parts of the solution or project needing regeneration. You can also choose to build project elements, including devices and POUs. When building POUs, **ISaGRAF** only verifies the programming syntax without producing code.

When managing code, you can perform the following tasks:

- Building Solutions and Project Elements
- Rebuilding Solutions
- Cleaning Solutions and Project Elements

# Building Solutions and Project Elements

You can choose to compile project files that were modified since the last build. You can build modified project files belonging to entire solutions. Once a project has been built, subsequent builds only recompile the parts of the project needing recompiling.

When a solution contains more than one project, you can build the modified project files for individual projects. You can also choose to build individual project elements including devices and POUs.

You can rebuild solutions to ensure that the compiled version is up-to-date. When rebuilding solutions, intermediate and output files are deleted, then a build operation is performed. Deleting the intermediate and output files ensures that the entire solution is compiled during a rebuild operation. After rebuilding solutions, online changes become unavailable.

The compiler generates different code for simulation than for targets. Therefore, you need to specify the applicable target in the properties of devices before building.

When building solutions and project elements, you can view the progress of the build in the Output window. When the build is complete, you can view generated errors in the Error List.

**To build a solution or project element**

This operation builds the code for all devices of the projects and builds information used to recognize your systems on networks. You cannot build projects open in read-only mode. Before building a project, make sure the applicable target type is specified for the devices.

- In the Solution Explorer, right-click the required solution or project element, then click **Build**.

The build process is initiated for the required project element or solution.

**To view the build progress and generated errors**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Projects**, click **General**, then select the following options, and then click **OK**.

- Always show Error List if build finishes with errors
- Show Output window when build starts

3. Build the required solution or project element.

The Output and Error List windows are displayed.

**See Also**

Downloading Code to Targets
Rebuilding Solutions

# Rebuilding Solutions

You can choose to clean solutions, deleting the intermediate and output files, then rebuild all project files and components. After rebuilding solutions, online changes become unavailable.

You can view the progress of rebuild operations in the Output window. When the rebuild is complete, you can view generated errors in the Error List.

**To rebuild a solution**

1. In the Solution Explorer, click the solution element.
2. From the Build menu, click **Rebuild Solution**.

The rebuild process is initiated for the solution.

**To view the rebuild progress and generated errors**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Projects**, click **General**, then select the following options, and then click **OK**.
   - Always show Error List if build finishes with errors
   - Show Output window when build starts
3. Build the required solution.

The Output and Error List windows are displayed.

### See Also

Downloading Code to Targets

# Cleaning Solutions and Project Elements

You can clean solutions, projects, and devices. Cleaning these deletes the intermediate and output files generated during the last build operation. Performing cleaning operations removes the capacity to perform online changes for the selected element. For example, after cleaning a device, online changes become unavailable.

**To clean a solution**

- In the Solution Explorer, right-click the solution, then click **Clean Solution**.

The intermediate and output files are deleted for the solution.

**To clean a project or device**

- In the Solution Explorer, right-click the required project or device then click **Clean Selection**.

The intermediate and output files are deleted for the project element.

### See Also

Building Solutions and Project Elements
Rebuilding Solutions

# Running an Application Online

Running online signifies that an application is connected to a target allowing for the normal execution where target cycles are triggered by the cycle timing. While running online, you can perform target management, debugging, and monitoring operations. However, you cannot perform target management and debugging operations at the same time. You can also simulate the running of an application for debugging purposes.

Before running an application on a target, you need to build the project code and download the application code onto the target.

**To run an application online**

1. Specify the applicable target type and IP addresses for the devices in the project.

**Note:** The compiler generates different code for simulation than for targets.

2. Build the project code.
3. To run an application online, download the application code onto the target.
4. In the Debug toolbar, from the drop-down combo-box, select **Online**.
5. In Debug menu, click **Start Debugging**.

### See Also

Simulating
Debugging
Monitoring

# Downloading Code to Targets

You perform download operations for projects having devices with code to send to targets. When simulating a project, you do not need to perform a download operation.

The code (corresponding to the run-time engine capabilities) must first be generated by building the project. The code type is determined by the target definition.

The Configuration manager must be running on the target platform.

The computer where the **Automation Collaborative Platform** is installed must be connected to the hardware equipment through a network supported by the Debugger. The standard networks used by the **Automation Collaborative Platform** are TCP/IP and Serial COM port (ISaRSI).

**To download project code to a target**

1. Build the project code.
2. In the Solution Explorer, right-click the project element, and then click **Download**.

# Debugging

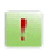
When developing an application, you can choose to debug, i.e., detect and remove errors, from a project while running the application online, i.e., on a target, or simulating. Before running an application online, you need to download the application code onto the target.

While in real-time mode, the device is executed by a virtual machine on the real platform. A download operation is required to download the device code onto the corresponding platform.

A device where real-time mode is activated is in the RUN state.

When debugging, the state of a device is displayed in its icon in the Solution Explorer. The possible states of a device are the following:

 The device is running on the target. The device is in the RUN, STOP, or ERROR state.

 The device is not running on the target or no code is available on the target. The device is in the DISCONNECTED or NO APPLICATION state.

To enable debugging a project, you must first build the project, then download the project code to the target.

When switching an application to debugging, the **Automation Collaborative Platform** verifies the coherency between the current device definitions and the devices' compiled code. The **Automation Collaborative Platform** also verifies the coherency between all versions of the device code.

You can execute a device in one of two execution modes:

- Real-time, the run time normal execution mode where target cycles are triggered by the programmed cycle timing. While in real-time mode, you can switch the device to cycle-to-cycle mode.

- Cycle-to-cycle, a cyclical execution mode where the virtual machine loads the device code but does not execute it until you execute one cycle or activate real-time mode.

The state of the device appears next to the device icon in the Solution Explorer.

| Device State | Description |
|---|---|
| RUN | The device is running in real-time mode |
| STOP | The device is in cycle-to-cycle mode.<br>Possible operations are:<br>- switch the device to real-time mode<br>- execute one cycle |
| ERROR | The device is in error.<br>Possible operations are:<br>- switch the device to real-time mode<br>- switch the device to cycle-to-cycle mode<br>- execute one cycle |
| DISCONNECTED | Unable to establish communication with the target run-time. |
| NO APPLICATION | The device is not running on the target or no code is available on the target. |

When running online, a device is activated in the RUN state. When viewing the values of variables in dictionary instances, the logical and physical values display the following temporary messages before loading the actual values:

- OFFLINE, indication that the variable is not present in the running application code
- WAIT, indication that the variable is either:
    - In online mode and attempting to connect to the target
    - In simulation mode and attempting to connect to the simulator

**To debug an application**

Before debugging an application, you need to build the application code and download the code to the target.

1. Build the project code.
2. Download the code to the target.
3. In the Debug toolbar, from the drop-down combo-box, select **Online**.

4. From the Debug menu, click **Start Debugging** (or press **F5**).

### See Also

Devices
Forcing the Values of Variables

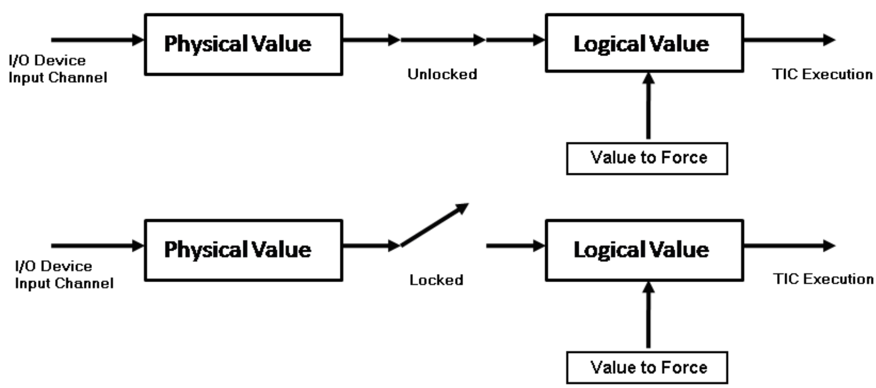
# Forcing the Values of Variables

While debugging, you can force, i.e., override, the values of variables. These variables can be user-defined or directly represented. The behavior of a variable is defined by its logical value, physical value, lock state, and direction. When forcing the values of variables, the value to overwrite depends on the direction of the variable. You lock, unlock, and force the values of variables from the Dictionary.

For locked variables, the values displayed in the Logical Value and Physical Value columns differ depending on their direction. Variable direction is determined from the direct representation definition for the I/O wiring.

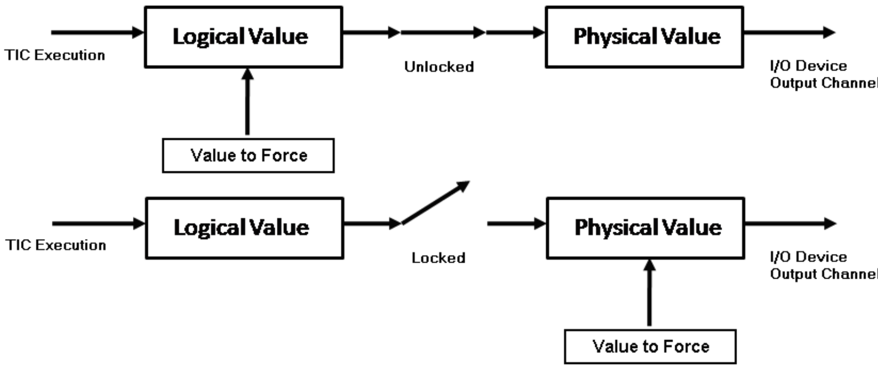
### Input Variable (Read) Behavior

Example: To force the temperature reading from a sensor.



### Output Variable (Write) Behavior

Example: To force the closing of an actuator valve.




When forcing the values of unlocked variables, these values may be overwritten by the next cycle execution.

### To force the value of a variable

While debugging, you can force the values of locked directly-represented variables.

1. From the Dictionary instance, double-click the required variable.

   The *Write* dialog box is displayed.

2. To modify the lock on the variable, in the *Lock* field, click the slider, then click **Write**.

3. To write the required value for the variable, modify the *DataType value* field, then click **Write**.

   When modifying a date in the *DataType value* field, a calender box is displayed. To select a date, click within the calender box. You can move between months using the arrow buttons.

**See Also**
Debugging

# Simulating

Simulating the running of an application signifies that virtual machines execute the code of the device and the Windows platform performs aspects such as POU execution. Virtual machines ignore inputs and outputs.

The compiler generates different code for simulation than for online.

Before simulating an application on a target, you need to build the project code.

**To simulate the running of an application**

1. In the Device properties, specify the applicable target type and host address for the device.
2. Build the project code.
3. In the Debug toolbar, from the drop-down combo-box select **Simulation**.
4. From the Debug menu, click **Start Debugging**.

# Monitoring

While running an application online, debugging, or simulating, you can monitor variables, updated by the running online (TIC) code or simulation code, in Dictionary instances as well as graphical programs and function block instances. Generating monitoring information increases the size of the TIC code created.

For dictionary instances, the logical values, physical values, and lock status of variables are displayed in their respective columns. For graphical programs and function block instances, values are displayed differently depending on their type:

- Boolean type variables are displayed using color. The variable color continues to the next input. The default colors are red when True and blue when False.
- DINT, REAL, MESSAGE, and TIME type variables are displayed as a numeric or textual value. When the variable is a structure type, the displayed value is the selected member.

When variables are unavailable, in Dictionary instances, the logical and physical values for variables display the following messages:

- OFFLINE, indication that the variable is not present in the running application code
- WAIT, indication that the variable is either:
    - In online mode and attempting to connect to the target
    - In simulation mode and attempting to connect to the simulator

**See Also**
Running an Application Online
Debugging
Simulating

# Error Messages

The following describes the error types:

| Error Type | Source | Solution |
|---|---|---|
| System Errors | Error due to target software or hardware<br>Report this type of error to **ISaGRAF** Support | Perform a hard reset of your target, then attempt to run other applications. |
| Application Errors | Error due to application parameters, size, or content | Load a previously validated application |
| Program errors | Error due to a particular program sequence | Start the application in cycle-to-cycle mode or stop the critical program |

Possible error messages encountered in **ISaGRAF 3** Concrete Automation Model include:

| Message | Type | Description |
|---|---|---|
| Unable to allocate memory for run-time data base | System | Indication that no memory available Verify your hardware memory |
| Incorrect application database or bad CRC | Application | Indication that the application file is incorrect<br>Occurs when the application is generated for INTEL and downloaded on MOTOROLA (and reverse), or if the file has been altered. |
| Unable to allocate communication mailbox | System | The communication task is unable to allocate space 3 for inter-task communication. |
| Unable to link kernel database | System | The communication task cannot find a kernel running with the slave number specified in its command line. |
| Time-out while sending request to kernel | System | The communication task cannot send a request to the kernel. The kernel is not running or busy. |

| | | |
|---|---|---|
| Time-out while awaiting response from kernel | System | The communication task cannot receive an answer from the kernel. The kernel is not running or busy |
| Unable to initiate communication | System | The communication layer cannot initialize a physical link or no communication path is specified.<br><br>This error does not interfere with target function. |
| Unable to allocate memory for retain variables | Application | **ISaGRAF** cannot manage retained variables:<br><br>- the string passed as a parameter to the host target is not syntactically correct<br><br>- the size of memory specified for each block is not sufficient<br><br>To resolve, verify the syntax of your 'retain variable' parameter or reduce the number of retained variables. |
| Application stopped | Application | The application was stopped from the debugger. |
| Unknown TIC instruction | Application | The kernel has detected a problem in the Target Independent Code for a program for one of the following reasons:<br><br>- an external program is writing to the application code. Locate this problem in cycle-to-cycle mode and verify all I/O interface parameters.<br><br>- the target has a reduced set of instructions and the application uses a non-authorized instruction or variable type. |

| | | |
|---|---|---|
| Unable to answer read data request | System | A communication error was detected when answering the specific **ISaGRAF** Modbus request function code 18 (file read).<br><br>Verify the connection and system configuration on the target and master sides. |
| Unable to answer write data request | System | A communication error was detected when answering specific **ISaGRAF** Modbus request function code 17 (file write).<br><br>Verify the connection and system configuration on the target and master sides. |
| Unable to answer debugger session request | System | A communication error was detected when answering a debugger request.<br><br>Verify the connection and system configuration on the target and master sides. |
| Unable to answer modbus request | System | A communication error was detected when answering a Modbus request.<br><br>Verify the connection and system configuration on the target and master sides. |
| Unable to answer debugger application request | System | A communication error was detected when answering a debugger request.<br><br>Verify the connection and system configuration on the target and master sides. |
| Unable to answer debugger | System | A communication error was detected when answering a debugger request.<br><br>Verify the connection and system configuration on the target and master sides. |

| | | |
|---|---|---|
| Unknown request code | System | Unable to interpret the debugger request |
| Ethernet communication error | System | The connection is closed and the debugger is closed. Otherwise, an Ethernet communication error was detected.<br><br>Verify the connection and system configuration on the target and master sides.<br><br>When a second field is provided, verify for the following possible errors:<br><br>1: error while sending or receiving<br><br>2: error while creating the socket<br><br>3: error while binding or listening the socket<br><br>4: error while accepting a new client |
| Communication synchronization error | System | The tasks providing communication between the target and the master are not synchronized.<br><br>Verify the connection and system configuration on the target and master. |
| Unable to allocate memory for application | System | There is insufficient memory available.<br><br>Ensure that enough hardware memory is provided to accommodate the size of the application. |
| Unable to allocate memory for application update | System | There is insufficient memory available.<br><br>Ensure that enough hardware memory is provided to accommodate the size of the application. |

| | | |
|---|---|---|
| Unknown OEM key code | Application | A board used in your applications has a manufacturer code that is not recognized.<br><br>In the workbench, verify the following:<br>- I/O connections<br>- locate the board use the 'VIRTUAL' attribute |
| Unable to initiate boolean input board | Application | A Boolean input board initialization has failed.<br><br>In the workbench, verify the following:<br>- I/O connections<br>- parameters of the Boolean input boards |
| Unable to initiate analog input board | Application | An analog input board initialization has failed.<br><br>In the workbench, verify the following:<br>- I/O connections<br>- parameters of the analog input boards |
| Unable to initiate message input board | Application | A message input board initialization has failed.<br><br>In the workbench, verify the following:<br>- I/O connections<br>- parameters of the message input boards |
| Unable to initiate boolean output board | Application | A Boolean output board initialization has failed.<br><br>In the workbench, verify the following:<br>- I/O connections<br>- parameters of the Boolean output boards |
| Unable to initiate analog output board | Application | An analog output board initialization has failed.<br><br>In the workbench, verify the following:<br>- I/O connections<br>- parameters of the analog output boards |

| | | |
|---|---|---|
| Unable to initiate message output board | Application | A message output board initialization has failed.<br><br>In the workbench, verify the following:<br>- I/O connections<br>- parameters of the message output boards |
| Unable to input boolean board | Application | An error has been detected while refreshing a Boolean input board.<br><br>In the workbench, verify the I/O connection and the board parameters. |
| Unable to input analog board | Application | An error has been detected while refreshing an analog input board.<br><br>In the workbench, verify the I/O connection and the board parameters. |
| Unable to input message board | Application | An error has been detected while refreshing a message input board.<br><br>In the workbench, verify the I/O connection and the board parameters. |
| Unable to update boolean output variable | Application | An error has been detected while updating an Boolean output variable.<br><br>In the workbench, verify the I/O connection and the board parameters. |
| Unable to update analog output variable | Application | An error has been detected while updating an output analog variable.<br><br>In the workbench, verify the I/O connection and the board parameters. |
| Unable to update message output variable | Application | An error has been detected while updating an output message variable.<br><br>In the workbench, verify the I/O connection and the board parameters. |

| | | |
|---|---|---|
| Unable to perform Operate call on boolean variable | Application | An error has been detected while executing an OPERATE call to a Boolean variable.<br><br>Verify the OPERATE parameters and consult the help files for your I/O board. |
| Unable to perform Operate call on analog variable | Application | An error has been detected while executing an OPERATE call to a analog variable.<br><br>Verify the OPERATE parameters and consult the help files for your I/O board. |
| Unable to perform Operate call on message variable | Application | An error has been detected while executing an OPERATE call to a message variable.<br><br>Verify the OPERATE parameters and consult the help files for your I/O board. |
| Unable to open board | Application | The application uses a board reference, which is unknown to the target.<br><br>In the workbench, verify the following:<br>- I/O connections<br>- ensure that the workbench library version corresponds to the target version |
| Unable to close board | Application | The application uses a board reference, which is unknown to the target.<br><br>In the workbench, verify the following:<br>- I/O connections |
| Unknown system request code | Program | A program is using a SYSTEM call with an invalid code. |

| | | |
|---|---|---|
| Sampling period overflow | Program | The target cycle period is longer than specified in the workbench. For a multitasking system, there is not enough CPU time to execute a cycle, even when the 'current cycle duration' is less than the specified period. For a single task system, there are too many operations in one target cycle.<br><br>To resolve, do the following:<br>- reduce the number of operations performed at the instant where the warning is detected.<br>- reduce the number of tokens and valid transitions, and optimize complex processing, etc.<br>- reduce the CPU load dedicated to other tasks<br>- reduce the communication traffic<br>- adapt the cycle duration to different process stages by use dynamic cycle duration modification<br>- set cycle duration to zero |
| User function not implemented | Application | A program is using a C function that is unknown to the target.<br><br>Ensure that the workbench library version corresponds to the target version. |
| Integer divided by zero | Program | A program has attempted to divide an integer analog by zero. This type of event can have unpredictable effects. When a divide by zero operation occurs, **ISaGRAF** places the maximum analog value as the result. When the operand is negative, the result is inverted. |

| | | |
|---|---|---|
| Conversion function not implemented | Application | A program is using a C conversion function that is unfamiliar to the target. **ISaGRAF** has not converted the value.<br><br>Ensure that the workbench library version corresponds to the target version. |
| Function block not implemented | Application | A program is using a C function block that is unfamiliar to the target.<br><br>Ensure that the workbench library version corresponds to the target version. |
| Standard function not implemented | Application | A program is using a function block that is unfamiliar to the target. This function block is available for most targets.<br><br>Please contact your supplier. |
| Real divided by zero | Program | A program has attempted to divide a real analog by zero. This type of event can have unpredictable effects. When a divide by zero occurs, **ISaGRAF** places the maximum real analog value as the result. When the operand is negative, the result is inverted. |
| Invalid operate parameters | Application | Your application has used an OPERATE call with incorrect parameters.<br><br>To resolve, do the following:<br>- verify that the timer parameters are correct<br>- verify that all variables are inputs or outputs |
| Application symbols cannot be modified | Application | After attempting to perform an update, **ISaGRAF** was unable to start the application. The application symbols have changed. One or more variables or instances of function blocks were added, removed or modified. |

| | | |
|---|---|---|
| Unable to update: different set of boolean variables | Application | **ISaGRAF** is unable to start the modified application. One or more Boolean variables have been added or removed. |
| Unable to update: different set of analog variables | Application | **ISaGRAF** is unable to start the modified application. One or more analog variables have been added or removed. |
| Unable to update: different set of timer variables | Application | **ISaGRAF** is unable to start the modified application. One or more timer variables have been added or removed. |
| Unable to update: different set of message variables | Application | **ISaGRAF** is unable to start the modified application. One or more message variables have been added or removed. |
| Unable to update: cannot find new application | Application | Unable to find the modified application in memory. An error may have occurred when downloading. |

# Getting Started

The **ISaGRAF 3** Concrete Automation Model enables the creation of applications supporting multi-process control. Applications consist of virtual machines running on hardware components, called target platforms. The development process consists of creating a project composed of one device that is downloaded to a target platform. At runtime, the device becomes a virtual machine running on the target platform.

Projects containing a device and one or more programs are developed in the following languages of the IEC 61131-3 standard: FBD: Function Block Diagram, LD: Ladder Diagram, and ST: Structured Text. When building, a device is compiled to produce very fast "target independent code" (TIC) or "C" code.

Within devices, you can declare variables using standard IEC 61131-3 data types (i.e., BOOL, DINT, REAL, MESSAGE, and TIME) or user-defined types such as one-dimensional arrays.

You develop projects on a Windows® development platform. The Automation Collaborative Platform graphically represents and organizes the device, POUs, variables, and networks within a project from many views:

- Add-in Manager
- Block Selector
- Data Types
- Description Window
- Dictionary
- Error List
- Find and Replace
- ISaVIEW
- Locked Variables Viewer
- Options...
- Parameters View
- Solution Explorer
- Toolbox
- Variable Selector
- Block Library
- Customize...
- Deployment View
- Device View
- Document Overview
- External Tools
- I/O Wiring
- Language Editors
- Navigation Window
- Output Window
- Properties Window
- Spy Lists
- Variable Dependencies

Libraries are special projects made up of devices enabling the definition of functions and function blocks for reuse throughout projects.

Projects are downloaded, using the TCP-IP or Serial network driver, onto target platforms running real-time operating systems. Communication between devices can be implemented using the default TCP-IP network or proprietary network protocol.

Before downloading project code onto the target platform, you need to build the code for the entire solution. You can then choose to debug the application while running online or simulating. You can also monitor variables while running the application online, debugging, or simulating.

The following information guides you in getting started with the **ISaGRAF 3** Concrete Automation Model:

- System Requirements for Development Platforms
- Naming Conventions and Limitations
- Introducing the Automation Collaborative Platform (ACP)
- Walking Through an Existing Application
- Starting with a Basic Application
- Importing an Existing Application

# System Requirements for Development Platforms

## Suggested Requirements

To use **ISaGRAF**, you need the following hardware and software.

### Hardware

- A computer with a 2.2 GHz or faster processor.
- RAM
  - 1 GB of RAM for x86 operating systems
  - 2 GB of RAM for x64 operating systems
  - When running ISaGRAF on a Virtual Machine, an additional 512 MB of RAM is necessary
- 4 GB of available hard disk space
- A hard disk running at 5400 RPM
- A CD-ROM drive on the Windows network (for installation from disk)
- A TCP/IP network
- An SVGA monitor having at least 1024 X 768 pixels screen resolution
- A DirectX 9-capable video card that runs at a display resolution of 1024 x 768 or higher

### Software

**ISaGRAF** supports the following operating systems:

- Windows® 7 (x86 and x64)
- Windows® 8 (x86 and x64)

**Note:** If Visual Studio 2010 was previously installed, when running the ISaGRAF installation the Visual Studio 2010 Service Pack 1 will be installed. This may affect Visual Studio functionality.

# Naming Conventions and Limitations

**Projects**

| | |
|---|---|
| Project names | Project names are recommended to have up to 32 characters |
| Device quantity | Projects contain one device |

**Devices**

| | |
|---|---|
| Device names | Device names can have up to eight (8) characters and must begin with a letter followed by letters, digits, and single underscores. |

**Networks**

| | |
|---|---|
| Network instance names | Network instance names can have up to eight (8) characters and must begin with a letter followed by letters, digits, and single underscores. |

**POUs (Programs, Functions, and Function Blocks)**

| | |
|---|---|
| POU names | POU names can have up to eight (8) characters and must begin with a letter followed by letters, digits, and single underscores. |
| POUs per project | The maximum number of POUs is directly dependent on the ISaGRAF 3 run-time 16-bit application. |
| Hierarchical levels | The maximum hierarchical levels for POUs is 20 |
| Function parameters | Functions can have a maximum of 32 parameters (31 inputs and one output) |
| Function parameter names | Function parameter names can have up to 32 characters and must begin with a letter followed by letters, digits, and single underscores. |
| Function block parameters | Function blocks can have a maximum of 32 parameters (inputs and outputs) |
| Function block parameter names | Function block parameter names can have up to 32 characters and must begin with a letter followed by letters, digits, and single underscores. |

**Variables**

| | |
|---|---|
| Variable quantity | The maximum number of variables is directly dependent on the ISaGRAF 3 run-time 16-bit application. |

| | |
|---|---|
| Variable names | Variable names can have up to 32 characters and must begin with a letter followed by letters, digits, and single underscores.<br><br>The names of variables having a defined Modbus address, initial value, or retained property are calculated by combining the variable and POU names for a maximum of 32 characters beginning with a letter followed by letters, digits, and single underscores. |
| Boolean variables | Boolean variables can have the boolean value TRUE (1) or FALSE (0) and can have an internal, constant, input, or output attribute. |
| DINT variables | DINT variable integer values range from -2147483647 to +2147483647 and can have an internal, constant, input, or output attribute. Integer literals must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| Real variables | Real variables have six significant digits. For larger values, the maximum possible value is ±3.402823466E+38 while for smaller values, the minimum possible value is ±1.175494351E-38. Real literal values can be written with either decimal or scientific representation. The exponent part of a real scientific expression must be a signed integer value ranging from -37 to +37. The scientific representation uses the 'E' or 'F' letter to separate the mantissa part and the exponent. Real variables can have an internal, constant, input, or output attribute. |
| Time variables | Time variables can have positive values ranging from 0 to 23h59m59s999ms. The time literal value must begin with the "T#" or "TIME#" prefix. Time variables can have an internal or constant attribute. |
| MESSAGE variables | MESSAGE variable string capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string. Characters must be preceded and followed by single quote (') characters. When placing single quote (‘) characters within a message literal, these characters must be preceded by the dollar ($) character. |
| Modbus Address | The Modbus address of a variable consists of four hexadecimal digits ranging from 0001 to FFFF. |
| Arrays | One-dimensional arrays can have a maximum of 255 elements. |

**Defined Words**

| | |
|---|---|
| Defined Word names | Defined word names can have up to 32 characters and must begin with a letter follow by letters, digits, and single underscores. |
| Defined word equivalents | Defined word equivalents can have up to 255 characters. |

**I/O Wiring**

| | |
|---|---|
| I/O devices per project | The maximum number of I/O device instances is directly dependent on the ISaGRAF 3 run-time 16-bit application. |
| Hardware racks | A hardware rack can contain up to 255 I/O boards |
| I/O boards per project | The maximum number of single I/O boards in a project is 255. |
| I/O channels | Each I/O board can have up to 128 I/O channels. These channels can be inputs or outputs. |
| I/O device order | The I/O device order ranges from 0 to 254 |
| Conversion table names | Conversion table names can have up to 16 characters and must begin with a letter follow by letters, digits, and single underscores. |

**FBD Programs**

| | |
|---|---|
| Label elements | Label elements can have up to 32 characters and must begin with a letter follow by letters, digits, and single underscores. |

**LD Programs**

| | |
|---|---|
| Label elements | Label elements can have up to 455 characters and must begin with a letter follow by letters, digits, and single underscores. |
| Rung comments | Rung comments can have up to 251 characters. |

**ST Programs**

| | |
|---|---|
| ST statements | ST statements (i.e., one line of code) are recommended to have less than 4096 characters. |

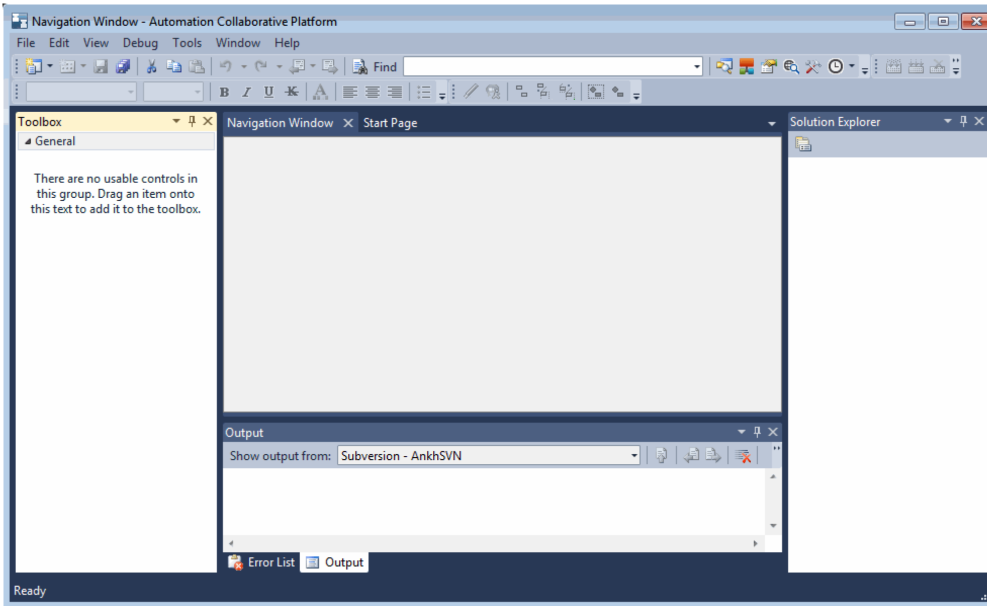
# Introducing the Automation Collaborative Platform (ACP)

The **Automation Collaborative Platform** (**ACP**) provides a robust integrated development environment (IDE) enabling the development of process control applications. The **ACP** workbench offers a complete suite of tools for building applications.

**To get to know the aspects of the ACP**

1. From the Start menu, click **All Programs**, then **ISaGRAF 6.4**, and then click **Automation Collaborative Platform**.

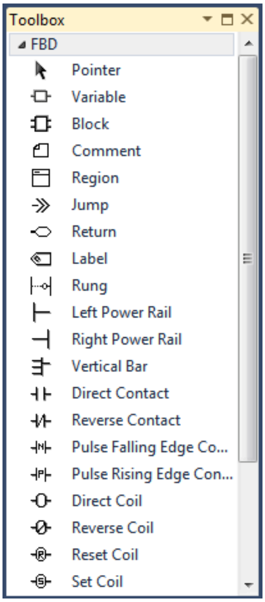
   The **ACP** is launched displaying the Start Page, Solution Explorer, Navigation Window, Toolbox, Output window, and Error List.



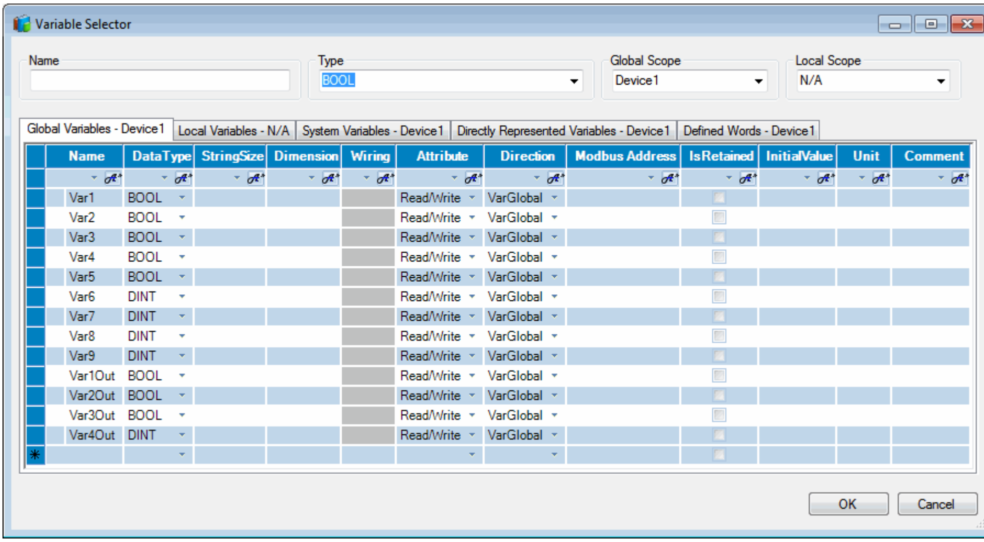

The **Start Page** enables opening new or recent projects, viewing tutorials, as well as accessing the Getting Started help pages. The **Solution Explorer** displays open solutions consisting of projects and their elements. The **Navigation Window** provides a global view of the solution and enables accessing the device view and deployment view. The **Toolbox** displays the available elements for insertion in programs. The **Output window** displays the compilation progress and errors. The **Error List** displays the errors, warnings, and messages produced when editing and compiling programs.

2. When adding elements in the language container, you can use the following **ACP** features:

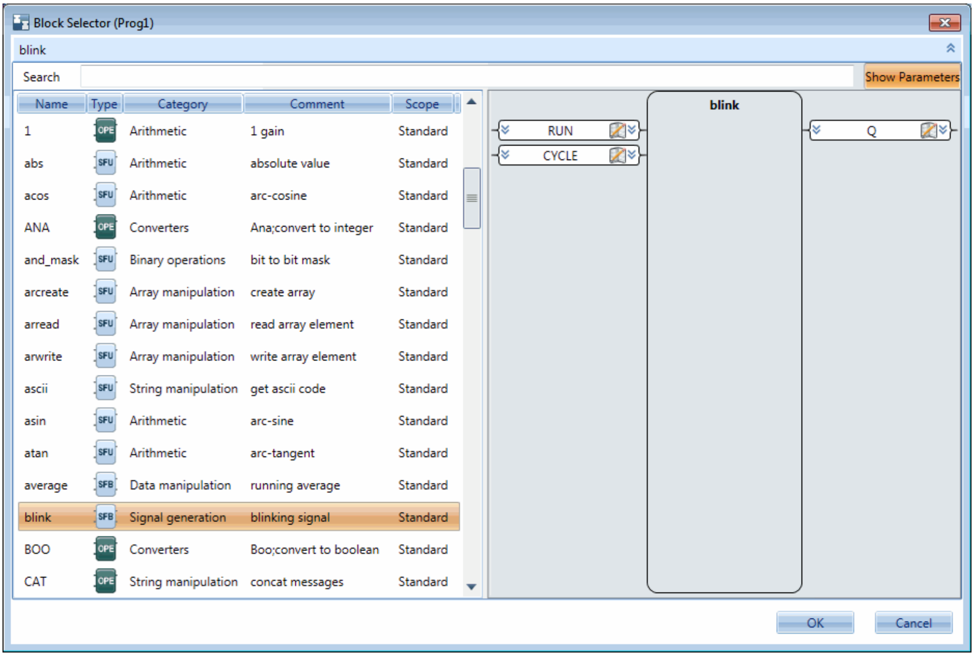
   - To display program-specific elements for insertion in the language container, from the View menu, click **Toolbox**.



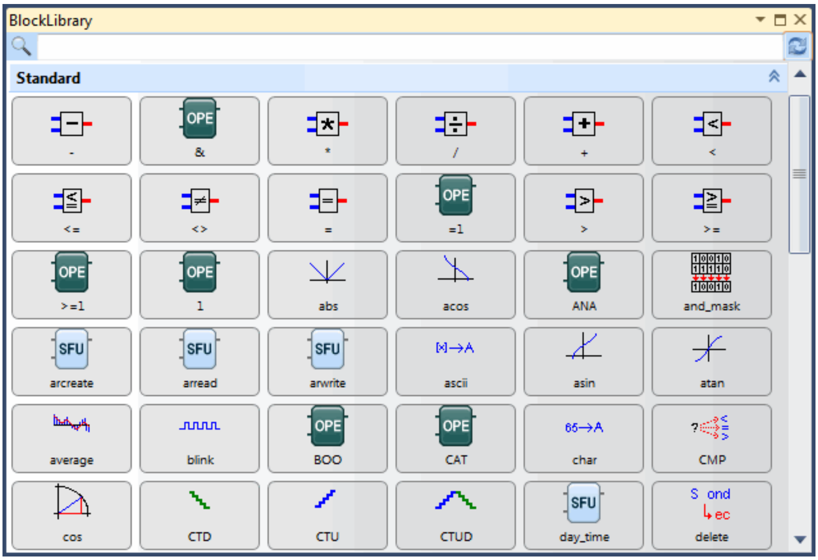
▣ To display variables defined for a program, from the Toolbox, drag the Variable icon into the language container. The **Variable Selector** is displayed.



- To display the list of blocks available for a program, from the Toolbox, drag the Block icon into the language container. The **Block Selector** is displayed. You can also access the **Parameters** display from the Block Selector.

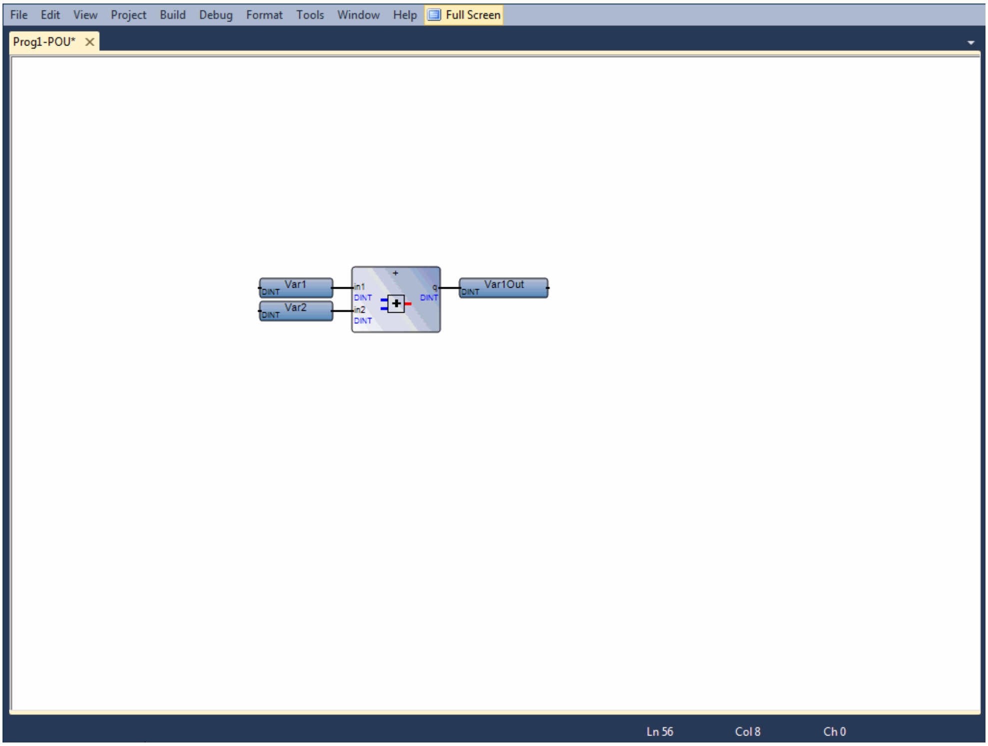


- To display a graphical view of standard operators, as well as standard and user-defined functions and function blocks available for the POUs of a project, from the View menu, click **Block Library**.

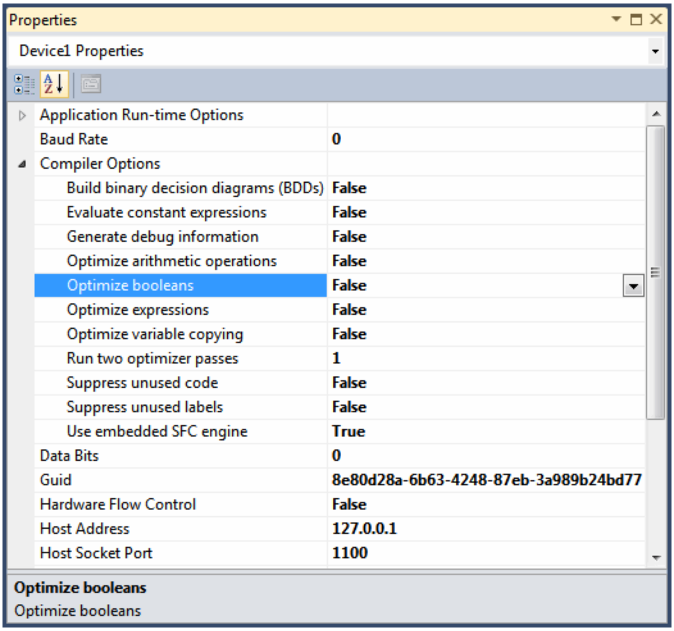



- To view, add, or edit the rich text descriptions for ISaGRAF projects, devices, and POUs, select the required element in the Solution Explorer, then from the View menu, click **Description Window**.

**3.** To work in full screen mode, from the View menu, click **Full Screen**. Full screen mode enlarges the workspace to fill the screen, hiding other tabbed windows.

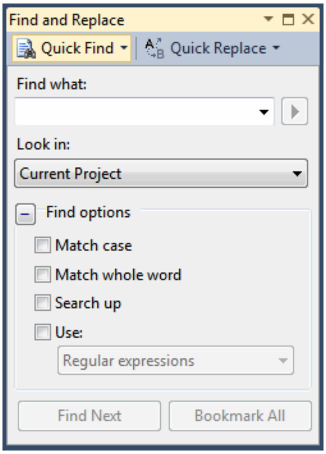



**4.** To display the Properties window, from the View menu, click **Properties Window**. The properties window enables viewing and editing the properties of items selected within language containers, ISaVIEW instances, the Solution Explorer, and the Deployment View. You can view properties alphabetically or categorically.

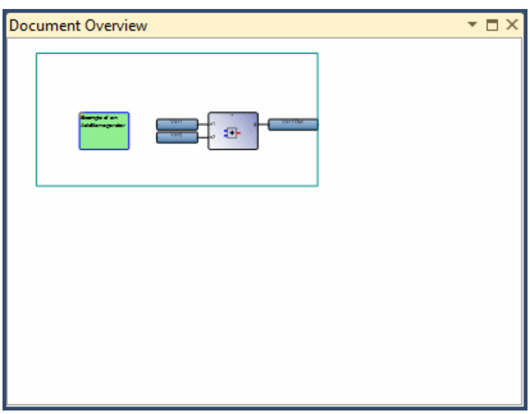



**5.** You can navigate through program content, including application code, using the following **ACP** features:

- To find and replace strings and expressions in files, from the Edit menu, point to **Find and Replace**, then click the required option. For example, click Quick Find to display Quick Find options.

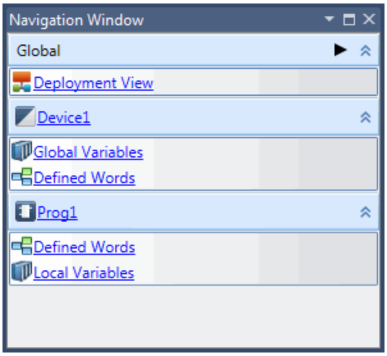



- To focus on an area displayed within a program opened for editing, from the View menu, click **Document Overview**.



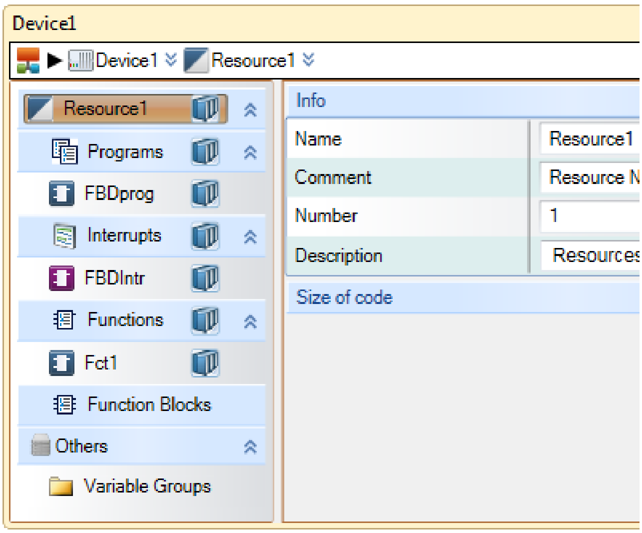

**6.** You can navigate through the different elements and aspects of projects using the following **ACP** features:

- To navigate through project aspects and elements, from the View menu, click **Navigation Window**. The environment provides a global view of the solution and enables accessing the Device view and Deployment view.

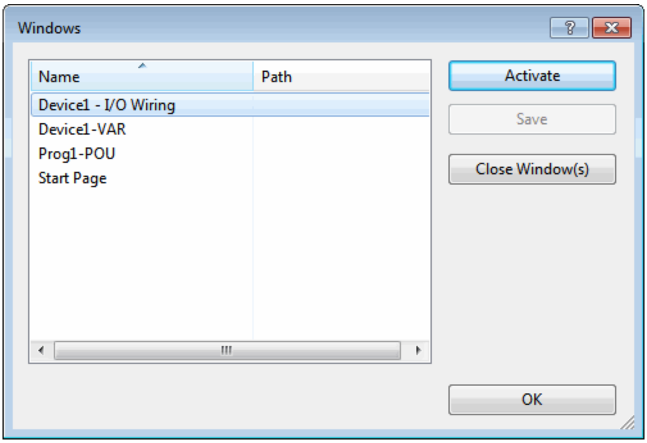



The initial aspects and elements displayed vary depending on the item selected in the Solution Explorer.

- To navigate through project elements, from the Solution Explorer, right-click the required device and then click **Open**. The **Device View** is displayed and enables accessing device information such as available POUs, function and function block parameters, and defined words.

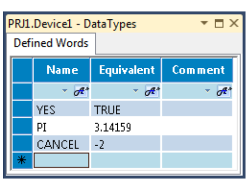


- To navigate through Active Files open in the current project, from the Window menu, click **Windows**. Active files consist of language containers, the Deployment view, and other windows docked in the workspace.

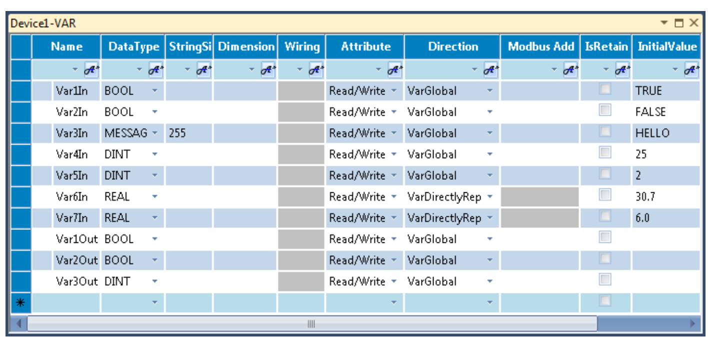



7. When managing elements, you can use the following **ACP** features:

- To manage local variables, global variables, and defined words, in the Solution Explorer, double click the required Local Variables, Global Variables, or Defined Words instance. The **Dictionary** is displayed.

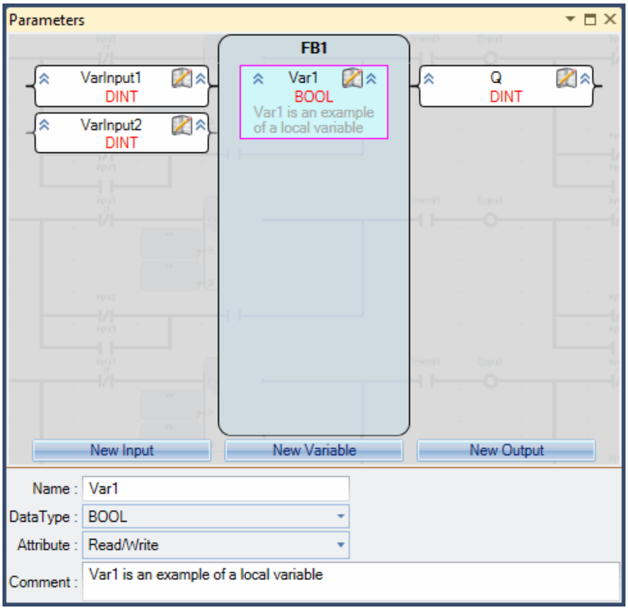



| Name | Equivalent | Comment |
|---|---|---|
| YES | TRUE | |
| PI | 3.14159 | |
| CANCEL | -2 | |



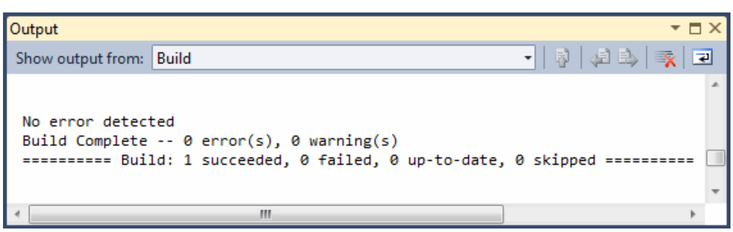

| Name | DataType | StringSi | Dimension | Wiring | Attribute | Direction | Modbus Add | IsRetain | InitialValue |
|---|---|---|---|---|---|---|---|---|---|
| Var1In | BOOL | | | | Read/Write | VarGlobal | | | TRUE |
| Var2In | BOOL | | | | Read/Write | VarGlobal | | | FALSE |
| Var3In | MESSAG | 255 | | | Read/Write | VarGlobal | | | HELLO |
| Var4In | DINT | | | | Read/Write | VarGlobal | | | 25 |
| Var5In | DINT | | | | Read/Write | VarGlobal | | | 2 |
| Var6In | REAL | | | | Read/Write | VarDirectlyRep | | | 30.7 |
| Var7In | REAL | | | | Read/Write | VarDirectlyRep | | | 6.0 |
| Var1Out | BOOL | | | | Read/Write | VarGlobal | | | |
| Var2Out | BOOL | | | | Read/Write | VarGlobal | | | |
| Var3Out | DINT | | | | Read/Write | VarGlobal | | | |

- To manage parameters and local variables for user-defined POUs, right-click the POU, and then click **Parameters**.

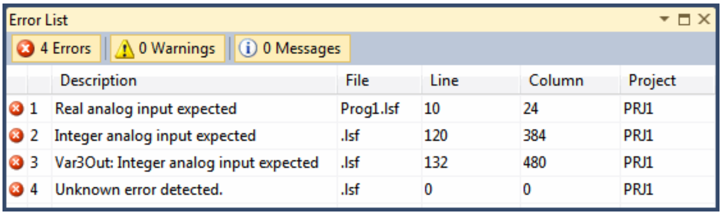



**8.** When debugging applications, you can oversee application performance using the following **ACP** features:

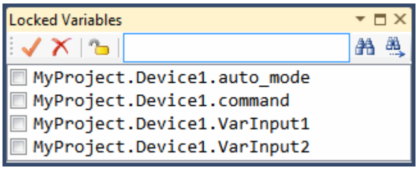
- To view the build information, from the View menu, click **Output**.



- To view the errors, warnings, and messages produced when editing and building programs, from the View menu, click **Error List**.

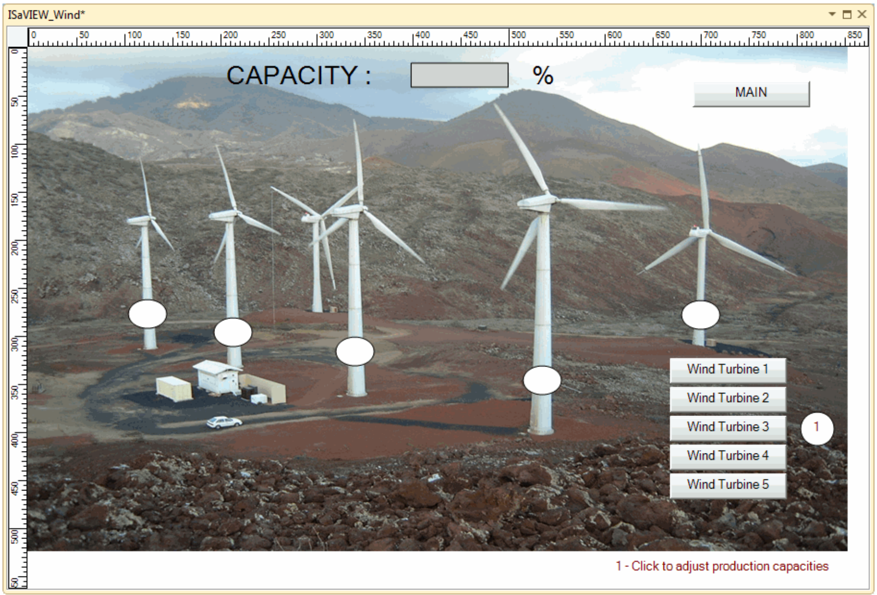



- To view or unlock locked variables while debugging, running online, and simulating, from the Debug menu, click **Locked Variables.**

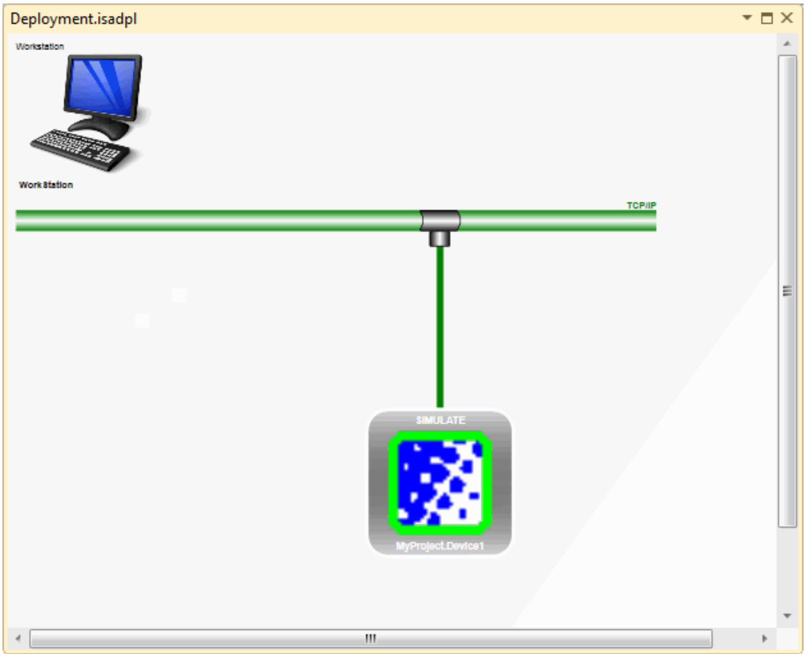



**9.** To add an ISaVIEW screen, right-click the device or program in the Solution Explorer, point to **Add**, and then click **New ISaVIEW.**

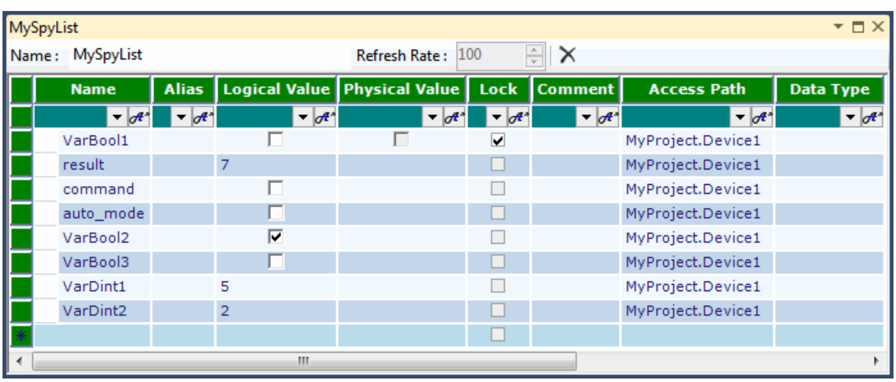
You can monitor or run control processes, locally or remotely, by creating ISaVIEW screens. You can define animation effects for the objects inserted in the ISaVIEW screens. Design mode enables editing the screen objects and animation mode executes the animation effects.




**10.** To graphically display the device, networks, and connections of a project, from the View menu, click **Deployment View**.

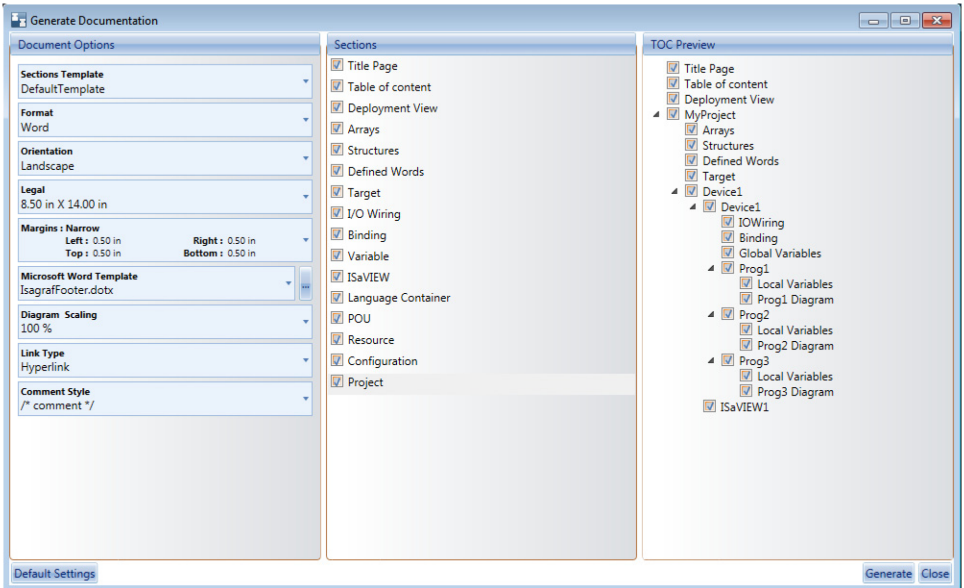



11. To view changes in the values of variables and function block instances, from the Debug menu, point to **Spy Lists,** then click the required spy list instance.



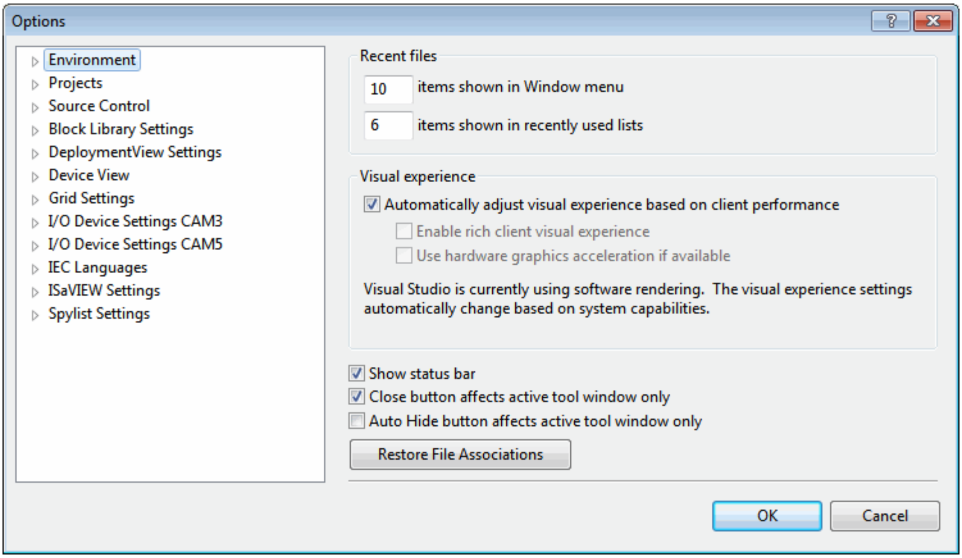

| Name | Alias | Logical Value | Physical Value | Lock | Comment | Access Path | Data Type |
|---|---|---|---|---|---|---|---|
| VarBool1 | | ☐ | ☐ | ☑ | | MyProject.Device1 | |
| result | | 7 | | ☐ | | MyProject.Device1 | |
| command | | ☐ | | ☐ | | MyProject.Device1 | |
| auto_mode | | ☐ | | ☐ | | MyProject.Device1 | |
| VarBool2 | | ☑ | | ☐ | | MyProject.Device1 | |
| VarBool3 | | ☐ | | ☐ | | MyProject.Device1 | |
| VarDint1 | | 5 | | ☐ | | MyProject.Device1 | |
| VarDint2 | | 2 | | ☐ | | MyProject.Device1 | |
| | | | | ☐ | | | |

12. To generate documentation for projects, devices, programs, and variables, from the File menu, click **Generate Documentation**.

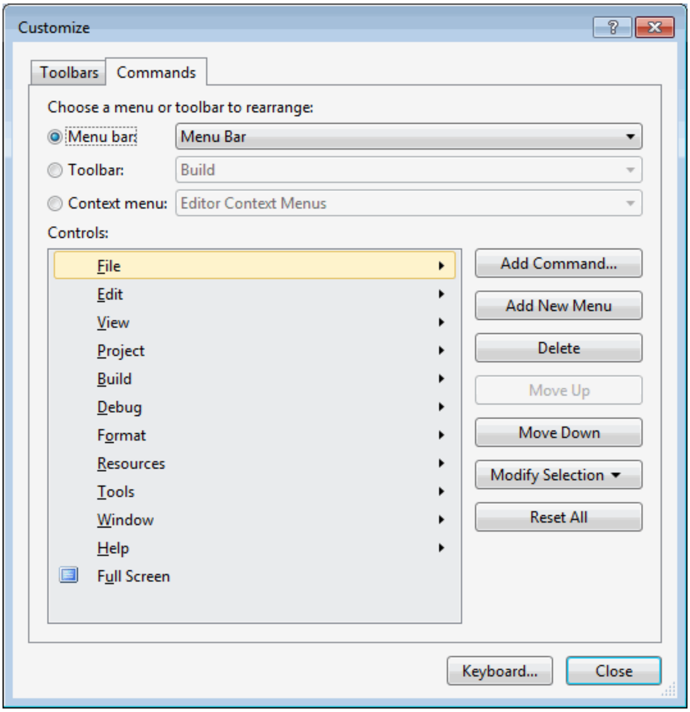



**13.** You can customize the Workbench using the following **ACP** features:

- To customize the environment, project, Source Control, Block Library, Deployment view, Device view, various grids, I/O device, IEC languages, ISaVIEW, and Spy List options, from the Tools menu, click **Options...**

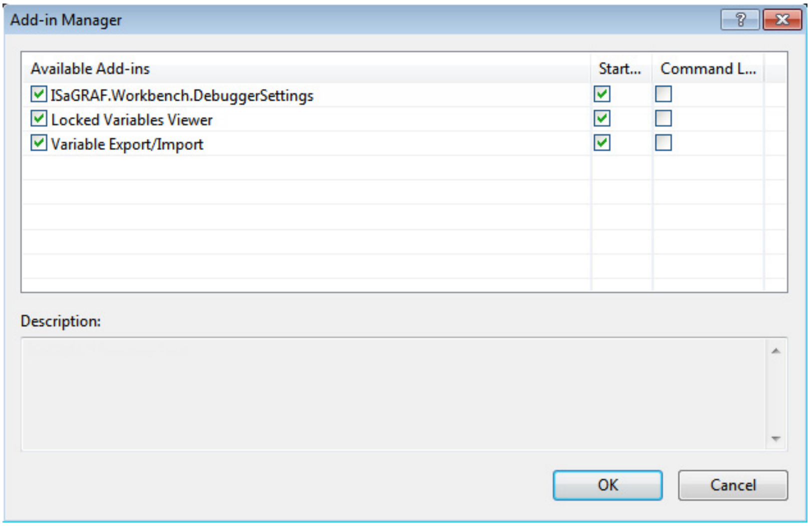


- To create or customize toolbars, menu bars, and context menus, from the Tools menu, click **Customize...**

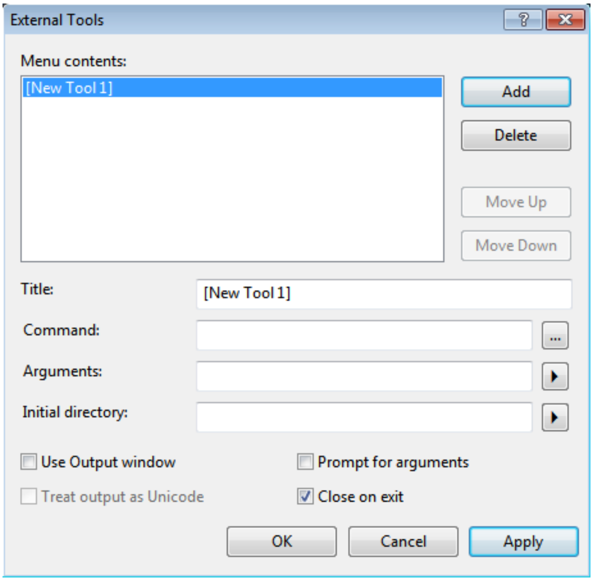



**14.** You can manage add-ins and external tools using the following **ACP** features:

- To manage registered add-ins, from the Tools menu, click **Add-in Manager...**




- To add external tools, from the Tools menu, click **External Tools...**



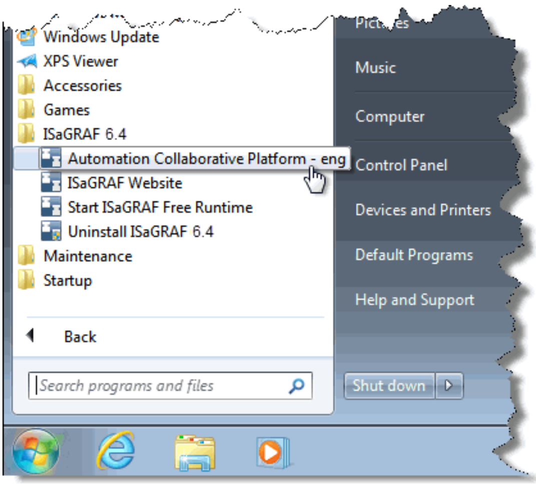
# Walking Through an Existing Application

This section describes a demo project included with the default installation.

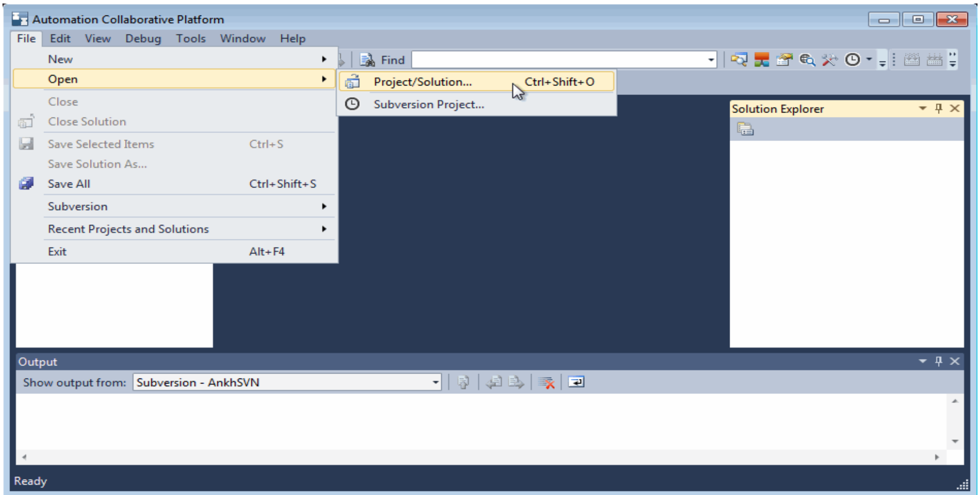
**To walk through an existing application**

1. Launch the **ACP** and open an existing application.

   a) From the Start menu, click **All Programs**, then **ISaGRAF 6.4**, and then click **Automation Collaborative Platform**.

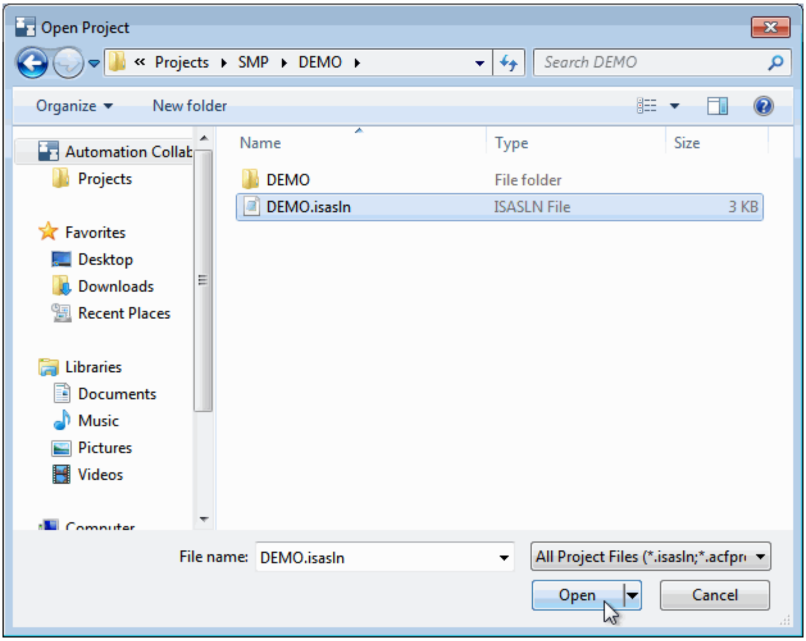



   b) From the File menu, point to **Open**, then click **Project/Solution...**.




**c)** In the Open Project dialog box, select and open the **DEMO.isasln** solution, located in the following directory:

%USERPROFILE%\My Documents\ISaGRAF 6.*x*\Projects\SMP\DEMO\DEMO.isasln

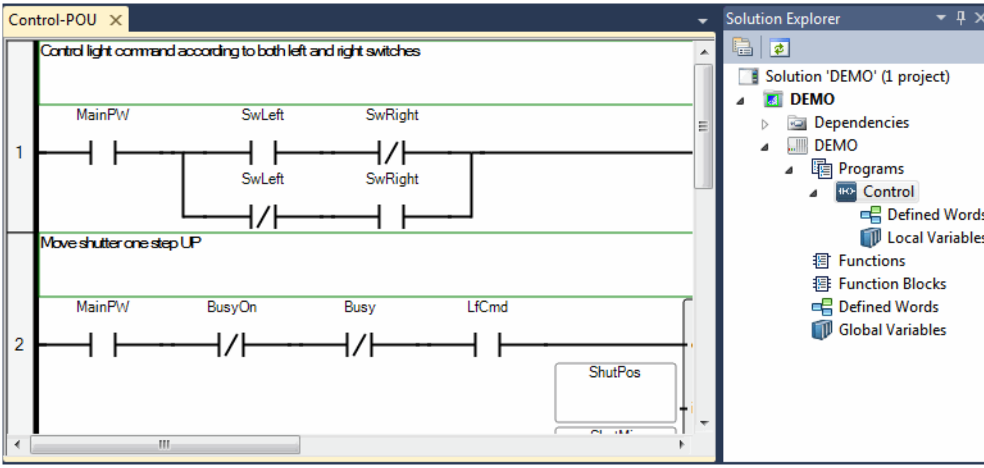



The DEMO project is displayed.

**2.** Review the application components.

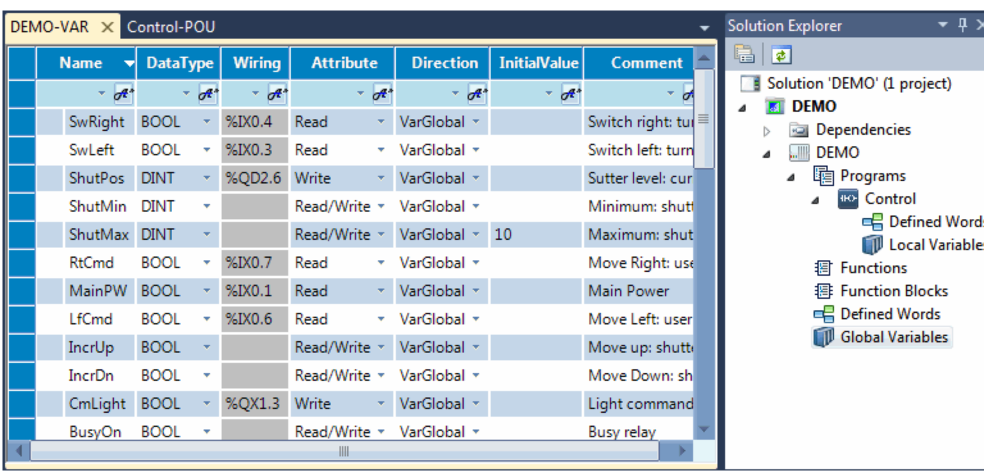
**a)** From the View menu, click **Solution Explorer**.

**b)** To view available programs, expand the project, device, and program elements, then view the programs by double-clicking the required program instance.



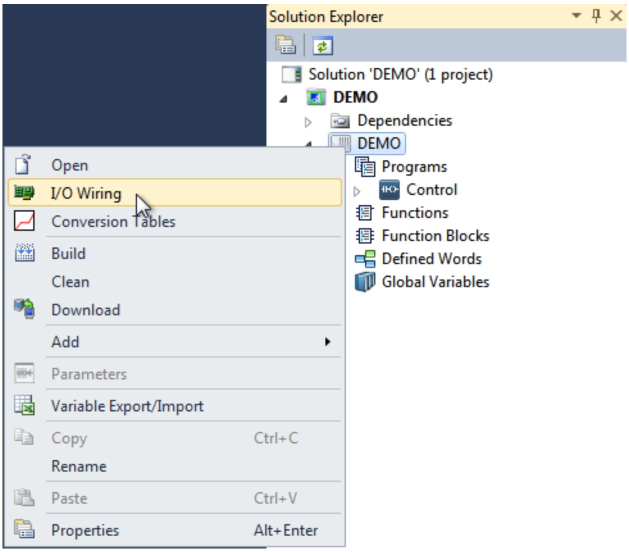

Opened programs are displayed in the language container.

**c)** To view the dictionary variables, in the Solution Explorer double-click **Global Variables**.

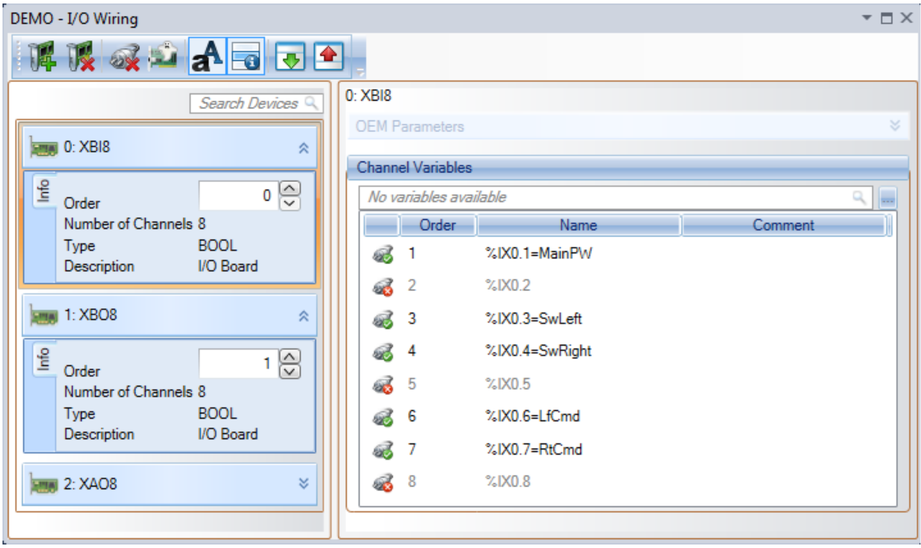



The dictionary is displayed in the workspace. You can add, edit, and remove variables. You can sort and filter the variables displayed, as well as arrange the columns to display.

**d)** To view the I/O variables connected to the channels of I/O devices, in the Solution Explorer, right-click the device, and then click **I/O Wiring**.

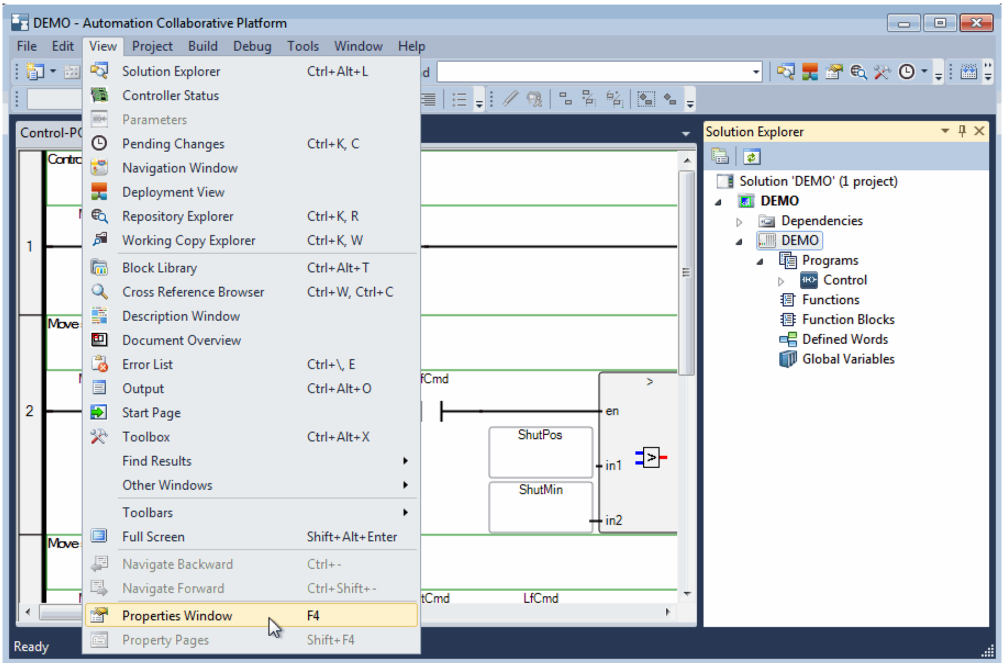



The I/O Wiring view is displayed. You can add I/O devices and set the real or virtual attribute. You can also wire or free the channels of an I/O device.

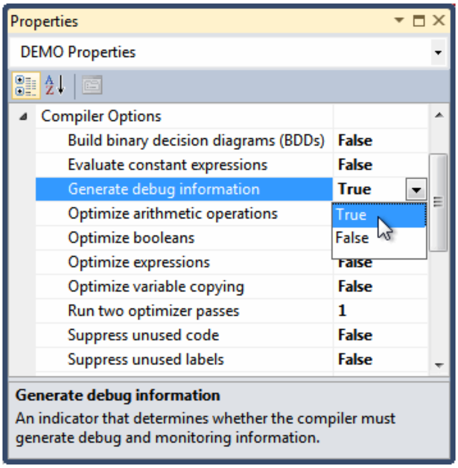

DEMO - I/O Wiring
Search Devices
0: XBI8
Info
Order 0
Number of Channels 8
Type BOOL
Description I/O Board
1: XBO8
Info
Order 1
Number of Channels 8
Type BOOL
Description I/O Board
2: XAO8
0: XBI8
OEM Parameters
Channel Variables
No variables available
Order
Name
Comment
1 %IX0.1=MainPW
2 %IX0.2
3 %IX0.3=SwLeft
4 %IX0.4=SwRight
5 %IX0.5
6 %IX0.6=LfCmd
7 %IX0.7=RtCmd
8 %IX0.8

**3.** For the device, set the properties for debugging.

**a)** From the View menu, click **Properties Window**.

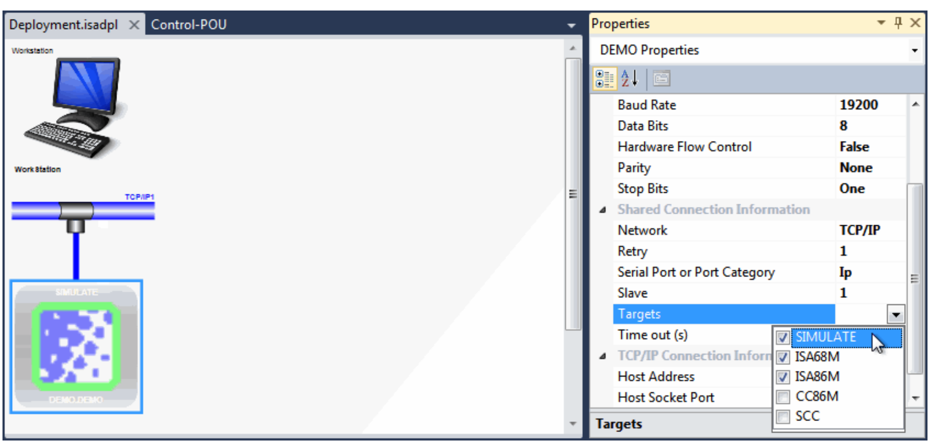



**b)** In the Solution Explorer, select the device, then from the Properties window, set *Generate debug information* to **True**.

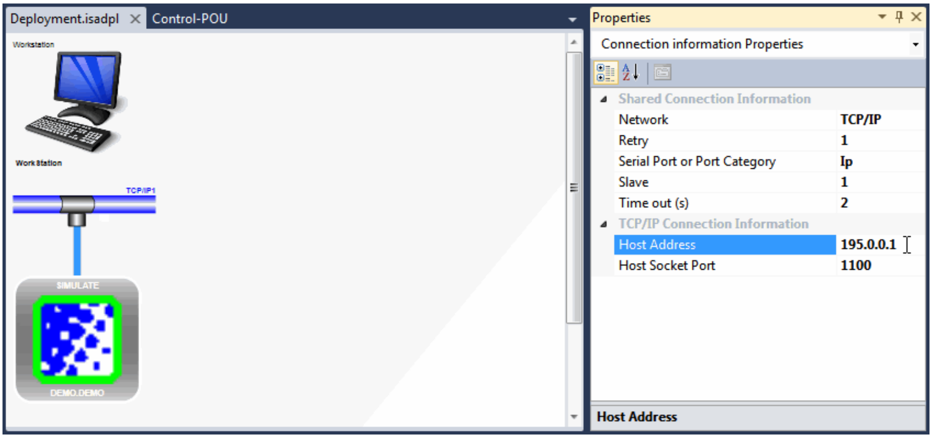



c) From the View menu, click **Deployment View**, then make the following modifications to the properties:

- In the Deployment view, select the target, then in the Properties window, for the *Target*s property, select up to three target types from the drop-down combo box.

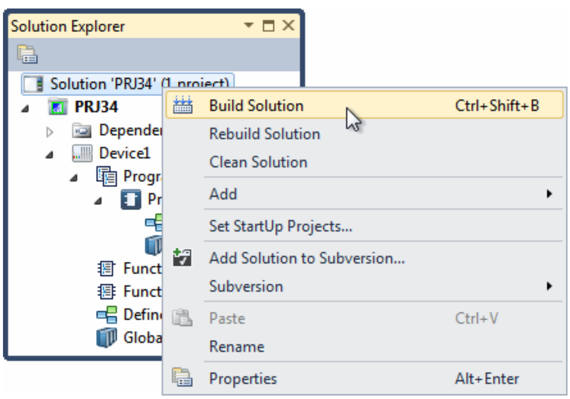



- In the Deployment view, select the connection between the target and the network, then in the Properties window, in the *Host Address* property, type the required IP address or socket host name.

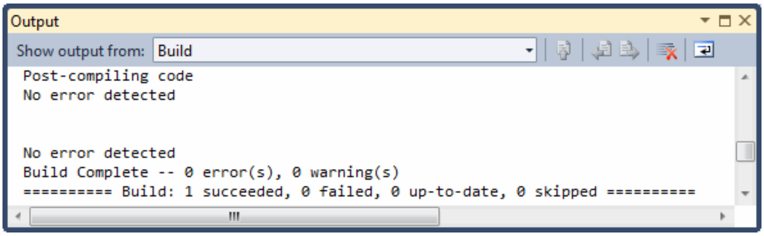



**4.** Build the solution, then view any generated errors, warnings, and messages.

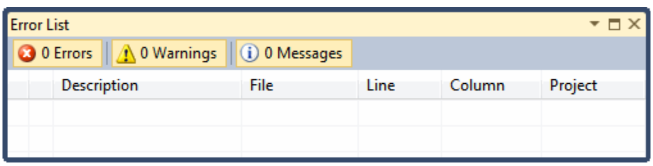
**a)** In the Solution Explorer, right-click the solution element, then click **Build Solution**.




**b)** To view the build information, from the View menu, click **Output**.




**c)** To view the errors, warnings, and messages generated during the build, from the View menu, click **Error List**.

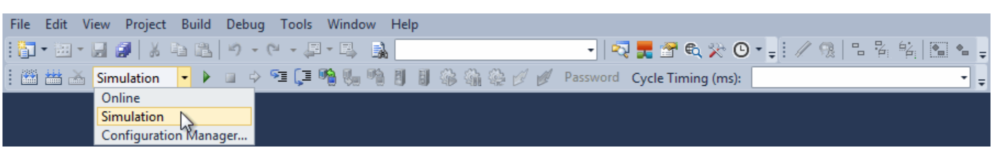



You can choose to display errors, warnings, or messages in the Error List. You can also sort the list of errors, warnings, and messages displayed.

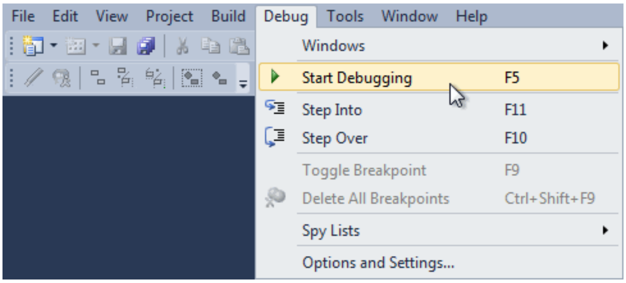
**5.** Debug the project.

You can simulate the running of an application without downloading code onto your target platform. However, when running an application online, you must download the project code onto the target before debugging.

**a)** In the Target Execution toolbar, from the Solution Configurations drop-down combo-box, select **Simulation**.




**b)** To begin the debugging process, from the Debug menu, click **Start Debugging**.

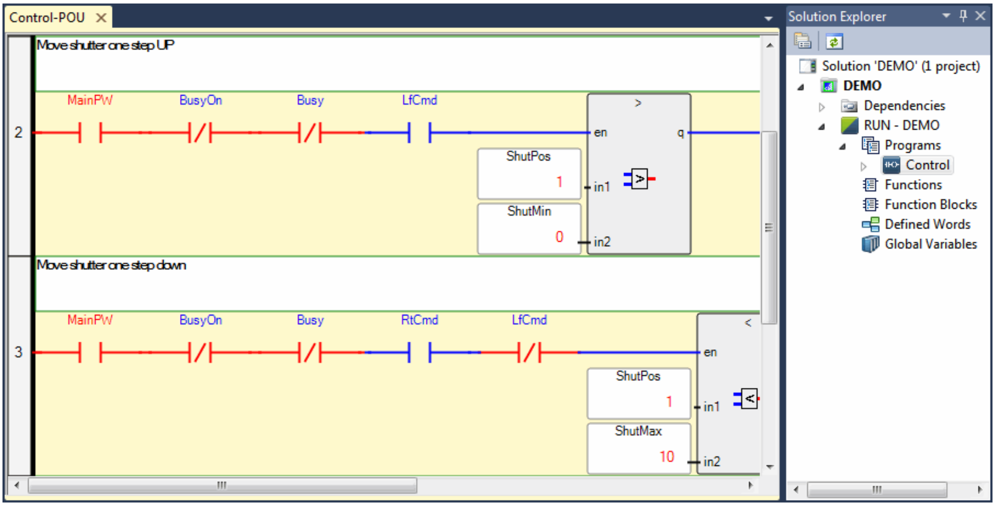



You can monitor the progress of the simulation using the Output window.

**6.** While in debug mode, view the programs and dictionary variables.

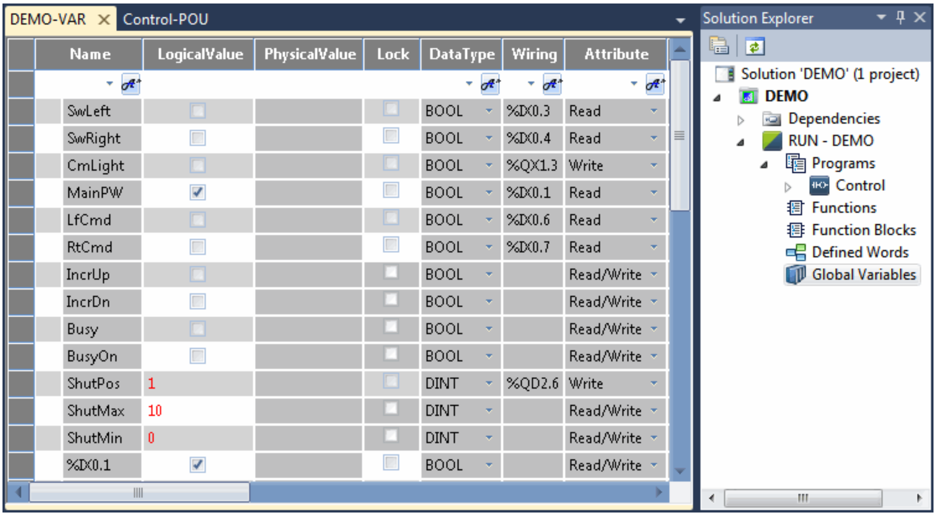
**a)** From the Solution Explorer, view the individual programs by double-clicking the required program instance.

The program is displayed in the language container. Boolean variables are displayed using color: red when True and blue when False. Numerical and textual values are displayed in red.



b) From the Solution Explorer, view the dictionary variables by double-clicking **Global Variables**.

The dictionary is displayed in the workspace. Note that the logical and physical values are displayed in red.




7. To stop the debugging process, from the Debug menu, click **Stop Debugging**.

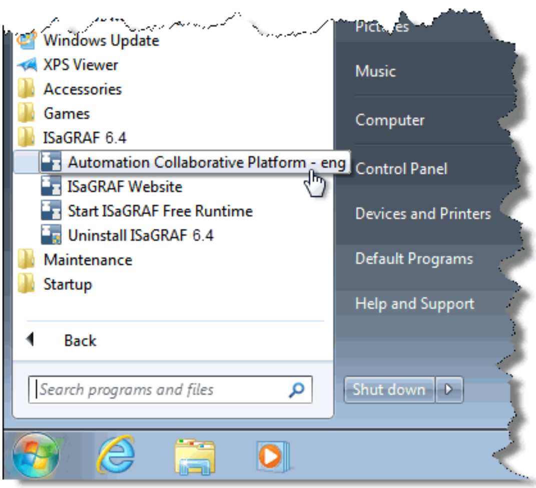
# Starting with a Basic Application

This section is a guideline to creating a basic solution and project by following the required steps. The project detailed in this section uses the ISaGRAF 3 Project template consisting of one device.

**To start a new project having one device**

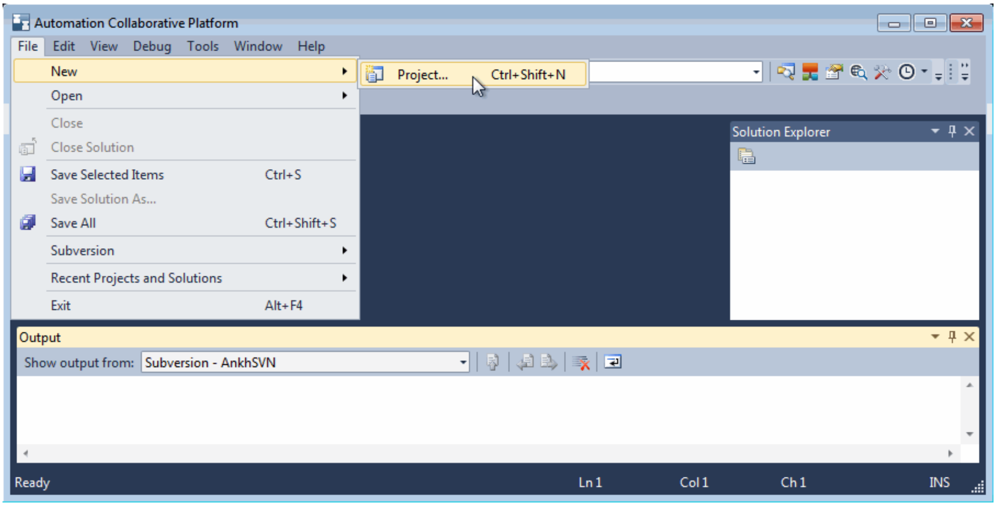
**1.** To launch the **ACP** and create a new solution, perform the following:

**a)** From the Start menu, click **All Programs**, then **ISaGRAF 6.4**, and then click **Automation Collaborative Platform**.

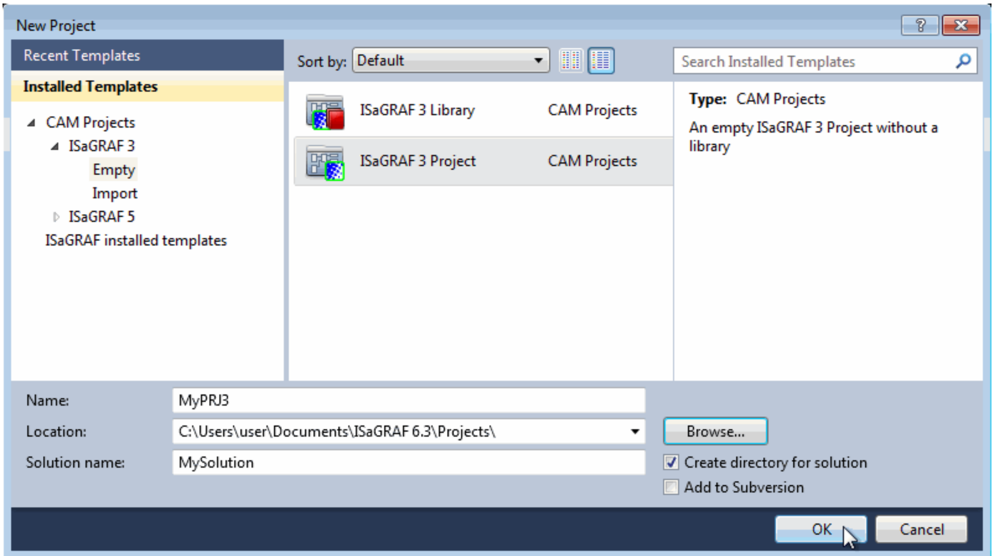



The Workbench is displayed.

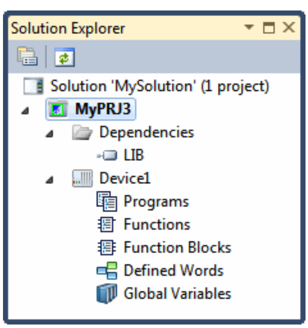
**b)** From the File menu, point to **New**, then click **Project...**




**c)** In the New Project dialog box, expand the **ISaGRAF 3** projects node, click the Empty template section, and then click the **ISaGRAF 3 Project** template. You then select *Create directory for solution* and specify a solution name. You must also specify a name and save location for the project, then click OK.

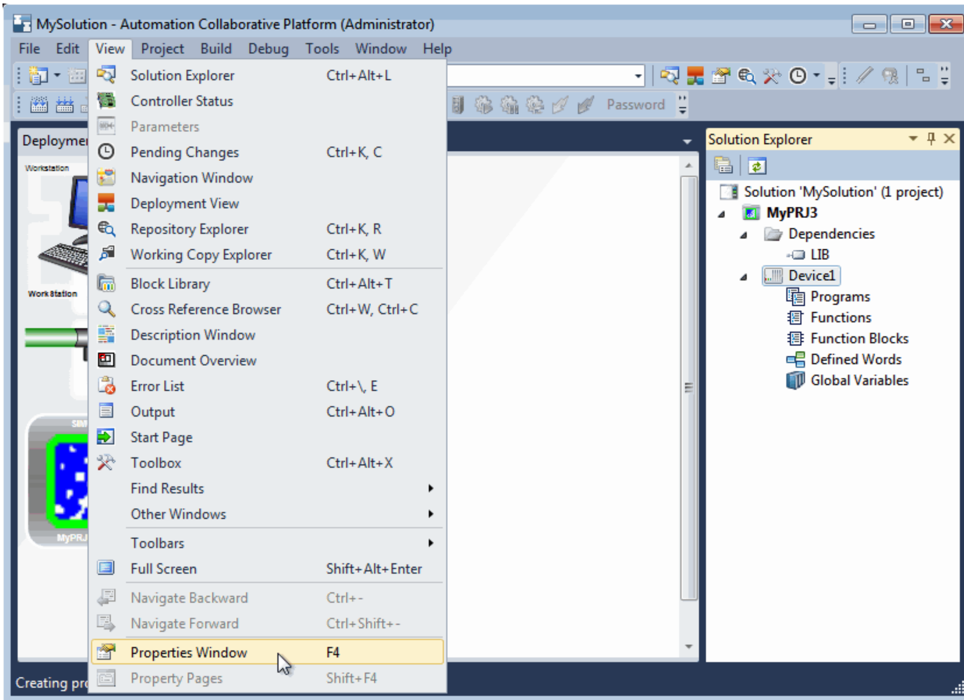



2. In the Solution Explorer, expand the project elements and note the device created from the **ISaGRAF 3 Project** template.

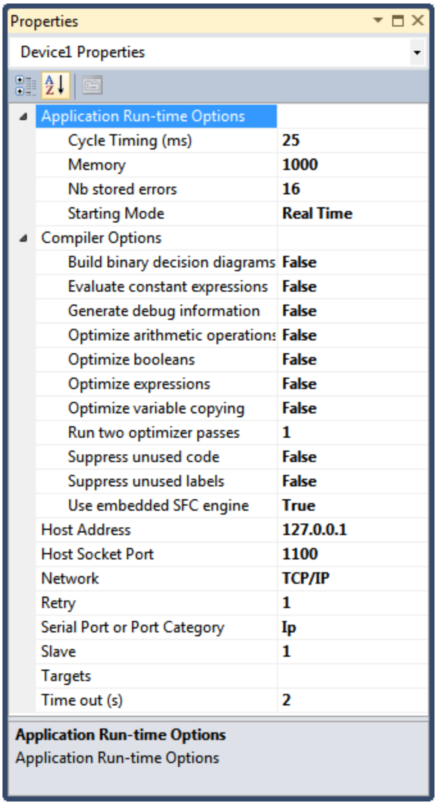



3. Specify the properties for the device.

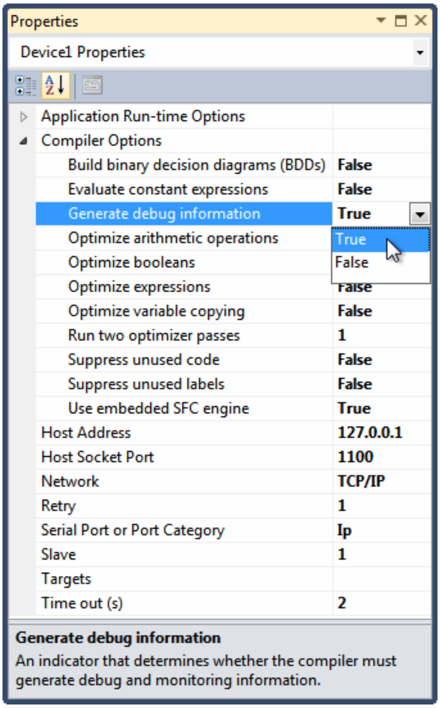
   a) In the Solution Explorer, select the device, then from the View menu, click **Properties Window**.


MySolution - Automation Collaborative Platform (Administrator)
File Edit View Project Build Debug Tools Window Help
Solution Explorer Ctrl+Alt+L
Controller Status
Parameters
Pending Changes Ctrl+K, C
Navigation Window
Deployment View
Repository Explorer Ctrl+K, R
Working Copy Explorer Ctrl+K, W
Block Library Ctrl+Alt+T
Cross Reference Browser Ctrl+W, Ctrl+C
Description Window
Document Overview
Error List Ctrl+\, E
Output Ctrl+Alt+O
Start Page
Toolbox Ctrl+Alt+X
Find Results
Other Windows
Toolbars
Full Screen Shift+Alt+Enter
Navigate Backward Ctrl+-
Navigate Forward Ctrl+Shift+-
Properties Window F4
Property Pages Shift+F4
Password
Solution Explorer
Solution 'MySolution' (1 project)
MyPRJ3
Dependencies
LIB
Device1
Programs
Functions
Function Blocks
Defined Words
Global Variables

**b)** In the Properties window, note the definitions for the device properties as well as the application run-time options and compiler options.

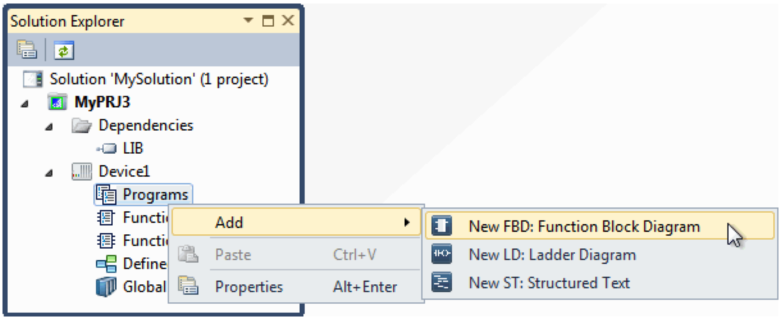



**c)** In the Properties window, expand the **Compiler Options** and set *Generate debug information* to **True**.



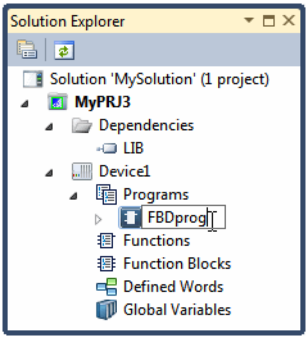

**4.** In the Solution Explorer, add a program and define the program name.

**a)** Right-click the program element, point to **Add**, then click the desired programming language.

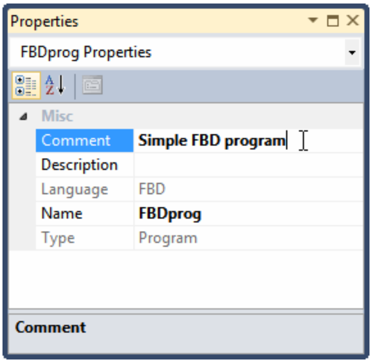


b) Right-click the added program, click **Rename**, then type the desired name in the space provided.

Program names can contain a maximum of eight characters.



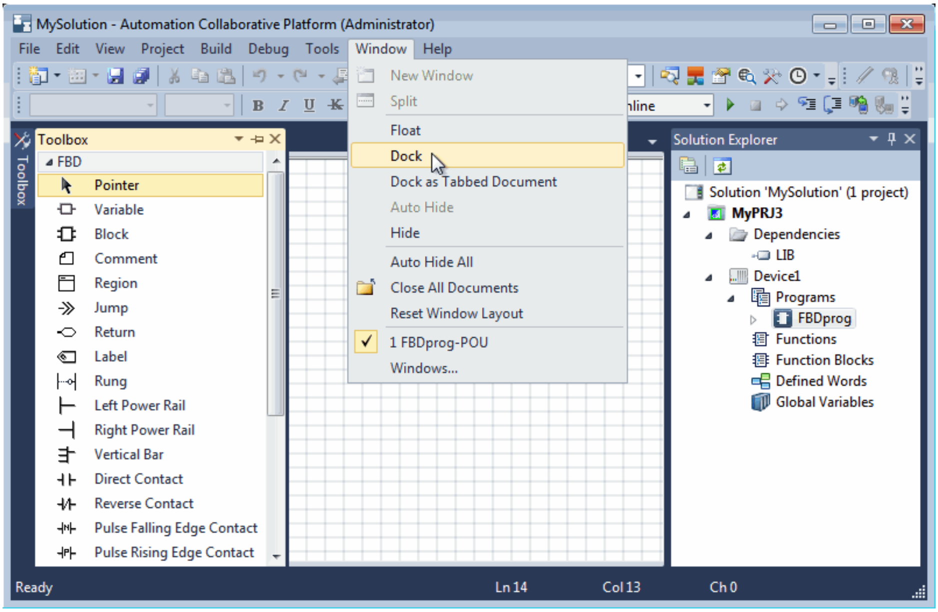

5. In the Properties window, define the *Comment* property for the program.




6. In the language container, add elements to the program.

a) From the Solution Explorer, double-click the program instance. The program is displayed in the language container. By default, the Toolbox is auto-hidden as a tab on the left edge of the Integrated Development Environment (IDE).

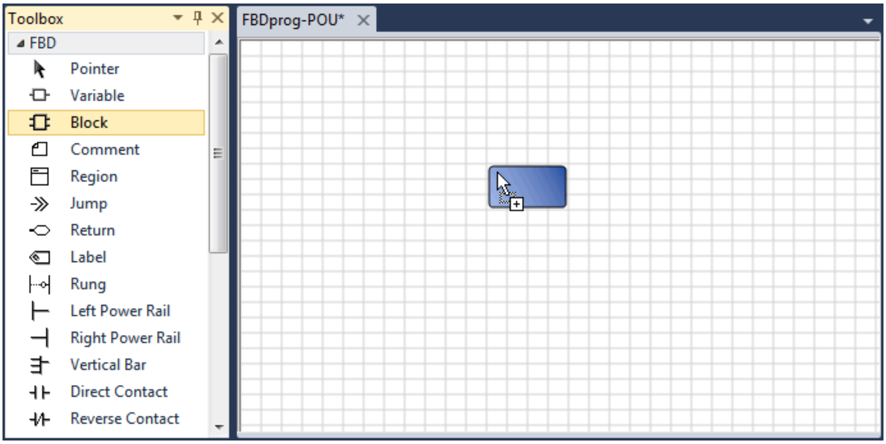
b) To display the Toolbox, click the tab so the Toolbox slides into view. From the Window menu, click **Dock**.




The Toolbox window is docked in the IDE.

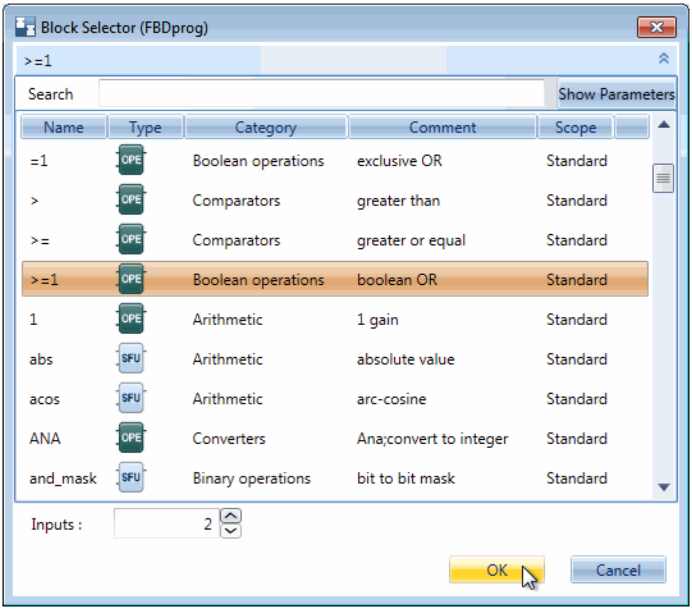
**c)** Add a block in the language container.

i) From the Toolbox, drag the **Block** element into the language container.




The Block Selector is displayed.

ii) In the block list, select the required POU, specify the number of inputs (when applicable), then click **OK**.



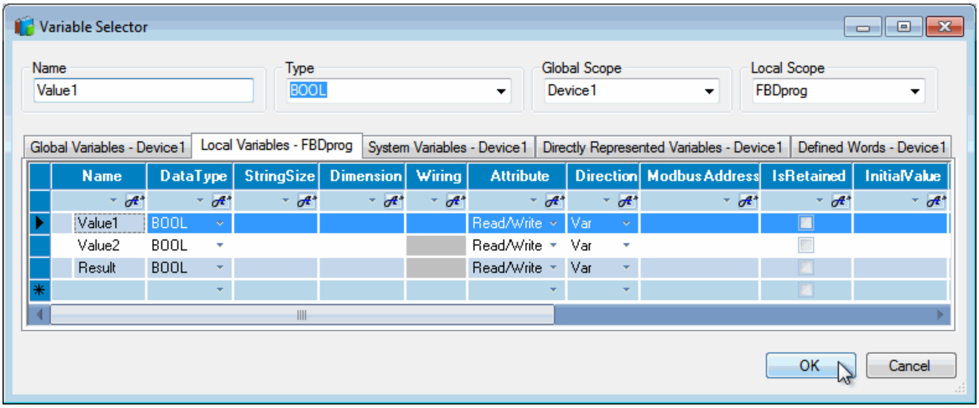

The block is displayed in the language container.

**d)** Add a variable in the language container.

i) From the Toolbox, drag the **Variable** element into the language container.

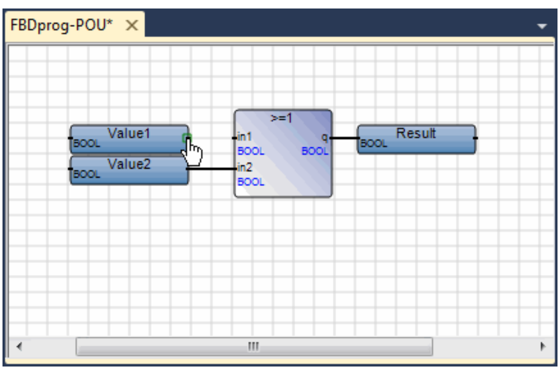
The Variable Selector is displayed, with tabs containing lists for *Global variable*s, *Local variables*, *System variables, Directly Represented Variables*, and *Defined Words*.

ii) In the *Local Variable* list, enter the variable name, data type, and other required information into the cells provided, then click **OK**.



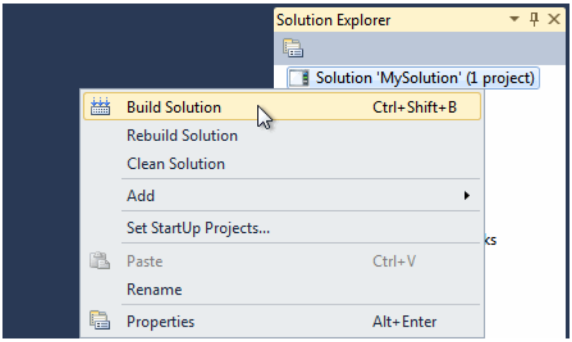

The variable is displayed in the language container.

**e)** Draw links from output to input points (in the direction of the data flow).

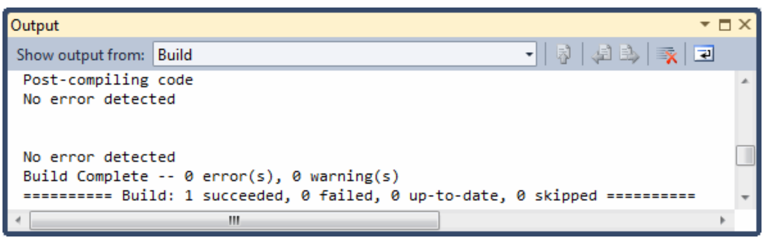



**7.** From the Solution Explorer, build the solution, then view any generated errors, warnings, and messages.

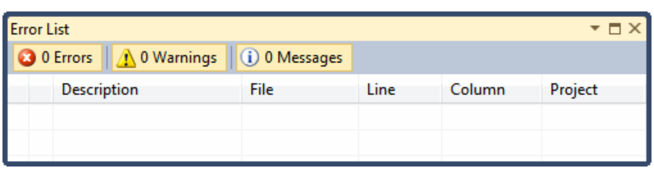
**a)** Right-click the solution element, then click **Build Solution**.




**b)** To view the build information, from the View menu, click **Output**.

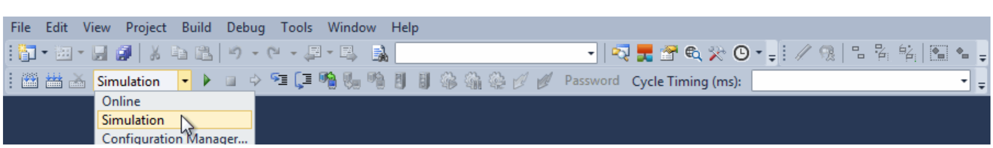



**c)** To view the errors, warnings, and messages generated during the build, from the View menu, click **Error List**.

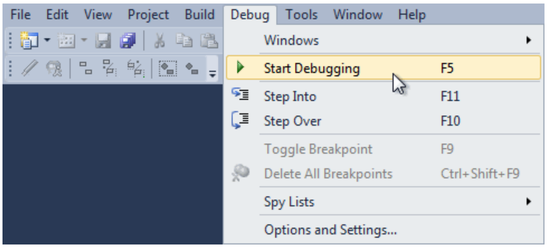



**8.** Begin the debugging process, then view the programs and dictionary variables.

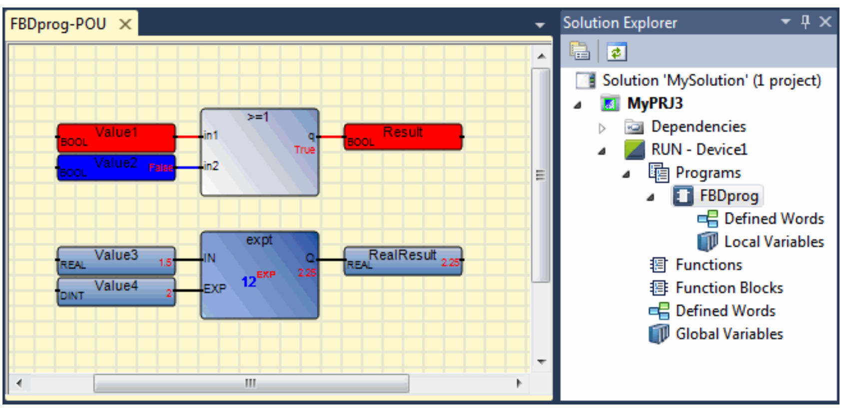
**a)** From the Target Execution toolbar, in the Solution Configuration drop-down combo-box, select **Simulation**.



**b)** From the Debug menu, click **Start Debugging**.

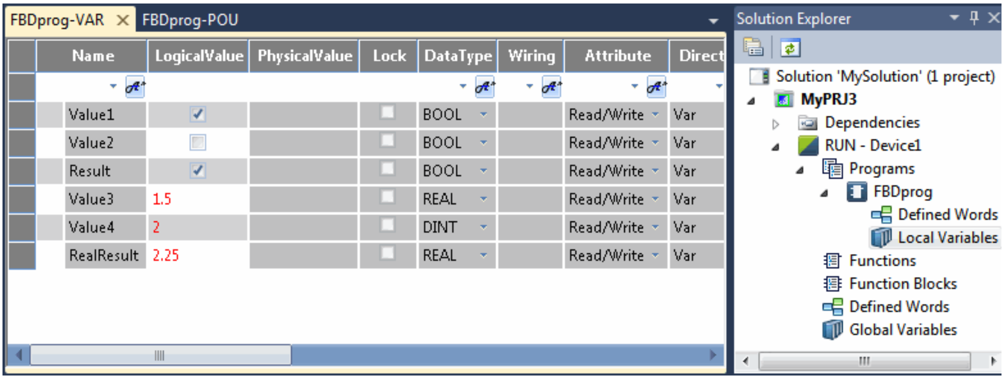



**c)** From the Solution Explorer, view the program by double-clicking the program element.




Note the debugging information regarding boolean variables is displayed using color: red when True and blue when False. Numerical and textual values are displayed in red.

**d)** From the Solution Explorer, view the dictionary variables by double-clicking **Local Variables** for the required program.




Note the logical values are displayed in red. Physical values are only displayed when running online.

**9.** To stop the debugging process, from the Debug menu, click **Stop Debugging**.

# Importing an Existing Application

When importing applications created with **ISaGRAF 3**, some features of your projects are converted for use in the current environment.

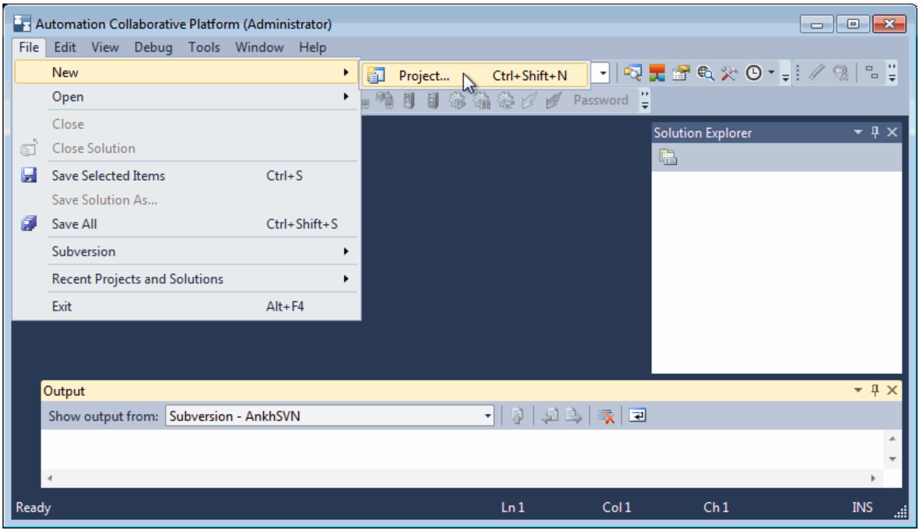
**Warning:** The **ISaGRAF 3 CAM** belonging to **ISaGRAF 6** does not support all the programming languages defined for **ISaGRAF 3**. The Sequential Function Chart (SFC), Flow Chart (FC), and Instruction List (IL) programming languages are not supported and therefore cannot be modified. However, projects containing these programming languages can still be imported and compiled.

### To import an ISaGRAF 3 project into ISaGRAF 6

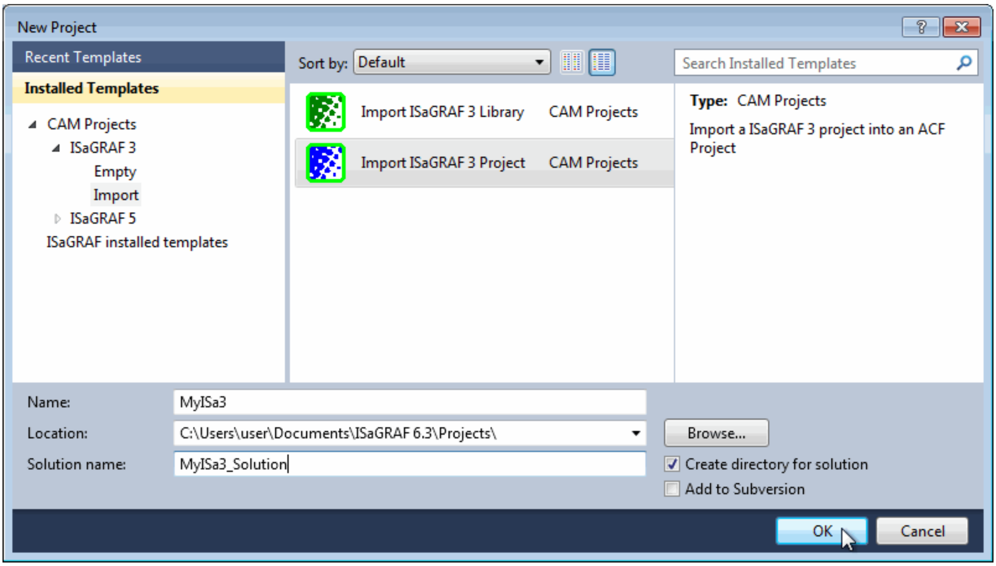
When importing **ISaGRAF 3** projects into **ISaGRAF 6**, the targets associated with the **ISaGRAF 3** projects must be supported by **ISaGRAF 6**.

**1.** Import the **ISaGRAF 3** project into **ISaGRAF 6**.

**a)** From the File menu, point to **New**, then click **Project**.

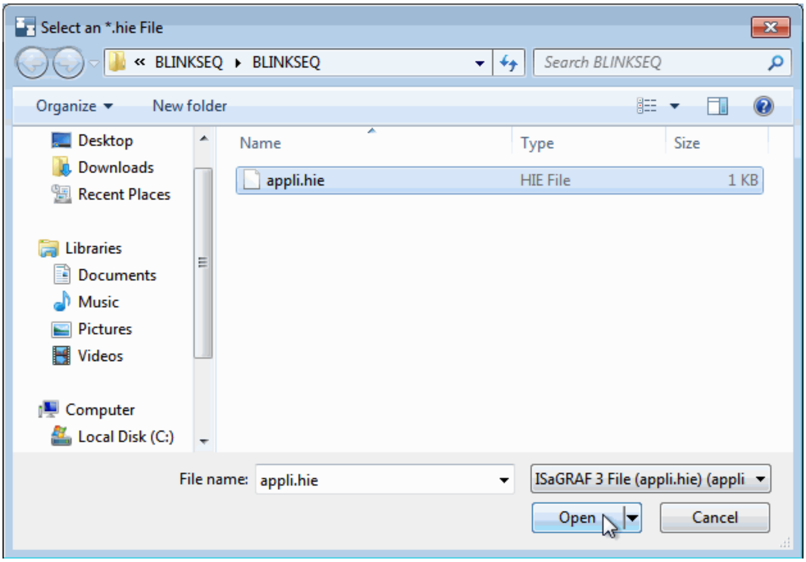


b) From the New Project dialog box, expand the **ISaGRAF 3** projects node, click the Import template section, and then click **Import ISaGRAF 3 Project**. You then enter the required information in the fields provided and click **OK**.




c) From the Select an .hie File dialog box, select the **ISaGRAF 3** project file, then click **Open**.

You may encounter a message asking if you want to update the database to the current version. To continue the importation process, click **OK**.

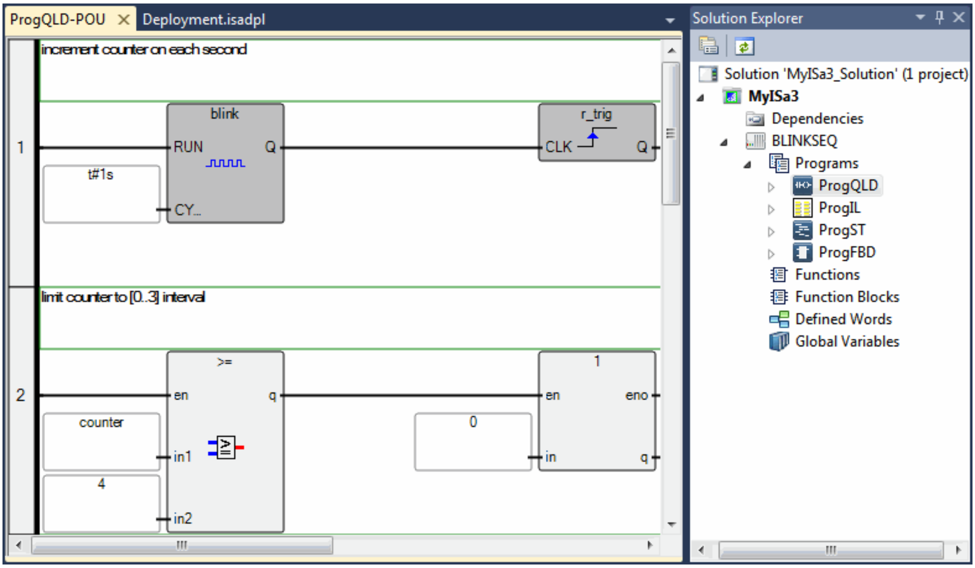



The **ISaGRAF 3** project is imported.

**2.** View the project in **ISaGRAF 6**.

**a)** In the Solution Explorer, expand the project, device, and program elements, then view the programs by double-clicking the required program instance.

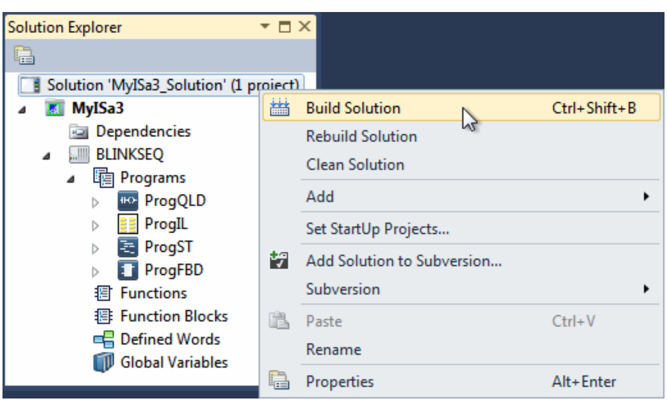
Opened programs are displayed in the language container.



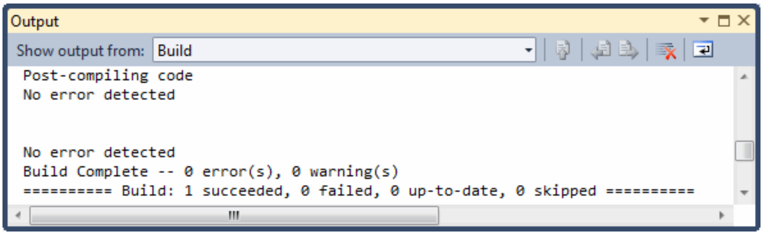

**Warning:** Since the IL programming language is not supported in **ISaGRAF 6**, the language editor cannot open the *ProgIL* program displayed in the previous example.

**3.** Build the solution, then view any generated errors, warnings, and messages.

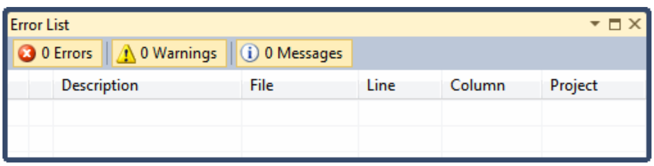
**a)** In the Solution Explorer, right-click the solution element, then click **Build Solution**.




**b)** To view the build information, from the View menu, click **Output**.

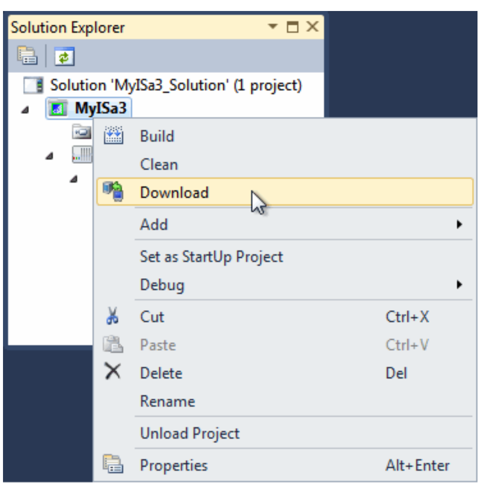



**c)** To view the errors, warnings, and messages generated during the build, from the View menu, click **Error List**.



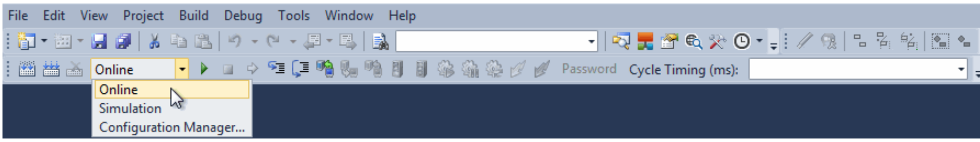

**4.** Debug the project.

**a)** To download the application code to the target, in the Solution Explorer, right-click the project element, then click **Download**.

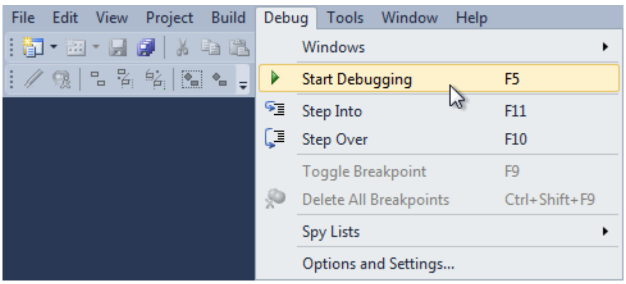



You can monitor the progress of the download operation using the Output window.

**b)** In the Target Execution toolbar, from the drop-down combo-box, select **Online**.




**c)** To begin the debugging process, from the Debug menu, click **Start Debugging**.




**5.** To stop the debugging process, from the Debug menu, click **Stop Debugging**.

# Dictionary

The Dictionary, i.e., tag editor, is the environment where you manage variables and defined words. The Dictionary is made up of multiple grids having different purposes.

- Defined Words Grid, enables managing the defined words for a project
- Variables Grid, enables managing the variables for devices and programs. Each device and program has its instance of the grid. For devices, the grid displays global variables. For programs, the grid displays local variables.

The grids each display the properties for the type of element. You can open multiple grid instances simultaneously. When working in a grid, you can navigate the cells using the mouse controls. For complex data types, you can expand fields using Ctrl+PLUS SIGN on numeric keypad (+) and collapse fields using Ctrl+MINUS SIGN on numeric keypad (-).

You access Dictionary grids from the Solution Explorer.

You can customize the Dictionary environment by arranging the columns to display and setting the display colors.

**To access a Dictionary grid instance**

1. From the Solution Explorer, expand the project and device nodes.
2. For the variables of a device, expand the required device node, then double-click the **Dictionary** element.

   The Dictionary instance is displayed containing the variables belonging to the device.
3. For the variables of a program, expand the required program node, then double-click the **Dictionary** element.

   The Dictionary instance is displayed containing the variables belonging to the program.
4. For the defined words of a project, double-click the **Defined Words** node.

   The Defined Words grid is displayed.

**To arrange the columns to display**

To retain customized display settings, you must save the Dictionary instance before closing.

1. To move a column, drag the column header to another location.

   When dragging a column header, arrows indicate the current position of the header.

2. To hide a column, right-click a column header, and then click **Hide Column**.

3. To show a column, right-click any column header, point to **Show Column**, and then click the required column name.

## Defined Words Grid

The Defined Words grid of the Dictionary enables managing the defined words for a project. You can perform the following tasks from the defined words grid:

- Creating defined words
- Editing existing defined words
- Deleting defined words
- Sorting defined words in the grid
- Filtering defined words in the grid

For defined words, the properties are the following:

| Column | Description | Possible Values |
|---|---|---|
| Name | Name of the defined word | Limited to 32 characters beginning with a letter followed by letters, digits, and underscores. Defined words cannot contain defined words. |
| Equivalent | String replacing the defined word during compilation. For example, the defined word "PI" is replaced by its equivalent "3.14159" | Limited to 255 characters |
| Comment | Comment for the defined word | Free-format text |

You can customize the Dictionary environment by arranging the columns to display.

**To create a defined word**

1. From the Solution Explorer, double-click the Defined Words node for the project.
2. In the Defined Words grid, define the required properties, then press ENTER.

### To edit an existing defined word

1. From the Solution Explorer, double-click the Defined Words node for the project.

2. In the Defined Words grid, make the required changes.

### To delete a defined word

You can delete defined words from the Defined Words grid.

1. From the Solution Explorer, double-click the Defined Words node for the project.

2. In the Defined Words grid, right-click the defined word to delete, and then click **Delete**.

### To sort defined words in the grid

You can sort the defined words in the grid using an ascending or descending order for the individual columns.

1. From the Solution Explorer, double-click the Defined Words node for the project.

2. In the Defined Words grid, select the required column header.

   An arrow showing the current order is displayed on the column header.

3. Toggle the column header to switch between ascending and descending order.

### To filter defined words in the grid

You can filter defined words in Defined Words grid. When filtering, you create a view displaying only the defined words containing specified characters.

The filter row is the top row of the grid. You can filter defined words by typing alphabetical and numerical characters in the cells of the filter row. You can also select from the drop-down-combo box. Matching defined words are automatically displayed.

1. From the Solution Explorer, double-click the Defined Words node for the project.

2. In the filter row of the Defined Words grid, click the required cell, then do one of the following:

- Type the characters to use in the filtering operation
- Select the required defined word from the drop-down combo-box

**See Also**

Dictionary

## Variables Grid

The variables grid of the Dictionary enables managing the variables for a device or program. Each device and program has its instance of the grid. For devices, the grid displays global variables. For programs, the grid displays the local variables. You can perform the following tasks from the variables grid:

- Creating variables
- Editing existing variables
- Dragging variables
- Deleting variables
- Sorting variables in the grid
- Filtering variables in the grid

For variables of devices or programs, the properties are the following:

| Column | Description | Possible Values |
|---|---|---|
| Name | Name of the variable | Limited to 32 characters beginning with a letter followed by letters, digits, and single underscore characters. These names cannot have two consecutive underscore characters. |
| Logical Value | Available while running online, monitoring, and simulating applications. Displays the value used by code being executed on the virtual machine. You can force the value of variables. | Values are displayed according to the variable data type |
| Physical Value | Available while running online and monitoring applications. Displays the value sent to and received from the drivers. You can force the value of variables. | Values are displayed according to the variable data type |

| Column | Description | Possible Values |
|---|---|---|
| Lock | Available while running online, monitoring, and simulating applications. The indication of whether the value of the variable is locked. Locking operates differently for simple variables, array elements, and function block parameters. For simple variables, individual variables are locked directly. For array elements, locking an element locks all elements of the array. | Yes or No |
| Data Type | Data type of the variable | BOOL, DINT, REAL, TIME, MESSAGE. To create a one-dimensional array, specify a dimension. |
| String Size | For String type variables, indicates the maximum length | String capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string |
| Dimension | The size (number of elements) of an array. | Arrays are only available for the BOOL, DINT, REAL, and TIME data types; these are not available for the MESSAGE type. Arrays can have a maximum of 255 elements. For example, [6] represents a one-dimensional array containing elements from 0 to 5. |
| Wiring | Read-only cell, generated by the I/O wiring tool indicating the I/O channel to which the variable is wired | Uses the syntax of Directly Represented Variables |

| Column | Description | Possible Values |
|---|---|---|
| Attribute | The property of a variable indicating its read and write access rights. | Read, Write, or Read/Write |
| Direction | For I/O wiring, indicates whether a variable is an input, output, or internal. | VarInput, VarOutput, or Var |
| Modbus Address | Modbus address of the variable | Possible variables are Var direction local to programs and functions, or global variables; unavailable variables are input, output, directly represented, and local to function blocks as well as arrays. The format is four hexadecimal digits ranging from 0001 to FFFF. |
| Retained | The indication of whether the value of the variable is saved by the virtual machine at each cycle. For details on retaining, i.e., backing up, variables, refer to the SYSTEM operator. | Yes or No |
| Initial Value | Value held by a variable when the virtual machine starts the execution of the device code | The initial value of a variable can be the default value, a value given by the user when the variable is defined or the value of the retain variable after the virtual machine has stopped. |
| Unit | User-defined text indicating the unit of measure of the logical and physical values | Free format |
| Comment | User-defined text | Free format |

You can customize the Dictionary environment b y arranging the columns to display.

**To create a variable**

1. From the Solution Explorer, access the Dictionary instance for the required device or program.

2. In an empty row of the variables grid, define the required properties for the variable, then press ENTER.

**To edit an existing variable**

1. From the Solution Explorer, access the Dictionary instance for the required device or program.

2. In the variables grid, make the required changes.

**To drag a variable**

You can drag variables from a Dictionary instance to multiple locations within a project. These locations include other Dictionary instances as well as elements within a language container.

You drag variables to other locations individually. When dragging a variable to another Dictionary instance, you can place the variable anywhere in the grid. When dragging a variable into a language container, you can place the variable anywhere in the language container. To retain changes made to Dictionary instances and language containers, save the respective instance or POU before closing.

1. From the Solution Explorer, access the Dictionary instance containing the required variable and the destination for the variable.

2. From the Dictionary instance containing the required variable, in the variables grid, select the variable by clicking the cell in the left-most column.

   The selection indicator () is displayed in the leftmost column.

3. Drag , placing the variable in the grid or open language container.

   The variable is displayed at the destination.

**To delete a variable**

You can delete variables from Dictionary instances. Deleting variables from an instance opened for a program element removes the variables from the instance only

1. From the Solution Explorer, access the Dictionary instance for the required device or program.

2. In the variables grid, right-click the variable to delete, then click **Delete**.

**To sort variables in the grid**

You can sort the variables in the grid using an ascending or descending order for the individual columns.

1. From the Solution Explorer, access the Dictionary instance for the required device or program.

2. In the variables grid, select the required column header.

   An arrow showing the current order is displayed on the column header.

3. Toggle the column header to switch between ascending and descending order.

**To filter variables in the grid**

You can filter variables in variables grid instances. When filtering, you create a view displaying only the variables containing specified characters.

The filter row is the top row of the grid. You can filter variables by typing alphabetical and numerical characters in the cells of the filter row.You can also select from the drop-down-combo box. Matching variables are automatically displayed.

1. From the Solution Explorer, access the Dictionary instance for the required device or program.

2. In the filter row of the variables grid, click the required cell, then do one of the following:
   - Type the characters to use in the filtering operation
   - Select the required defined word from the drop-down combo-box

### See Also

Dictionary

# Device View

The device view is a graphical environment enabling navigation through project elements such as POUs. The navigation consists of vertical links on the left pane and a breadcrumbs trail in the address field. For a device, you can access the following information:

- The programs defined in a device. You can open individual programs by double-clicking the required instance. You can also view the local defined words.
- The user-defined functions and function blocks defined in a device. You can view the parameters by single-clicking the instance or open the POU by double-clicking the instance. You can also view the local defined words.
- Other elements attached to the device including ISaVIEW screens and global defined words.

**To access the Device View**

- In the Solution Explorer, right-click the required device, and then click **Open**.

  The device view is displayed in the workspace.

**To display device elements**

1. For programs, click  on the **Programs** item, then perform the following:
   - To open a program in the language container, double-click the required program instance.
   - To display the local defined words for a program, expand the arrow beside the required program instance, then double-click **Defined Words**.
2. For functions, click  on the **Functions** item, then perform the following:

- To display the Parameters view for a function, click the required function instance.
- To open a function in the language container, double-click the function instance.
- To display the local defined words for a function, expand the arrow beside the required function instance, then double-click **Defined Words**.

**3.** For function blocks, click  on the **Function Blocks** item, then perform the following:

- To display the Parameters view for a function block, click the function block instance.
- To open a function block in the language container, double-click the function block instance.
- To display the local defined words for a function block, expand the arrow beside the required function block instance, then double-click **Defined Words**.

**4.** To display global defined words, click  on the **Others** item, then double-click **Defined Words**.

**5.** To display ISaVIEW screens, click  on the **Others** item, then double-click the required ISaVIEW instance.

# I/O Wiring

I/O wiring enables the definition of connections between variables defined for a project and channels of complex equipment or I/O boards existing on a target system. Complex equipment and I/O boards are available for use in a project when these are defined in a library to which a dependency exists.

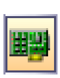
The I/O wiring instance for a device represents a hardware rack having multiple slots for complex equipment and I/O boards. A rack can contain up to 255 boards where each board can have up to 128 I/O channels. The total number of single I/O boards (including single equipment and boards of complex equipment) cannot exceed 255.

The I/O wiring view consists of two sections:

- A rack list, displaying defined complex equipment and I/O boards in the slots. An order number identifies each slot. Expanding the equipment accesses information and single devices.
- A channel variables list, enabling the association of channels with variables. This list displays the name of all variables. When online, the channel variables list also displays the logical value, physical value, and lock status of all variables.

You define connections from the I/O Wiring view where you add complex equipment and I/O boards, then wire the channels to variables. When defining I/O wiring for the first time, a device instance is empty.

The I/O Wiring toolbar enables performing many tasks in device instances:



Adding complex equipment and I/O boards to the rack list

Deleting complex equipment and I/O boards from the rack list

Freeing all channels of a complex equipment or I/O board

Toggles a complex equipment or I/O board between real and virtual

| | |
|---|---|
|  | Displays the complete names of equipment |
|  | Displays the empty slots in the rack list |
|  | Expands all complex equipment and I/O boards to display their information |
|  | Reduces all complex equipment and I/O boards to hide their information |

**To define connections between complex equipment or I/O board channels and variables**

1. From the Solution Explorer, right-click a device, then click **I/O Wiring**.
2. Add complex equipment and I/O boards to the rack list.
3. Select the individual I/O boards and connect the individual channels to the required variables, in the channel variables list. To display the channels for complex equipment, access the individual simple devices by clicking the Devices tab.

**See Also**

I/O Devices
I/O Channels

# I/O Devices

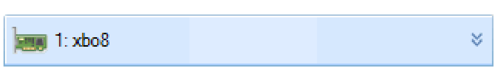
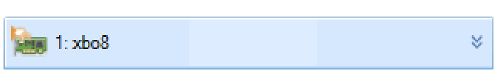
An I/O device represents a complex equipment or an I/O board. Individual single equipment and I/O boards can have up to 128 I/O channels. An I/O device contains channels having the same data type and direction.

When adding I/O devices, the Device Selector enables selecting from complex equipment and I/O boards available from a library to which a dependency exists. Each device is automatically assigned a device order number ranging from 0 to 254 and has a defined number of channels. You can include a comment.

While running online, when devices are set to real, I/O variables are directly linked to the corresponding I/O devices. Input or output operations in the programs correspond directly to the input or output conditions of the actual I/O device fields. When devices are set to virtual, I/O variables are processed and updated in memory. The debugger can read or update these to enable simulating I/O processing, but no actual connection is made.

When adding complex devices, the number of channels, i.e., device size, of individual simple devices making up a complex device varies depending on the definition of the complex device in the library.

You manage I/O devices from the rack list containing the following types:

| | |
|---|---|
|  | Real complex equipment |
|  | Virtual complex equipment (indicated by the flag) |
|  | Real I/O board |
|  | Virtual I/O board (indicated by the flag) |

When selecting I/O devices, you can access their properties by expanding individual devices.

From the I/O Wiring view, you can perform the following tasks when managing I/O devices:

- Adding I/O devices
- Freeing the channels of I/O devices
- Toggling I/O devices between real and virtual
- Accessing simple devices of complex equipment
- Displaying I/O device information
- Deleting I/O devices

**To add an I/O device**

I/O devices are available for use in a project when these are defined in a library to which a dependency exists.

**1.** On the I/O Wiring toolbar, click .

**2.** In the Device Selector, select the required I/O device from the list of available devices.

The device order number and number of channels is defined for the I/O device in the library. You can add a comment for the I/O device.

**To free the channels of an I/O device**

**1.** From the rack list, select the I/O device for which to free all channels.

**2.** From the I/O Wiring toolbar, click  .

**To toggle the real/virtual attribute**

You can toggle between the real and virtual attribute for a selected I/O device. Virtual I/O devices are displayed with a red star.

**1.** From the rack list, select the I/O device for which to change the attribute.

2. From the I/O Wiring toolbar, click .

**To access simple devices of complex equipment**

1. From the rack list, expand the required I/O device by clicking ⊟ .
2. To view simple devices of a complex equipment, click **Devices**.

**To display I/O device information**

You can toggle between displaying and hiding I/O device information.

1. From the rack list, expand the required I/O device by clicking ⊟ .
2. To view information about the device, click **Info**.

**To delete an I/O device**

You can delete devices. When deleting devices, all variables are unwired from the device.

1. From the rack list, select the I/O device to delete.
2. From the I/O Wiring toolbar, click  .

The device is removed from the rack list.

## See Also

I/O Wiring

# I/O Channels

I/O channels represent hardware I/O points. These can be inputs or outputs. A variable is generally connected to a channel to be used in POUs. Directly represented variables can also be used in POUs. When adding I/O devices, the number of channels is defined for the device. All I/O channels of a device have the same type and direction.

You wire variables to channels of an I/O device in the channel variables list. In this list, the displayed variable names are their direct representations.

You can use direct variable representation (%IX1.1) to access I/O values when I/O channels have no wiring.

An unwired channel is represented in the Dictionary as a directly represented variable under the same name. Wiring the channel removes its Dictionary instance.

After wiring channels of a device to variables, you can choose to free all wired channels of a device.

When debugging, you can choose to lock, unlock, and force the values of I/O variables.

**To wire the channels of an I/O device**

1. Access the I/O Wiring for the required device.
2. From the rack list, click the I/O device having the I/O channels to wire. For complex devices, expand the device to access the simple devices, then click **Devices**.
3. In the channels variable list, double-click the channel to wire.
4. From the Variable Selector, select the variable for the channel, then click **OK**.

   The channel's Name field indicates the wired variable's direct representation.
5. To set conversion operations for channels, select the channel in the list, then from the Conversion Function field, choose the required operation from the drop-down list:

- For Boolean channels, set the direct or reversion operations
- For numerical channels, set a Gain and Offset factor

**To free individual channels of an I/O device**

You can free individual wired channels.

1. Access the I/O Wiring for the device and select the channel to unwire. To unwire multiple channels, hold the Ctrl button while clicking each required channel.
2. To unwire individual channels, right-click, and then click **Free selected channels**.

**To lock and unlock an I/O variable**

While debugging, you can lock and unlock I/O variables.

1. Access the I/O Wiring for the device and select the channel to lock or unlock. To lock or unlock multiple channels, hold the Ctrl button while selecting each required channel.
2. To lock or unlock the variable, right-click the variable, and then click **Toggle lock on selected channels**.

**To force the value of an I/O variable**

While debugging, you can force the values of locked I/O variables. Variable direction is determined from the direct representation definition for the I/O wiring.

1. Access the I/O Wiring for the device and locate the required variable.
2. Write the required value in the respective value column:
   - For an input variable, write the value in the *Logical Value* column.
   - For an output variable, write the value in the *Physical Value* column.
3. To unlock a variable, click the checkbox in the *Lock* column.

## See Also

I/O Devices

I/O Wiring

## I/O Conversions

You can apply conversion operations to I/O variables. These conversions are possible using two methods:

- Conversion Tables
- Conversion Functions

## Conversion Tables

A conversion table is a set of points defining an analog conversion. You can attach a conversion table to an analog input or output variable to create a proportional relationship between electrical values (read on input sensor or sent to the output device) and physical values (used in application programming).

A conversion table enables filtering the values of any input or output analog variable of a project. You attach a conversion table to a variable from a dictionary instance.

You create conversion tables from the device level before attaching these to variables.

**To create a conversion table**

1. From the Solution Explorer, right-click the device, and then click **Conversion Tables**.

   The Conversion Tables editor is displayed.

2. In the Conversion Tables section, click **Add**.

   A conversion table is added to the list.

3. In the Details section, specify a name for the conversion table, then define the required points for the conversion by clicking Add and specifying the electrical and physical values.

**To edit the points of an existing conversion table**

1. From the Solution Explorer, right-click the device, and then click **Conversion Tables**.
2. In the Conversion Tables section, select the conversion table to modify.
3. In the Details section, perform the required modifications:

- To edit the values of existing points, replace the required value then press Enter.
- To add a set of points, click **Add**, then specify the values for the point.
- To remove a set of points, select the points and click **Remove**.

**To delete a conversion table**

1. From the Solution Explorer, right-click the device, and then click **Conversion Tables**.
2. In the Conversion Tables section, select the conversion table to remove, then click **Remove**.

## Conversion Functions

Conversion functions are "C" functions creating a relationship between an electrical value of a variable (read on the input sensor or sent to the output device) and its physical value (used in the application expressions). Such functions are called by the I/O manager each time an analog variable using the conversion is input to or output from the project. Conversion functions are divided into two parts: input conversion and output conversion.

You can apply conversion functions to integer or real analog variables since these are always defined using floating values. You attach a conversion function to a variable from a dictionary instance.

The interface is the same for all conversion functions. You provide the "C" definition for this interface in the "TACN0DEF.H" definition file. The library manager enables controlling the "C" source code of a conversion function.

## I/O Wiring Keyboard Shortcuts

The following keyboard shortcuts are available for use with I/O wiring. Some shortcuts do not apply or may differ while debugging.

| | |
|---|---|
| Ctrl+N | Adds a device (not available while debugging) |
| Ctrl+F | Frees all channels of selected devices (not available while debugging) |
| Ctrl+R | Frees selected channels of a device (not available while debugging) |
| Ctrl+H | Toggles between a real or virtual I/O device (not available while debugging) |
| Ctrl+L | While debugging, toggles between locking and unlocking selected channels |

# FBD Language

The Functional Block Diagram (FBD) is a graphic language enabling programmers to build complex procedures by taking existing functions from the standard library, function section, or function block section.

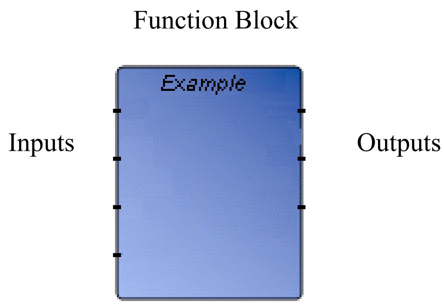
In FBD containers, you can also include LD elements such as coils, contacts, jumps, labels, and returns. However, in contrast to LD elements usage in LD containers where these elements follow strict graphical positioning regulations, LD elements within FBD container are independent of these regulations.

**See Also**

FBD Diagram Main Format
Debugging FBD Programs

# FBD Diagram Main Format

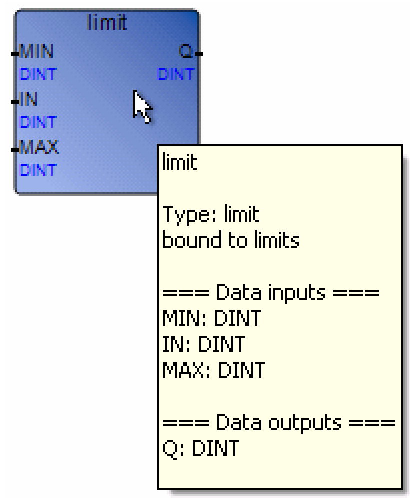
FBD diagrams describe a process between input variables and output variables. A process is described as a set of elementary blocks. Input and output variables are connected to blocks by connection lines. Outputs of blocks can also be connected to inputs of other blocks.




An entire process represented by an FBD program is built using the available variables, operators, functions, and function blocks. Each block has either a fixed or defined number of input and output connection points. A block is represented by a single rectangle. The inputs are connected on its left border. The outputs are connected on its right border. An elementary block performs a single function between its inputs and its outputs. The name of the function to be performed by the block is written inside its rectangular shape. Each input or output of a block is labeled and has a well defined type.

Function Name

Inputs




Output

Input variables of an FBD program must be connected to input connection points of blocks. The type of each variable must be the same as the type expected for the associated input. An input for FBD diagram can be a literal, any internal or input variable, an output variable, or a block output.

Output variables of an FBD program must be connected to output connection points of blocks. The type of each variable must be the same as the type expected for the associated block output. An output for FBD diagram can be any internal or output variable, or the name of the function (for functions only). When an output is the name of the currently edited function, it represents the assignment of the return value for the function (returned to the calling program).

Input and output variables, inputs and outputs of the blocks are wired together with connection lines, or links. Single lines can be used to connect two logical points of a diagram:

- An input variable and an input of a block
- An output of a block and an input of another block
- An output of a block and an output variable

The connection is oriented, meaning that the line carries associated data from left to right. The left and right ends of the connection line must be of the same data type.

Vertical bars accept several connections on the left and several connections on the right. Each connection on the right is equal to the OR combination of the connections on the left. All ends of the connections must be of the same data type.

### See Also

Execution Order of FBD Programs

# Execution Order of FBD Programs

You can show the order of execution in the form of numerical tags for the following elements in an FBD program: coils, contacts, LD vertical connections, corners, returns, jumps, functions, operators, function blocks, and variables where a value is assigned in the program. When the order cannot be determined, the tags display question marks (?). You can perform this task from the menu bar, the toolbar, or keyboard shortcut (Ctrl+W).

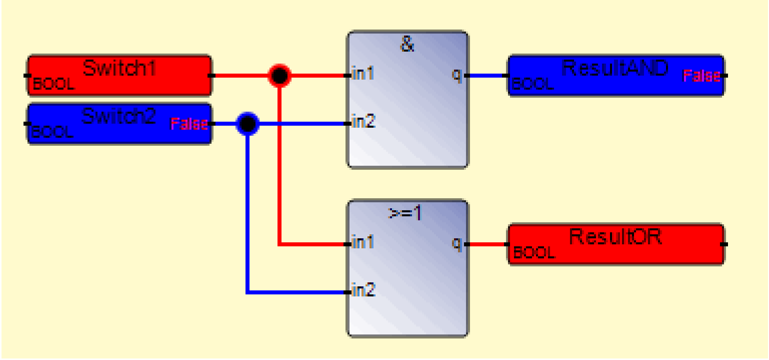
For the execution order of a program, a block is any object in the diagram, a network is a sequence of connected blocks, and the position of a block is based on its top-left corner. The following rules apply to the execution order of the program:

- Networks are executed from left to right, top to bottom.
- All inputs must be resolved before executing the block. When the inputs of two or more blocks are resolved at the same time, the decision for the execution is based on the position of the block (left to right and top to bottom).
- The outputs of a block are executed recursively from left to right and top to bottom.

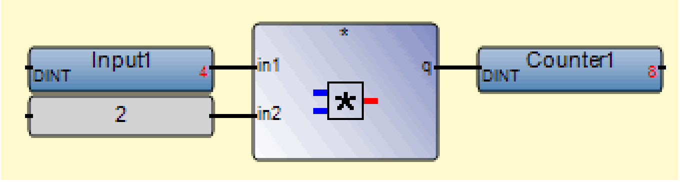
# Debugging FBD Programs

When power flow debugging FBD programs, you can monitor the output values of elements. These values are displayed using color, numeric, or textual values according to their data type:

- Output values of boolean type are displayed using color. The output value color continues to the next input. When the output value is unavailable, boolean elements remain black. The colors are red when True and blue when False.




- Output values of DINT, REAL, TIME, and MESSAGE type are displayed as a numeric or textual value in the element.




When the output value for a numeric or textual value is unavailable, the *WAIT* text is displayed in the output label. Values are also displayed in the corresponding dictionary instance.

# FBD Elements

When programming in FBD, you place elements in the workspace by dragging them from the Toolbox into the language container. For FBD POUs, the following elements are available:

- Blocks
- Variables
- Vertical Bars
- Labels
- Jumps
- Returns
- Rungs
- Left Power Rails
- Right Power Rails
- Coils
- Contacts
- Regions
- Comments

**See Also**

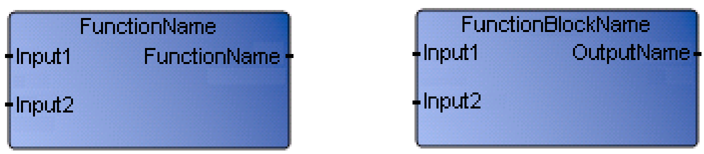
FBD Diagram Main Format
Execution Order of FBD Programs

## Blocks

Block elements can be operators, functions, or function blocks. You connect block inputs and outputs to variables, contacts or coils, or other block inputs and outputs. You insert block elements in language containers.

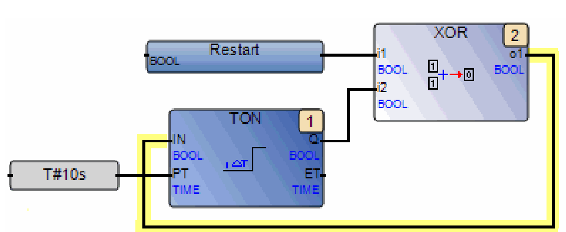
Functions and function blocks are represented by a box displaying the name of the function, function block, or operator, and the parameter names.

For functions, the return parameter is the only output. For function blocks, multiple return parameters can provide multiple outputs. The return parameter of a function has the same name as the function. The return parameters of a function block can have any name.




You define the parameters of POUs in the Parameters view.

For loops in blocks, you need to use local variables since these are initialized with a value. When using loops, the first execution may produce incorrect outputs due to the execution order of elements in the diagram or the initial values of temporary variables. For example, the following diagram produces a warning when compiling since the TON block is executed before the XOR operator. Whereas, moving the XOR operator to the upper left corner of the diagram eliminates the warning since the XOR operator becomes first in the execution order.



You can resize blocks elements.

**To access the Parameters view**

The Parameters view is available from functions or function blocks located in the Solution Explorer.

1. In the Solution Explorer, right-click the required function or function block, then click **Parameters** in the contextual menu.

   The Parameters view is displayed.

2. To define the parameters of a function or function block, select the block, then enter the required information in the fields provided.

**To insert a block element**

1. From the Toolbox, drag the block element into the language container.

   The block selector is displayed.

2. In the Block Selector, choose the required function block, then click **OK**. You can sort the block list according to the columns by setting these in ascending or descending order.

The selected block is displayed in the language container.

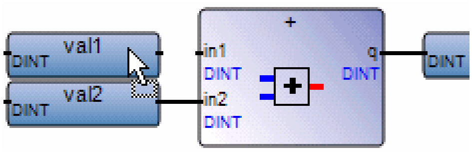
### See Also

FBD Diagram Main Format

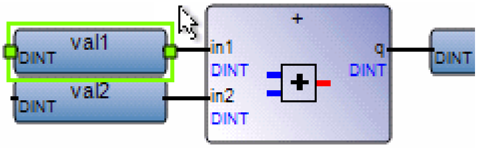
## Variables

To connect a new symbol to an existing one (another variable, a block input, or a block output) in the workspace, keep the mouse button depressed (the cursor becomes a "ghost" symbol) and drag the element until its connecting line on the left (or right) overlaps an existing connecting point. When the mouse is released, the new symbol is automatically created and linked.

Drag to place the existing element:

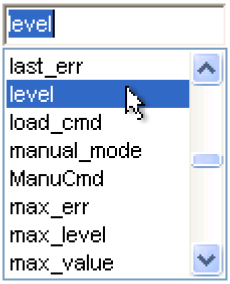



Release the mouse button. The variable is automatically connected:

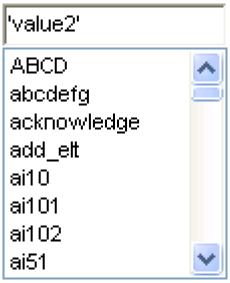



You replace existing variables in POUs by double-clicking them to access the Variable Selector or single-clicking them to select from a drop-down combo-box containing the global and local variables. Also, you can single-click a variable, then type a literal value in the text box provided. When inserting literal values beginning with a letter or an underscore, enclose these in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:

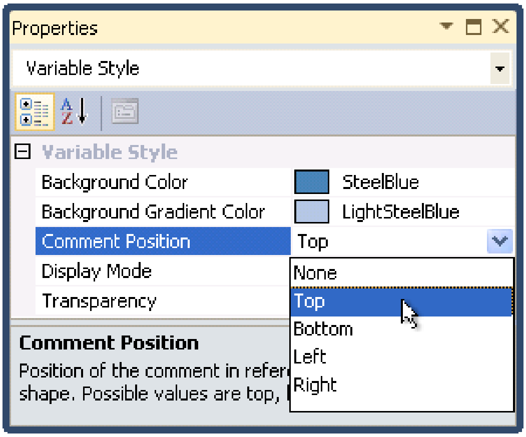


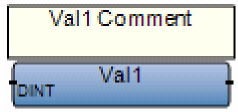

Type a literal value in the text box:




When selecting items such as local variables, global variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items.

For input and output variables, you can choose to display comments entered in the dictionary. From the View menu, you can access the Properties window where you can define the *Comment Position* property.




You can resize variables displayed in the workspace.

**To insert a variable**

1. From the Toolbox, drag the variable element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**

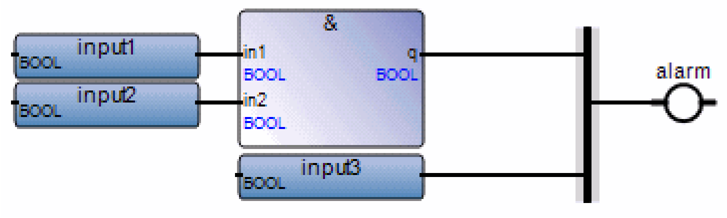
The variable is displayed in the language container.

## See Also

FBD Diagram Main Format

## Vertical Bars

Vertical bars are graphic components of FBD programs enables closing multiple parallel links. More than one horizontal links on the left side of a vertical bar are connected to one link on the right side. The Boolean state of the right end is the logical OR between all the left extremities.




**To insert a vertical bar**

- From the Toolbox, drag the vertical bar element into the language container.

The vertical bar is displayed in the language container.

## Labels

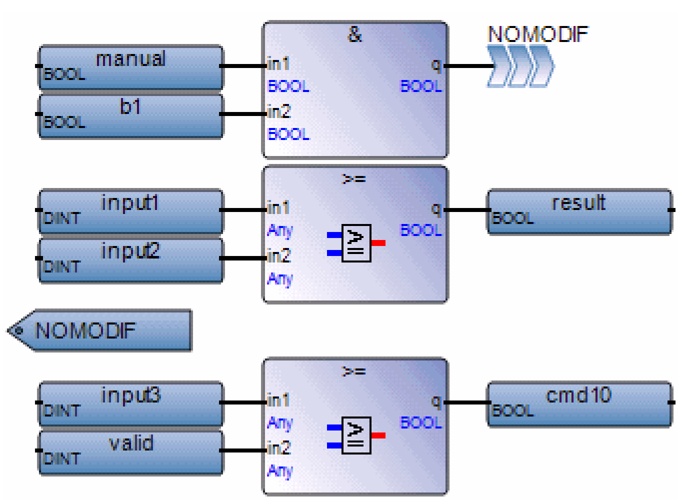
Labels can be placed anywhere in an FBD diagram. These are used as a target for jump instructions, to change the execution order of the diagram. Labels are not connected to other elements.

Place labels on the left of the diagram in order to increase diagram readability.

Labels are used to control the execution of the diagram. No other object may be connected on the right of a label symbol.

If the connection line on the left of the jump symbol has the Boolean state TRUE, the execution of the program directly jumps to after the corresponding label symbol.

### Example




**To insert a label**

1. From the Toolbox, drag the label element into the language container.

2. In the language container, click the label, then type a label name in the space provided.

The label is displayed in the language container.

**See Also**

Jumps

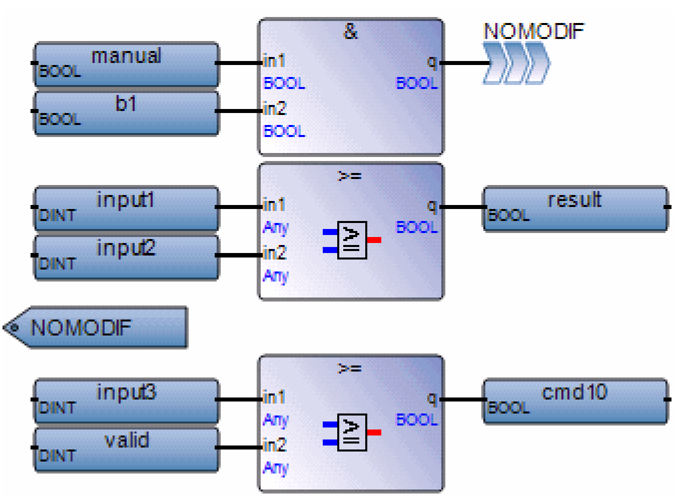
## Jumps

A Jump symbol must be linked to a Boolean point. When this Boolean (left) connection is TRUE, the execution of the diagram Jumps directly to the target Label.

Jumps are used to control the execution of the diagram. No other object may be connected on the right of a jump symbol.

If the connection line on the left of the jump symbol has the Boolean state TRUE, the execution of the program directly jumps to after the corresponding label symbol.

### Example




**To insert a jump to a label**

Before inserting jumps, define one or more labels within the program.

1. From the Toolbox, drag the jump element into the language container.
2. In the language container, click the jump element, then select the required label name from the drop-down combo-box.

The jump is displayed in the language container with the required label name.

### See Also

Labels

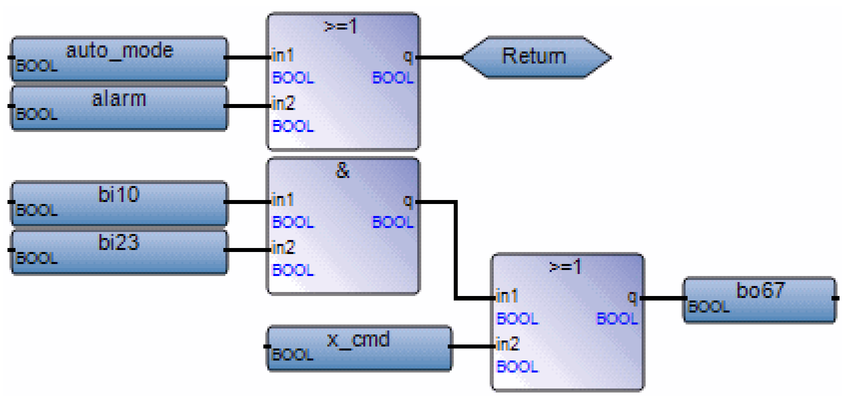
## Returns

If the connection line (to the left of the Return symbol) has the Boolean state TRUE, the Program ends - no further part of the diagram is executed.

No connection can be put on the right of a RETURN symbol.

The "<RETURN>" keyword may occur as a diagram output. It must be connected to a Boolean output connection point of a block. The RETURN statement represents a Conditional End of the program: if the output of the box connected to the statement has the Boolean value TRUE, the end (remaining part) of the diagram is not executed.

### Example




(* ST equivalence: *)

```
If auto_mode OR alarm Then
Return;
End_if;
bo67 := (bi10 AND bi23) OR x_cmd;
```

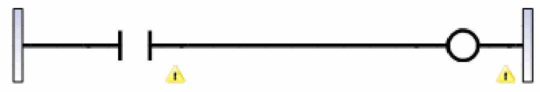
**To insert a return**

- From the **Toolbox**, drag the return element into the language container.

The return is displayed in the language container.

## Rungs

Rungs are graphic components of FBD programs and represent a group of circuit elements leading to the activation of a coil. Dragging the rung element into the workspace inserts a left power rail linked to a right power rail. Also, the rung contains a direct contact and a direct coil. Error symbols ( ) indicate that the direct contact and direct coil are undefined.



**To insert a rung**

- From the Toolbox, drag the rung element into the language container.

The rung is displayed in the language container.

## Left Power Rails

Left Power Rails are graphic components of FBD programs that represent the left boundary of a rung. Any horizontal link connected to a left power rail has the boolean state TRUE.

You can link left power rails to right power rails as well as many FBD and LD elements, including variables, blocks, jumps, returns, vertical bars, coils, and contacts.

**To insert a left power rail**

- From the Toolbox, drag the left power rail element into the language container.

The left power rail is displayed in the language container.

## Right Power Rails

Right Power Rails are graphic components of FBD programs that represent the right boundary of a rung.

You can link right power rails to left power rails as well as many FBD and LD elements, including variables, blocks, vertical bars, coils, and contacts.

**To insert a right power rail**

- From the Toolbox, drag the right power rail element into the language container.

The right power rail is displayed in the language container.

## Coils

Coils are graphic components of LD programs that you can use in FBD programs representing the assignment of Boolean outputs. A coil represents an action. It must be connected on the left to a Boolean symbol, such as a contact or the Boolean output of a block.

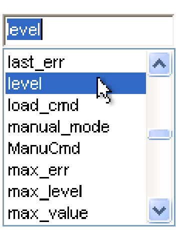
The following types of coils are available from the FBD toolbox:

- Direct Coil
- Reverse Coil
- Set Coil
- Reset Coil

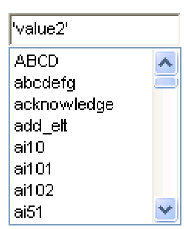
You can change the type of a coil at any time following its insertion.

When inserting coils in POUs, you assign variables using the Variable Selector. Names of assigned variables are displayed above the coil elements within POUs. You replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, you can single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:




Type a literal value in the text box:




### To insert a coil

You can insert coils from the Toolbox.

1. From the Toolbox, drag the desired coil type into the language container and place it on the rung.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.

The coil element and its associated variable name are displayed in the language container.

**To insert a parallel coil**

1. From the Toolbox, drag a coil element into the language container while placing it parallel to the existing coil.
2. Drag the left and right connections to the respective connection points on the rung.

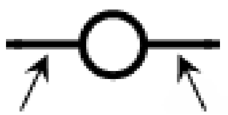
The required coil is displayed on the parallel branch.

**To change the type of a coil**

- In the language container, select the coil, then select the required type in the Modifier property of the Properties window.

## Direct Coil

Direct Coils enable a Boolean output of a connection line Boolean state.

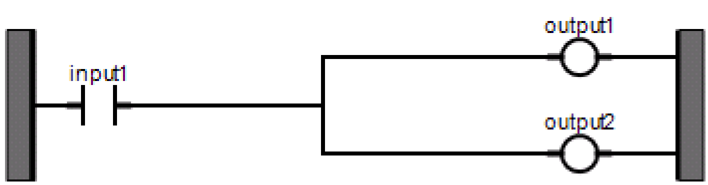


Left Connection    Right Connection

The associated variable is assigned with the Boolean state of the left connection. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

### Example



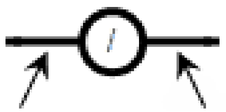

(* ST Equivalence: *)

```
output1 := input1;
output2 := input1;
```

### See Also
Coils

## Reverse Coil

Reverse coils enable a Boolean output according to the Boolean negation of a connection line state.

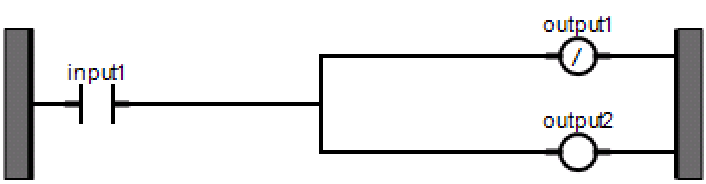


Left Connection   Right Connection

The associated variable is assigned with the Boolean negation of the state of the left connection. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

### Example




(* ST Equivalence: *)

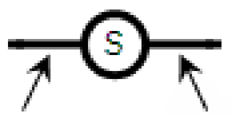
```
output1 := NOT (input1);
output2 := input1;
```

### See Also

Coils

## Set Coil

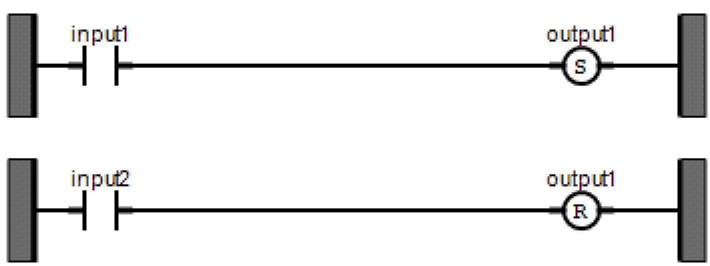
Set coils enable a Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is set to TRUE when the boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a RESET coil. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example



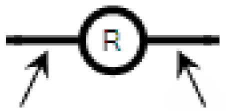

(* ST Equivalence: *)

```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

**See Also**
Coils

## Reset Coil

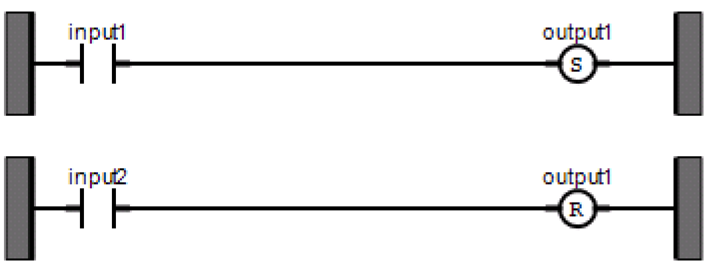
Reset coils enable Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is reset to FALSE when the Boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a SET coil. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

**See Also**

Coils

# Contacts

Contacts are graphic components of LD diagrams that you can use in FBD programs. Depending on the type of contact, it represents the value or function of an input or internal variable.

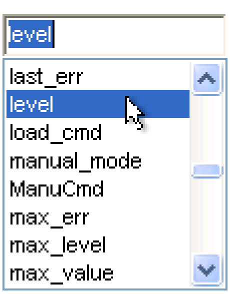
The following contact types are available from the FBD toolbox:

- Direct Contact
- Reverse Contact
- Pulse Rising Edge Contact
- Pulse Falling Edge Contact

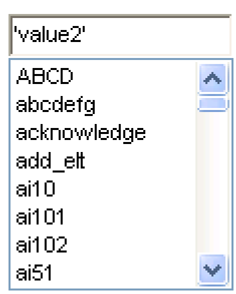
You can change the type of a contact at any time following its insertion.

When inserting contacts in POUs, you assign variables using the Variable Selector. Names of assigned variables are displayed above the contact elements within POUs. You replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, you can single-click existing variables, then type literal values in the text boxes provided. When inserting literal values that being with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:




Type a literal value in the text box:



### To insert a contact

You can insert contacts from the Toolbox.

1. From the Toolbox, drag the desired contact type into the language container and place it on the rung.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.

The contact and its associated variable name are displayed in the language container.

### To insert a parallel contact

1. From the Toolbox, drag the contact element into the language container while placing it parallel to the existing contact.
2. Drag the left and right connections to the respective connection points on the rung.

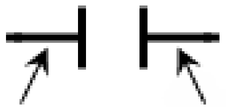
The required contact is displayed on the parallel branch.

### To change the type of a contact

- In the language container, select the contact, then select the required type in the Modifier property of the Properties window.

## Direct Contact

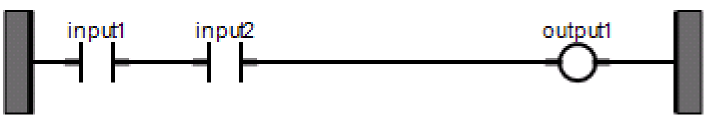
Direct contacts enable a Boolean operation between a connection line state and a Boolean variable.



Left Connection    Right Connection

The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the value of the variable associated with the contact.

### Example




(* ST Equivalence: *)

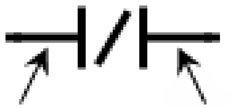
```
output1 := input1 AND input2;
```

### See Also

Contacts

## Reverse Contact

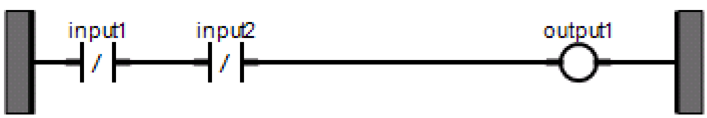
Reverse contacts enable a Boolean operation between a connection line state and the Boolean negation of a Boolean variable.



| | |
|---|---|
| Left Connection | Right Connection |

The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the Boolean negation of the value of the variable associated with the contact.

### Example




(* ST Equivalence: *)

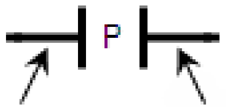
```
output1 := NOT (input1) AND NOT (input2);
```

### See Also

Contacts

## Pulse Rising Edge Contact

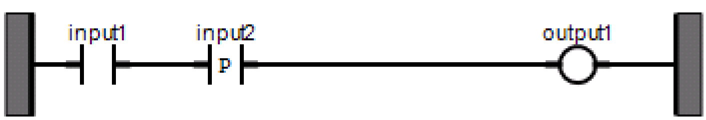
Pulse rising edge (positive) contacts enable a Boolean operation between a connection line state and the rising edge of a Boolean variable.




Left Connection    Right Connection

The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable rises from FALSE to TRUE. The state is reset to FALSE in all other cases.

### Example




(* ST Equivalence: *)

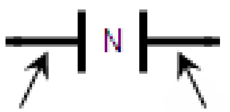
```
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)
```

### See Also

Contacts

## Pulse Falling Edge Contact

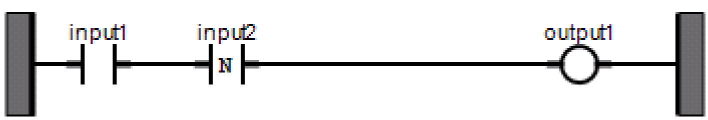
Pulse falling edge (negative) contacts enable a Boolean operation between a connection line state and the falling edge of a Boolean variable.




Left Connection    Right Connection

The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable falls from TRUE to FALSE. The state is reset to FALSE in all other cases.

### Example




(* ST Equivalence: *)

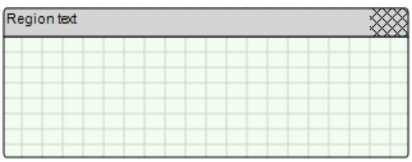
```
output1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev is the value of input2 at the previous cycle *)
```

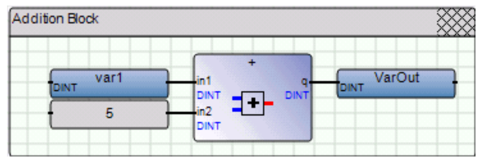
### See Also

Contacts

## Regions

Regions delineate and group together areas of an FBD POU. A region consists of a header and a delineated zone grouping together elements.The header section enables entering free-format text. After entering text in the header, click elsewhere in the region to exit editing mode. When moving the location of a region in the language container, you can also move all the content grouped within. You can resize regions.




**To insert a region**

- From the Toolbox, drag the region element into the language container.

The region element is displayed in the language container.

**To move a region**

1. In the language container, left-click the top right corner of the region element and hold the mouse button.
2. Drag the region element to the required location and release the mouse button.

The region and the elements contained inside have moved location in the language container.

### See Also

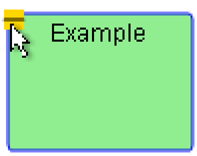
Comments

## Comments

Comments are free format text inserted anywhere in the FBD POU, for documentation purposes only. After entering text, click elsewhere in the workspace to exit editing mode.

You can expand and collapse comment elements displayed in the workspace by clicking the maximize and minimize buttons. You can also resize comments.

Minimize




Maximize

**To insert a comment**

You can apply text formatting options including bold, italic, underline, strikethrough, and justify from the Description Editor toolbar. You can also define the foreground color.

1. From the Toolbox, drag the comment element into the language container.
2. In the language container, double-click the comment, then type the required text within the space provided.

The comment is displayed in the language container.

# FBD Keyboard Shortcuts

The following keyboard shortcuts are available for use with the FBD language. Some shortcuts do not apply or may differ while debugging.

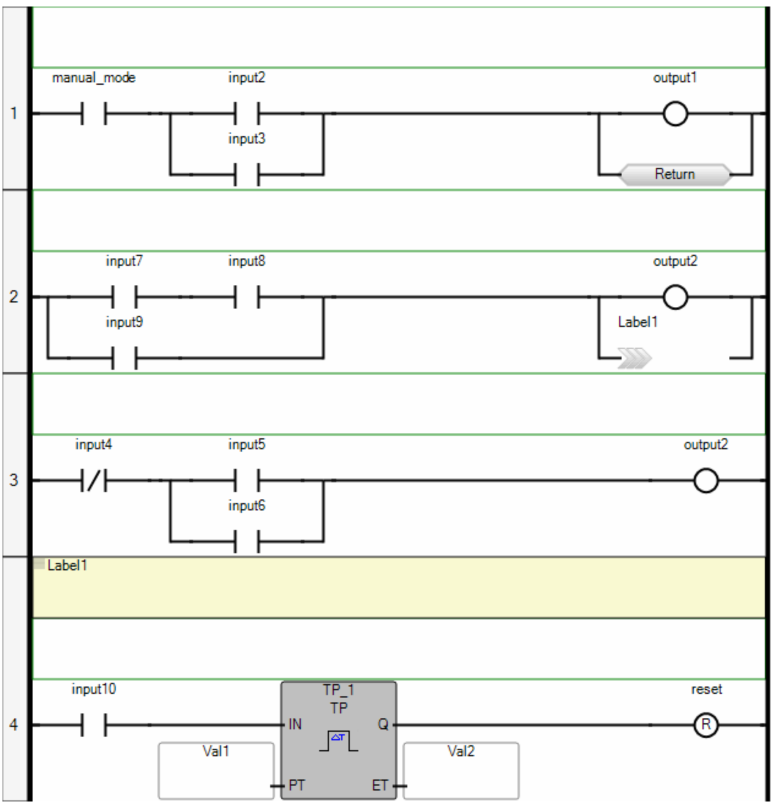
| | |
|---|---|
| Ctrl+A | Selects all elements (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while debugging) |
| Ctrl+V | Pastes elements saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl+X | Cuts the selected elements to the clipboard (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Ctrl+Alt+G | Enables/disables the grid in the language container |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+R | Toggles between Auto-Input and Manual-Input. Auto-Input automatically opens the Block Selector and Variable Selector (not available while debugging). |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctrl+I | Italicizes selected comment text (not available while debugging) |
| Ctrl+U | Underlines selected comment text (not available while debugging) |
| Enter | When a function block is selected, opens the Block Selector (not available while debugging).<br><br>When a variable is selected, opens the Variable Selector (not available while debugging).<br><br>When a comment is selected, starts editing it (not available while debugging). |
| Ctrl+Enter | When a variable is selected, opens the drop-down list of available variables (not available while debugging).<br><br>When editing a comment, confirms the text (not available while debugging). |
| Ctrl+- | Decreases the magnification |

| | |
|---|---|
| Ctrl+= | Increases the magnification |
| Ctrl+0 | 100% magnification |
| Ctrl+1 | Inserts a variable (not available while debugging) |
| Ctrl+2 | Inserts a function block (not available while debugging) |
| Ctrl+3 | Inserts a comment (not available while debugging) |
| Shift+Up Arrow | Reduces the height of the selected element (not available while debugging) |
| Shift+Down Arrow | Increases the height of the selected element (not available while debugging) |
| Shift+Left Arrow | Reduces the width of the selected element (not available while debugging) |
| Shift+Right Arrow | Increases the width of the selected element (not available while debugging) |
| Ctrl+Up Arrow | Moves the selection to the next element located higher in the diagram without keeping the previous element selected. While debugging, scrolls up. |
| Ctrl+Down Arrow | Moves the selection to the next element located lower in the diagram without keeping the previous element selected. While debugging, scrolls down. |
| Ctrl+Left Arrow | Moves the selection to the next element located to the left in the diagram without keeping the previous element selected. While debugging, scrolls left. |
| Ctrl+Right Arrow | Moves the selection to the next element located to the right in the diagram without keeping the previous element selected. While debugging, scrolls right. |
| Alt+Shift+Up Arrow | When a function block is selected, navigates up the different inputs and outputs (not available while debugging) |
| Alt+Shift+Down Arrow | When a function block is selected, navigates down the different inputs and outputs (not available while debugging) |
| Alt+Shift+Left Arrow | When a function block is selected, navigates left across the different inputs and outputs (not available while debugging) |
| Alt+Shift+Right Arrow | When a function block is selected, navigates right across the different inputs and outputs (not available while debugging) |

| | |
|---|---|
| Ctrl+Page Up | Jumps to the top of the language container |
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Alt+Up Arrow | Scrolls up |
| Alt+Down Arrow | Scrolls down |
| Alt+Left Arrow | Scrolls left |
| Alt+Right Arrow | Scrolls right |
| Up Arrow | Moves selected elements up the language container. While debugging, scrolls up. |
| Down Arrow | Moves selected elements down the language container. While debugging, scrolls down. |
| Left Arrow | Moves selected elements left across the language container. While debugging, scrolls left. |
| Right Arrow | Moves selected elements right across the language container. While debugging, scrolls right. |
| Delete | Removes the selected elements (not available while debugging) |

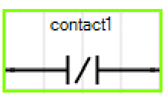
# LD Language

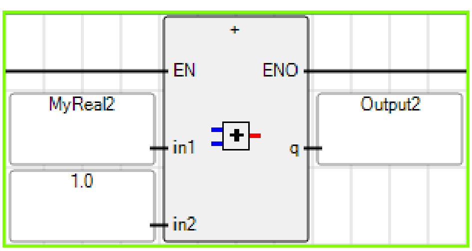
Ladder Diagram (LD) is a graphic representation of Boolean equations, combining contacts (input arguments) with coils (output results). The LD language enables the description of tests and modifications of Boolean data by placing graphic symbols into the program chart. LD graphic symbols are organized within the chart as an electric contact diagram. Thus, the term "ladder" coming from the concept of rungs connected to vertical power rails at both ends where each rung represents an individual circuit.



You can adjust editor and view settings for individual or all Ladder Diagrams. When working in a Ladder Diagram, you set the properties for the diagram from the Container properties in the Properties window. You set the properties for all Ladder Diagrams using the options available from the Tools menu. Some of the available properties include the following:

- background and gradient colors for operators, functions, and function blocks
- displaying the grid as well as the height and width of grid cells, in pixels
- the height and width of elements, in grid cells. Basic elements are blocks without inputs or outputs, coils, and contacts. For blocks, each input and output adds a basic element dimension. For example, note the contact using the default settings of one grid cell high by four grid cells wide. The following block uses a basic element width for the inputs, another for the block, and another for the outputs. The block uses a basic element height for the EN/ENO level, another for the first input and the output, and another for the second input.




- the font type, size, style, and color applied to the text displayed in elements
- various options such as displaying comments and labels, aligning coils, and setting the colors for variables, labels, comments, power rails, and rung headers

**See Also**

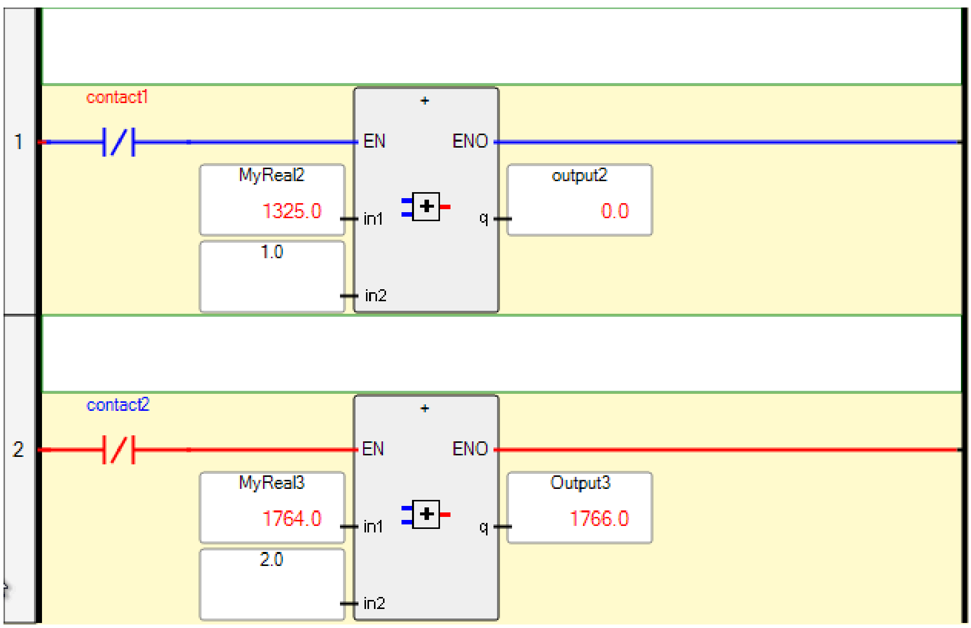
Debugging LD Programs

# Debugging LD Programs

When power flow debugging LD programs, you can monitor the output values of elements. These values are displayed using color, numeric, or textual values according to their data type:

- Output values of boolean type are displayed using color. The output value color continues to the next input. When the output value is unavailable, boolean elements remain black. The default colors are red when True and blue when False. You can customize the colors used for boolean items.

- Output values of DINT, REAL, TIME, and MESSAGE type are displayed as a numeric or textual value in the element. When the output is a structure type, the displayed value is the selected member.




When the output value for a numeric or textual value is unavailable, the *WAIT* text is displayed in the output label. Transitional elements such as Pulse rising edge (positive) contacts, having an unstable state, remain black. Values are also displayed in the corresponding dictionary instance.

When the device is in the DEBUGGING state, you can choose to perform one of the following operations:

- Switch execution to real-time mode
- Switch execution to cycle-to-cycle mode
- Execute one cycle

**To switch execution to real-time mode**

- From the Target Execution toolbar, click .

The POU executes in real-time mode.

**To switch execution to cycle-to-cycle mode**

- From the Target Execution toolbar, click .

The POU executes in cycle-to-cycle mode.

**To execute one cycle**

- From the Target Execution toolbar, click .

The POU executes one device cycle.

## LD Elements

When editing an LD POU, you can place elements in a language container by dragging them from the LD Toolbox. An element is inserted at the current position in the diagram. When inserting subsequent elements, these are placed to the right of the selected element on the rung, then onto the next rung. For LD POUs, the following elements are available:

- Rungs
- Blocks
- Coils
- Contacts
- Jumps
- Returns
- Branches

## Rungs

Rungs are graphic components of LD programs and represent a group of circuit elements leading to the activation of a coil. Rungs have labels to identify them within the diagram. Labels along with jumps enable controlling the execution of a diagram. The label and jump must have the same name. When the connection on the left of the jump element has the TRUE Boolean state, the diagram execution proceeds at the label element. Comments are free format text inserted above the rung, for documentation purposes only.

**To insert a rung**

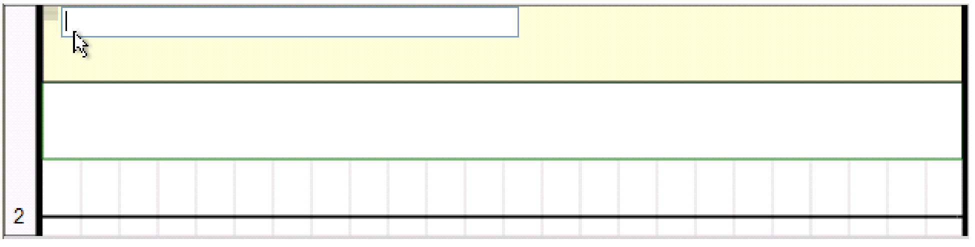
You can insert rungs from the Toolbox or using keyboard shortcuts.

- From the Toolbox, drag the rung element into the language container.

The rung is displayed in the language container.

**To define the label for a rung**

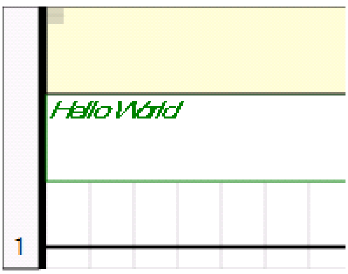
1. In the language container, click anywhere, then from the contextual menu, choose **Add Label**.
2. In the upper left-hand corner, click in the text area beside the grey square and type the required label text.



**To define the comment for a rung**

You place comments in the space above the rung. After entering text, click elsewhere in the workspace to 'validate' the comment. Text formatting options including bold, italic, underline, strikethrough, and justify, are available from the Format menu. Using the Format menu, you can also define the foreground color.

- In the language container, click the rectangular space above the rung, then type the required text.



## Blocks

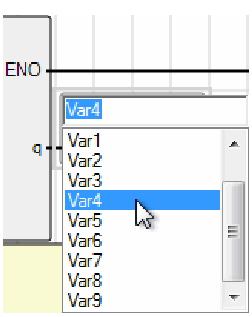
In a language container, you connect blocks to Boolean lines. Blocks can be operators, functions, or function blocks. Boolean inputs and outputs are not always contained within blocks. Boolean inputs connecting blocks to rungs are always executed each cycle. Boolean outputs connecting blocks to rungs control the remaining rung power flow. When inserting blocks in a diagram, the EN and ENO parameters are added to some block interfaces. You can also force the inclusion of the EN and ENO parameters for blocks having either one Boolean input, one Boolean output, or no Boolean input and output. You activate the Enable EN/ENO option from the Ladder Diagram options.

For functions and function blocks, you set the value of return parameters using coils. The return parameter of a function has the same name as the function. The return parameters of a function block can have any name.

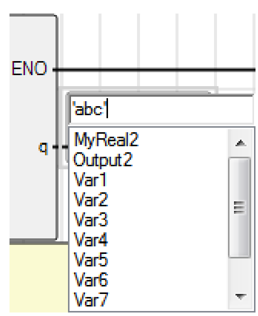
You insert blocks from the LD Toolbox. You can set the type of a block using the Block Selector at any time following insertion. When you set the type of block, variables are automatically displayed and are connected to the inputs and outputs of the block.

You replace input and output variables by double-clicking them to access the Variable Selector or single-clicking them to select from a drop-down combo-box containing the global and local variables. Also, you can single-click a variable, then type a literal value in the text box provided. When inserting literal values that being with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:



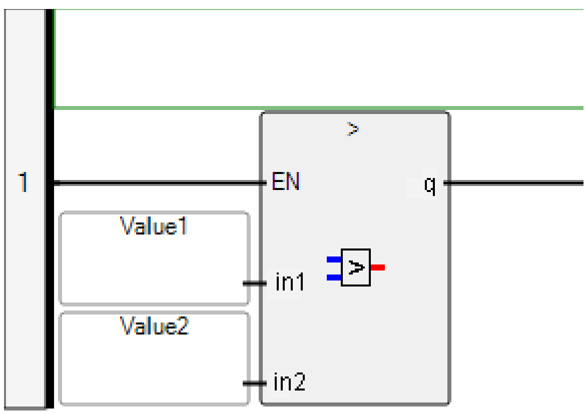

Type a literal value in the text box:



When selecting items such as local variables, global variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items.

### EN Input

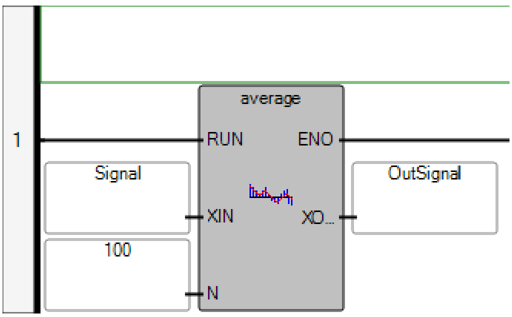
For operators, functions, and function blocks where the first input is not a Boolean data type, another input called EN is automatically inserted at the first position since the first input is always connected to the rung. The block is executed only when the EN input is TRUE. The following example shows a comparison operator and its equivalent code expressed in ST.




```
IF rung_state THEN
q := (value1 > value 2);
ELSE
q := FALSE;
END_IF;
(* continue rung with o1 state *)
```

### ENO Output

For operators, functions, and function blocks where the first output is not a Boolean data type, another output called ENO is automatically inserted at the first position since the first output is always connected to the rung. The ENO output always has the same state as the first input of the block. The following example shows the AVERAGE function block and its equivalent code expressed in ST.

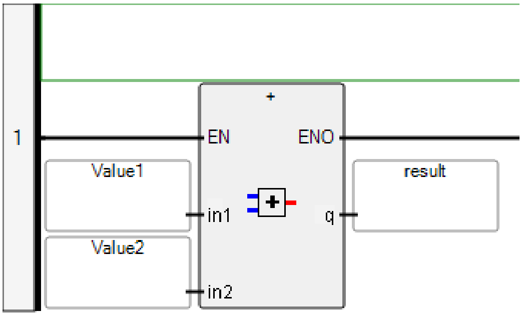



```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
```

(* continue rung with eno state *)

## EN and ENO Parameters

In some cases, both **EN** and **ENO** parameters are required. The following example shows an arithmetic operator and its equivalent code expressed in ST.




```
IF rung_state THEN
result := (value1 + value2);
END_IF;
eno := rung_state;
```

(* continue rung with eno state *)

**To access the Parameters view**

The Parameters view is available from functions or function blocks located in the Solution Explorer.

1. In the Solution Explorer, right-click the required function or function block, then click **Parameters** in the contextual menu.

   The Parameters view is displayed.

2. To define the parameters of a function or function block, enter the required information in the Parameters view.

**To insert a block**

You can insert blocks from the Toolbox or using keyboard shortcuts.

1. From the Toolbox, drag the function block element into the language container and place it on the rung.

   The Block Selector is displayed.

2. In the Block Selector, locate the required block. You can sort the block list according to the columns by setting these in ascending or descending order.

   - To force the inclusion of the EN/ENO parameters, select *Enable EN/ENO*.

3. Click **OK**.

The selected block is displayed on the rung.

**To insert a parallel block**

1. From the Toolbox, drag the branch element onto the existing block in the language container.

2. To place a block element on the branch, do the following:

   **a)** From the Toolbox, drag the block element into the language container, placing it on the branch.

   The Block Selector is displayed.

   **b)** In the Block Selector, locate the required block. You can sort the block list according to the columns by setting these in ascending or descending order.

   - To force the inclusion of the EN/ENO parameters, select *Enable EN/ENO*.

   **c)** Click **OK**.

The selected block is displayed on the branch.

## Coils

Coils are graphic components of LD programs and represent the assignment of Boolean outputs. In an LD program, a coil represents an action. It must be connected on the left to a Boolean symbol, such as a contact or the Boolean output of a block.

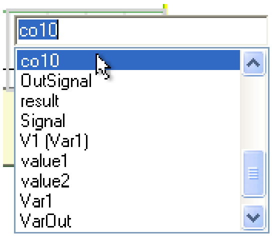
The following types of coils are available from the LD toolbox:

- Direct Coil
- Reverse Coil
- Pulse Rising Edge Coil
- Pulse Falling Edge Coil
- Set Coil
- Reset Coil

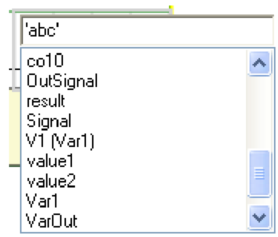
You can change the type of a coil at anytime following its insertion.

When inserting coils in POUs, you assign variables using the Variable Selector. Names of assigned variables are displayed above the coil elements within POUs. You replace existing variables by double-clicking the coil to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, you can single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:




Type a literal value in the text box:




When selecting items such as local variables, global variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items.

**To insert a coil**

You can insert coils from the Toolbox or using keyboard shortcuts.

1. From the Toolbox, drag the desired coil type into the language container and place it on the rung.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.

The coil element and its associated variable name are displayed on the rung.

**To insert a parallel coil**

1. From the Toolbox, drag the branch element into the language container, placing it on the required element.

2. From the Toolbox, drag a coil element into the language container, placing it on the branch element.

   The Variable Selector is displayed.

3. In the Variable Selector, select the required variable, then click **OK**.

The coil element and its associated variable name are displayed on the branch.

The coil is displayed on the branch.

**To change the type of a coil**

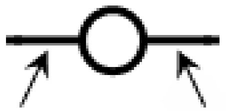
- In the language container, select the coil, then press the space bar.

**To align all coils in a diagram**

1. Right-click in the language container, then choose **Properties** from the contextual menu.
2. In the Properties window, set the *Coil Alignment* property to True.

## Direct Coil

Direct Coils enable a Boolean output of a connection line Boolean state.

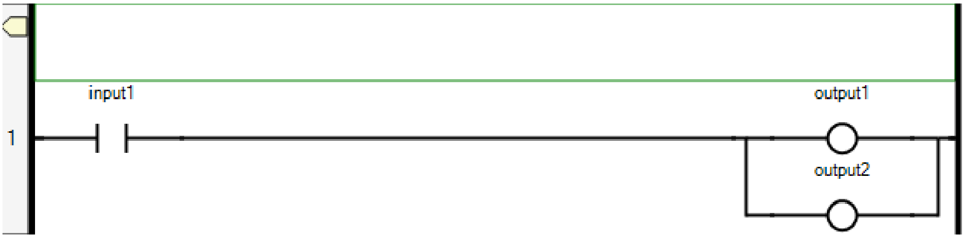


Left Connection Right Connection

The associated variable is assigned with the Boolean state of the left connection. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

### Example




(* ST Equivalence: *)

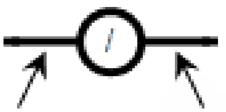
```
output1 := input1;
output2 := input1;
```

### See Also

Coils

## Reverse Coil

Reverse coils enable a Boolean output according to the Boolean negation of a connection line state.

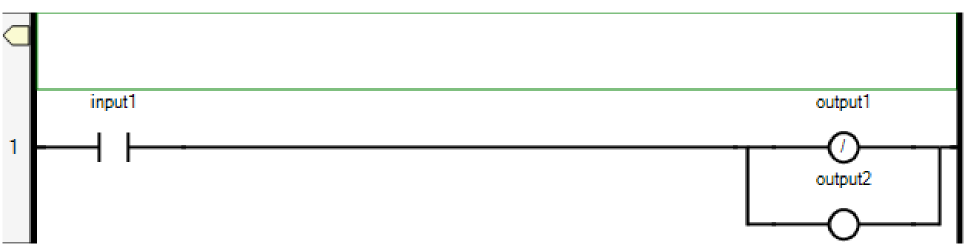


Left Connection    Right Connection

The associated variable is assigned with the Boolean negation of the state of the left connection. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

### Example



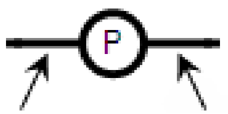

(* ST Equivalence: *)

```
output1 := NOT (input1);
output2 := input1;
```

### See Also
Coils

## Pulse Rising Edge Coil

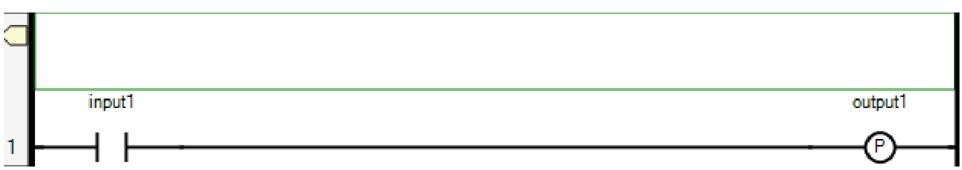
Pulse rising edge coils or "Positive" coils enable Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is set to TRUE when the Boolean state of the left connection rises from FALSE to TRUE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

```
IF (input1 and NOT(input1prev)) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
```

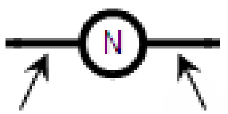
(* input1prev is the value of input1 at the previous cycle *)

### See Also

Coils

## Pulse Falling Edge Coil

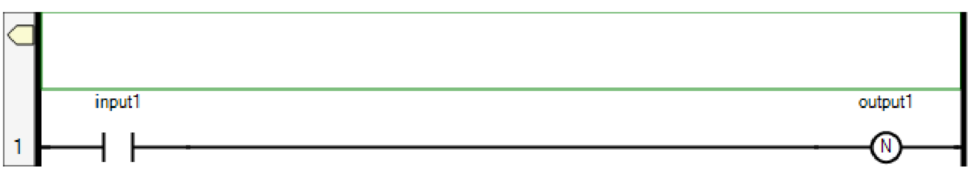
Pulse falling edge coils or "Negative" coils enable Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is set to TRUE when the Boolean state of the left connection falls from TRUE to FALSE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

```
IF (NOT(input1) and input1prev) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
```

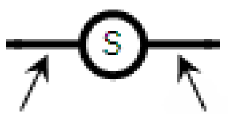
(* input1prev is the value of input1 at the previous cycle *)

### See Also

Coils

## Set Coil

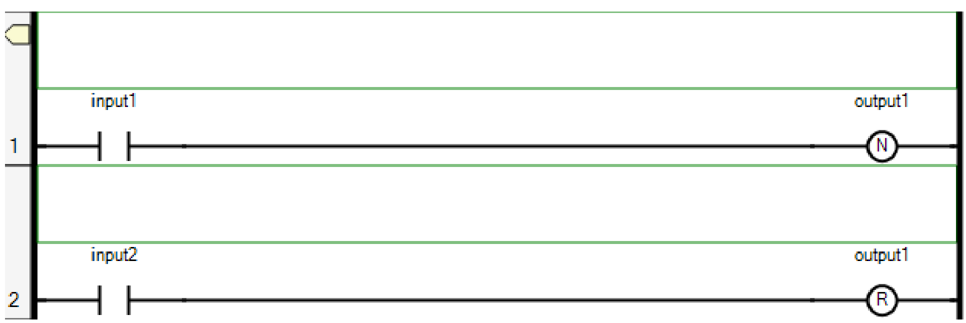
Set coils enable a Boolean output of a connection line Boolean state.



Left Connection    Right Connection

The associated variable is set to TRUE when the boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a RESET coil. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

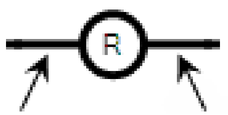
```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

**See Also**

Coils

## Reset Coil

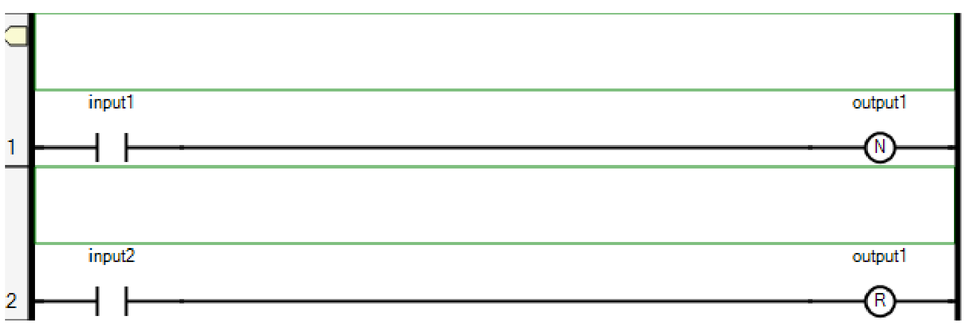
Reset coils enable Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is reset to FALSE when the Boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a SET coil. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

**See Also**

Coils

# Contacts

Contacts are graphic components of LD diagrams. Depending on the type of contact, it represents the value or function of an input or internal variable.

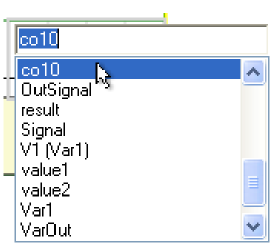
The following contact types are available from the LD toolbox:

- Direct Contact
- Reverse Contact
- Pulse Rising Edge Contact
- Pulse Falling Edge Contact

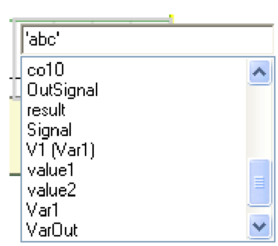
You can change the type of a contact at any time following its insertion.

When inserting contacts in POUs, you assign variables using the Variable Selector. Names of assigned variables are displayed above the contact elements within POUs. You replace existing variables by double-clicking the contact to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, you can single-click existing variables, then type literal values in the text boxes provided. When inserting literal values that being with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:




Type a literal value in the text box:




When selecting items such as local variables, global variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items.

**To insert a contact**

You can insert contacts from the Toolbox or using keyboard shortcuts.

1. From the Toolbox, drag the desired contact type into the language container and place it on the rung.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.

The contact and its associated variable name are displayed on the rung.

**To insert a parallel contact**

1. From the Toolbox, drag the branch element into the language container, placing it on the existing contact.

2. From the Toolbox, drag a contact element into the language container, placing it on the branch.

   The Variable Selector is displayed.

3. In the Variable Selector, select the required variable, then click **OK**.

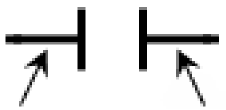
The contact and its associated variable name are displayed on the branch.

**To change the type of a contact**

- In the language container, select the contact, then press the space bar.

## Direct Contact

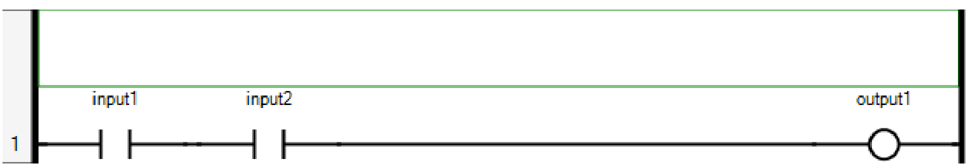
Direct contacts enable a Boolean operation between a connection line state and a Boolean variable.



Left Connection    Right Connection

The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the value of the variable associated with the contact.

### Example




(* ST Equivalence: *)

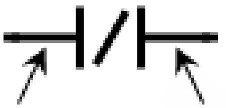
```
output1 := input1 AND input2;
```

### See Also

Contacts

## Reverse Contact

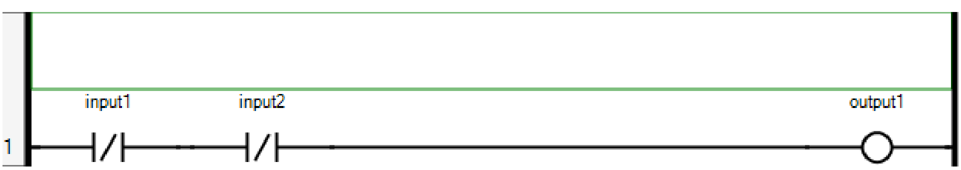
Reverse contacts enable a Boolean operation between a connection line state and the Boolean negation of a Boolean variable.



Left Connection    Right Connection

The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the Boolean negation of the value of the variable associated with the contact.

### Example




(* ST Equivalence: *)

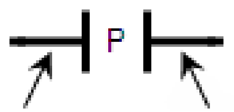
```
output1 := NOT (input1) AND NOT (input2);
```

### See Also

Contacts

## Pulse Rising Edge Contact

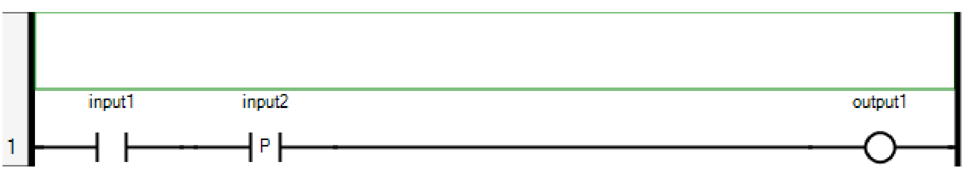
Pulse rising edge (positive) contacts enable a Boolean operation between a connection line state and the rising edge of a Boolean variable.




| | |
|---|---|
| Left Connection | Right Connection |

The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable rises from FALSE to TRUE. The state is reset to FALSE in all other cases.

### Example




(* ST Equivalence: *)

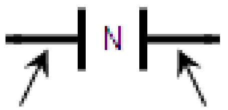
```
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)
```

### See Also

Contacts

## Pulse Falling Edge Contact

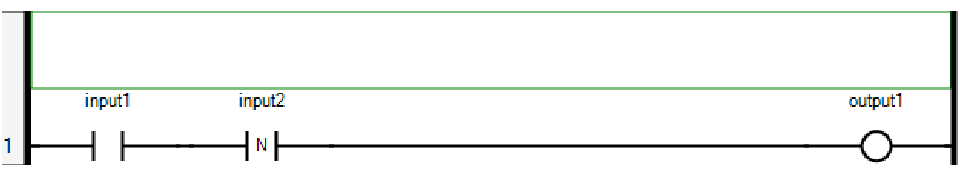
Pulse falling edge (negative) contacts enable a Boolean operation between a connection line state and the falling edge of a Boolean variable.




Left Connection    Right Connection

The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable falls from TRUE to FALSE. The state is reset to FALSE in all other cases.

### Example




(* ST Equivalence: *)

```
output1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev is the value of input2 at the previous cycle *)
```

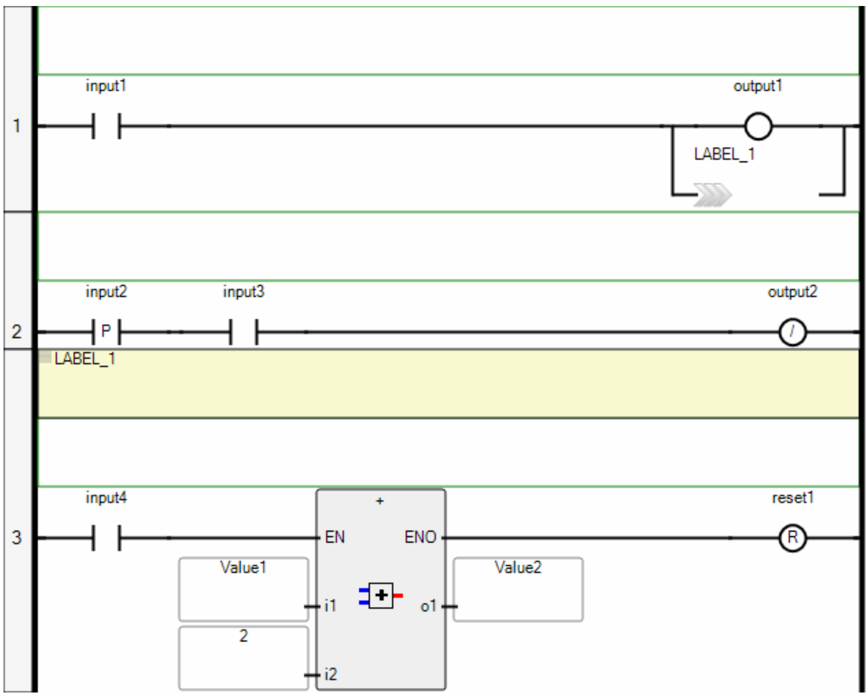
### See Also

Contacts

## Jumps

Conditional and unconditional jump elements enable controlling the execution of diagrams. You cannot place connections to the right of a jump element. When the connection on the left of the jump element has the TRUE Boolean state, the diagram execution proceeds at the label. The label and jump must have the same name.

### Example




**To insert a jump**

Before inserting jumps, define one or more labels within the program. You can insert jumps from the Toolbox or using keyboard shortcuts.

1. From the Toolbox, drag the jump element into the language container and place it on the rung.

2. In the language container, click the jump element, then select the required label name from the drop-down combo-box.

The jump is displayed on the rung with the required label name.

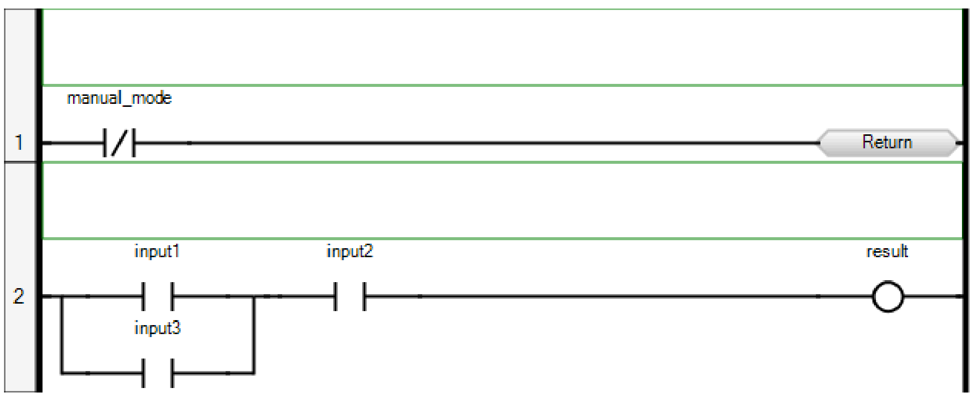
## Returns

You can use RETURN elements as outputs representing a conditional end of a diagram. You cannot place connections to the right of a RETURN element.

When the left connection line has the TRUE Boolean state, the diagram ends without executing the equations located on the next lines of the diagram.

When the LD diagram is a function, its name is associated with an output coil to set the return value (returned to the calling diagram).

### Example




(* ST Equivalence: *)

```
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;
```

**To insert a return**

You can insert returns from the Toolbox or using keyboard shortcuts.

- From the Toolbox, drag the return element into the language container, placing it on the rung.

The return element is displayed on the rung.

### See Also

Jumps

## Branches

Branches create alternative routing for connections. You can add parallel branches to elements on a rung.

**To insert a branch**

You can insert branches from the Toolbox or using keyboard shortcuts.

- From the Toolbox, drag the branch element into the language container and place in on the rung.

A parallel branch is displayed.

## LD Keyboard Shortcuts

The following keyboard shortcuts are available for use with the LD language. Some shortcuts do not apply or may differ while debugging.

| | |
|---|---|
| Ctrl+A | Selects all rungs (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while debugging) |
| Ctrl+V | Pastes elements saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl+X | Cuts the selected elements to the clipboard (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Ctrl+Alt+G | Enables/disables the grid in the language container |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+R | Toggles between Auto-Input and Manual-Input. Auto-Input automatically opens the Block Selector and Variable Selector (not available while debugging). |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctrl+I | Italicizes selected comment text (not available while debugging) |
| Ctrl+U | Underlines selected comment text (not available while debugging) |
| Enter | Calls the Variable/Block selector depending on the selected element (not available while debugging) |
| Space Bar | For coils or contacts, toggles between the available types (not available while debugging) |
| Ctrl+0 | Inserts a rung after a selected rung. When no rung is selected, a rung is added at the end of the rung list (not available while debugging). |
| Ctrl+Alt+0 | Inserts a rung before a selected rung. When no rung is selected, a rung is added at the end of the rung list (not available while debugging). |
| Ctrl+ 1 | Inserts a branch after a selected element (not available while debugging) |
| Ctrl+Alt+ 1 | Inserts a branch before a selected element (not available while debugging) |

| | |
|---|---|
| Ctrl+2 | Inserts a block after a selected element. When a branch is selected, a block is inserted on the branch (not available while debugging). |
| Ctrl+Alt+2 | Inserts a block before a selected element. When a branch is selected, a block is inserted on the branch (not available while debugging). |
| Ctrl+3 | Inserts a contact after a selected element. When a branch is selected, a contact is inserted on the branch (not available while debugging). |
| Ctrl+Alt+3 | Inserts a contact before a selected element. When a branch is selected, a contact is inserted on the branch (not available while debugging). |
| Ctrl+4 | When a rung or the last element on a rung is selected, inserts a coil at the end of the rung. When the last element selected on a rung is a branch, a coil is inserted on the branch (not available while debugging). |
| Ctrl+Alt+4 | When a rung or the last element on a rung is selected, inserts a coil at the end of the rung. When the last element selected on a rung is a branch, a coil is inserted on the branch (not available while debugging). |
| Ctrl+5 | When a rung or the last element on a rung is selected, inserts a jump at the end of the rung. When the last element selected on a rung is a branch, a jump is inserted on the branch (not available while debugging). |
| Ctrl+Alt+5 | When a rung or the last element on a rung is selected, inserts a jump at the end of the rung. When the last element selected on a rung is a branch, a jump is inserted on the branch (not available while debugging). |
| Ctrl+6 | When a rung or the last element on a rung is selected, inserts a return at the end of the rung. When the last element selected on a rung is a branch, a return is inserted on the branch (not available while debugging). |
| Ctrl+Alt+6 | When a rung or the last element on a rung is selected, inserts a return at the end of the rung. When the last element selected on a rung is a branch, a return is inserted on the branch (not available while debugging). |
| Ctrl+Page Up | Jumps to the top of the language container |
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Ctrl+Up Arrow | Slowly scrolls up. |
| Ctrl+Down Arrow | Slowly scrolls down. |

| | |
|---|---|
| Ctrl+Left Arrow | Slowly scrolls left. |
| Ctrl+Right Arrow | Slowly scrolls right. |
| Up Arrow | Moves up the elements. |
| Down Arrow | Moves down the elements. |
| Left Arrow | Moves to the left across the elements. |
| Right Arrow | Moves to the right across the elements. |
| Alt+Up Arrow | Selects the previous rung. When no element or rung is selected, selects the last rung. |
| Alt+Down Arrow | Selects the next rung. When no element or rung is selected, selects the first rung. |
| Alt+Left Arrow | Selects the rung of the selected element. When no element is selected, selects the first rung. |
| Alt+Right Arrow | Selects the rung of the selected element. When no element is selected, selects the first rung. |
| Shift+Up Arrow | Scrolls up |
| Shift+Down Arrow | Scrolls down |
| Shift+Left Arrow | Scrolls left |
| Shift+Right Arrow | Scrolls right |
| Delete | Removes a selected rung or element (not available while debugging) |

# ST Language

ST (Structured Text) is a high level structured language designed for automation processes. This language is mainly used to implement complex procedures that cannot be easily expressed with graphic languages. ST language is also used for the description of the actions within the Steps and conditions attached to the Transitions of the SFC Language.

**See Also**

ST Main Syntax
Debugging ST Programs

# ST Main Syntax

An ST program is a list of ST statements. Each statement ends with a semi-colon (";") separator. Names used in the source code (variable identifiers, constants, language keywords...) are separated with inactive separators (space character, end of line or tab stops) or by active separators, which have a well defined significance (for example, the ">" separator indicates a "greater than" comparison.

Comments enable the inclusion of non-executed information throughout code. You can insert comments anywhere in an ST program. Comments can run multiple lines and must begin with "(*" and end with "*)". You cannot use interleave comments, i.e., comments within comments.

When typing statements, a drop-down combo-box automatically lists the available items such as identifiers, operators, functions, and function blocks. The listed items are focused by typing letters, digits, and underscore characters.

The following are basic types of ST statements:

- assignment statement (variable := expression;)
- function call
- function block call
- selection statements (IF, THEN, ELSE, CASE...)
- iteration statements (FOR, WHILE, REPEAT...)
- control statements (RETURN, EXIT...)
- special statements for links with other languages

When entering ST syntax, basic coding is black while other items are displayed using customizable colors. The default colors for ST elements are the following:

- Comments are green
- The Editor background is white
- Identifiers are black

- Numbers are firebrick
- Operators are black
- POUs are blueviolet
- Punctuation marks are black
- Reserved words are fuchsia
- Strings of text are gray

Inactive separators between active separators, literals, and identifiers increase ST program legibility. ST inactive separators are the following: space (blank), tabs and end of line. You can place end of lines anywhere in a program. The following rules apply to using inactive separators:

- Write one statement on one line
- Use tabs to indent complex statements
- Insert comments to increase legibility of lines or paragraphs

### Example

**Low Readability**

```
imax := max_ite; cond := X12;
if not(cond (* alarm *)
then return; end_if;
for i (* index *) := 1 to max_ite
do if i <> 2 then Spcall();
end_if; end_for;
(* no effect if alarm *)
```

**High Readability**

```
(* imax : number of iterations *)
(* i: FOR statement index *)
(* cond: process validity *)
imax := max_ite;
cond := X12;
if not (cond) then
     return;
end_if;
(* process loop *)
for i := 1 to max_ite do
     if i <> 2 then
     Spcall ();
     end_if;
end_for;
```

**To customize the default display settings for ST programs**

1. From the Tools menu, click **Options**.
2. From the Options dialog, expand **IEC Languages**, then click **Structured Text (ST)**.
3. Expand the respective category, customize the required setting, then click **OK**.

The customized settings are now the default values for ST programs.

**To customize the display settings for the current ST program**

1. From the View menu, click **Properties Window**.

   The Properties Window is displayed.
2. Select the ST Container.
3. From the Properties Window you can:

- Customize the font for the required item by clicking [...]. The Font dialog box is displayed enabling customization of the font, text size, bold, italic, strikeout, and underline styles.
- Customize the text color for the required items. The possible colors are custom, web, and system colors.

The customized settings only affect the current ST program.

# Expressions and Parentheses

ST expressions combine ST operators and variable or constant operands. For each single expression (combining operands with one ST operator), the type of the operands must be the same. This single expression has the same data type as its operands, and can be used in a more complex expression. For example:

| | |
|---|---|
| (boo_var1 AND boo_var2) | has BOOL type |
| not (boo_var1) | has BOOL type |
| (sin (3.14) + 0.72) | has DINT type |
| (t#1s23 + 78) | is an invalid expression |

Parentheses are used to isolate sub parts of an expression and to explicitly order the priority of operations. When no parentheses are given for a complex expression, the operation sequence is implicitly given by the default priority between ST operators.

| **Precedence** | **Operators** | **Symbols** |
|---|---|---|
| 1 (Highest) | Function evaluation | *identifier*(*arguement list*)<br>For example: `MAX(X,Y)` |
| 2 | Negation | `-` |
| | Complement | `NOT` |
| 3 | Multiplication | `*` |
| | Division | `/` |
| 4 | Addition | `+` |
| | Subtraction | `-` |
| 5 | Comparison | `<, >, <=, >=` |
| 6 | Equality | `=` |
| | Inequality | `<>` |
| 7 | Boolean AND | `&, AND` |
| 8 | Boolean Exclusive OR | `XOR` |
| 9 (Lowest) | Boolean OR | `OR` |

**Examples:**

| | | |
|---|---|---|
| 2 + 3 * 6 | equals 2+18=20 | because multiplication operator has a higher priority |
| (2 + 3) * 6 | equals 5*6=30 | priority is given by parenthesis |

# Calling Functions

The ST programming language enables calling functions. Function calls can be used in any expression.

**Name:** name of the called function written in IEC 61131-3 language or in "C"

**Meaning:** calls a ST, LD or FBD functions or a "C" function and gets its return value

**Syntax:** <**variable**> **:=** <**funct**> **(**<**par1**>**,** ... <**parN**> **);**

**Operands:** The type of return value and calling parameters must follow the interface defined for the function.

**Return value:** value returned by the function

When setting the value of the return parameter in the body of a function, assign the return parameter using the same name as the function:
FunctionName := <expression>;

## Example

Example1: IEC 61131-3 function call

```
(* Main ST program *)
(* gets an integer value and converts it into a limited time value *)
dint_timeprog := SPlimit ( tprog_cmd );
appl_timer := TMR (dint_timeprog * 100);

(* Called FBD function named 'SPlimit' *)
```

Example2: "C" function call – same syntax as for IEC 61131-3 function calls

```
(* Functions used in complex expressions: min, max, right, mlen and
left are standard "C" functions *)
limited_value := min (16, max (0, input_value) );
rol_msg := right (message, mlen (message) - 1) + left (message, 1);
```

# Calling Function Blocks

The ST programming language enables calling function blocks. Function block calls can be used in any expression.

**Name:** name of the function block instance

**Meaning:** calls a ST, LD, or FBD function block or a "C" function block and gets its return parameters

**Syntax:** **(* call of the function block *)**
<**blockname**> **(** <**p1**>**,** <**p2**> ... **);**
**(* gets its return parameters *)**
<**result**> **:=** <**blockname**>**.** <**ret_param1**>**;**
...
<**result**> **:=** <**blockname**>**.** <**ret_paramN**>**;**

**Operands:** parameters are expressions which match the type of the parameters specified for that function block

**Return value:** See Syntax to get the return parameters.

When setting the value of the return parameter in the body of a function block, assign the return parameter using its name concatenated with the function block name:
FunctionBlockName.OutputParaName := <expression>;

### Example

```
(* ST program calling a function block *)

(* declare the instance of the block in the dictionary: *)
(* trigb1 : block R_TRIG - rising edge detection *)

(* Function block activation from ST language *)
trigb1 (b1);
(* return parameters access *)
If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;
```

# Debugging ST Programs

When debugging ST programs, you can monitor the output values of elements by viewing the dictionary instances. When the program is debugging you can choose to perform one of the following operations:

- Switch execution to real-time mode
- Switch execution to cycle-to-cycle mode
- Execute one cycle

**To switch execution to real-time mode**

- From the Target Execution toolbar, click .

The POU executes in real-time mode.

**To switch execution to cycle-to-cycle mode**

- From the Target Execution toolbar, click .

The POU executes in cycle-to-cycle mode.

**To execute one cycle**

- From the Target Execution toolbar, click .

The POU executes one device cycle.

# ST Basic Elements and Statements

The basic elements and statements of the ST language are the following:

- Assignments
- CASE Statement
- EXIT Statement
- FOR Statement
- IF-THEN-ELSIF-ELSE-END_IF Statement
- REPEAT Statement
- RETURN Statement
- WHILE Statement

**See Also**
ST Main Syntax

## Assignments

**Name:** :=

**Meaning:** Assigns a variable to an expression

**Syntax:** <variable> := <any_expression> ;

**Operands:** Variable must be an internal or output variable and the expression must have the same type

The expression can be a call to a function.

### Example

```
(* ST program with assignments *)

(* variable <<= variable *)
bo23 := bo10;

(* Variable <<= expression *)

bo56 := bx34 OR alrm100 & (level >= over_value);
result := (100 * input_value) / scale;

(* assignment with function call *)
limited_value := min (16, max (0, input_value) );
```

**To insert an Assignment**

- In the language container, type **:=**.

## CASE Statement

**Name:** **CASE ... OF ... ELSE ... END_CASE**

**Meaning:** executes one of several lists of ST statements
selection is made according to an integer expression

**Syntax:** **CASE <integer_expression> OF**
<value> : <statements> ;
<value> , <value> : <statements> ;
...
**ELSE**
<statements> ;
**END_CASE;**

CASE values must be integer constant expressions. Several values, separated by commas, can lead to the same list of statements. The ELSE statement is optional.

### Example

```
(* ST program using CASE statement *)

CASE error_code OF
  255: err_msg := 'Division by zero';
fatal_error := TRUE;
  1: err_msg := 'Overflow';
  2, 3: err_msg := 'Bad sign';
ELSE
  err_msg := 'Unknown error';
END_CASE;
```

**To insert a CASE**

- From the Toolbox, drag the **CASE** element into the language container.

## EXIT Statement

**Name:** **EXIT**

**Meaning:** exit from a FOR, WHILE or REPEAT iteration statement

**Syntax:** **EXIT**;

The EXIT is commonly used within an IF statement, inside a FOR, WHILE or REPEAT block.

### Example

```
(* ST program using EXIT statement *)
(* this program searches for a character in a string *)

length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
  code := ascii (message, index);
  IF (code = searched_char) THEN
     found := YES;
     EXIT;
  END_IF;
END_FOR;
```

**To insert an EXIT**

- In the language container, type **EXIT**.

## FOR Statement

**Name:** **FOR ... TO ... BY ... DO ... END_FOR**

**Meaning:** executes a limited number of iterations, using an integer index variable

**Syntax:** **FOR <index> := <mini> TO <maxi> BY <step> DO**
<statement> ;
<statement> ;
**END_FOR;**

**Operands:** **index**: internal integer variable increased at each loop
**mini**: initial value for index (before first loop)
**maxi**: maximum allowed value for index
**step**: index increment at each loop

The [ BY step ] statement is optional. If not specified, the increment step is 1

**Warning:** Because the virtual machine is a synchronous system, input variables are not refreshed during FOR iterations.

This is the "WHILE" equivalent of a FOR statement:

```
index := mini;
while (index <= maxi) do
  <statement> ;
  <statement> ;
  index := index + step;
end_while;
```

### Example

```
(* ST program using FOR statement *)
(* this program extracts the digit characters of a string *)

length := mlen (message);
target := ''; (* empty string *)
FOR index := 1 TO length BY 1 DO
  code := ascii (message, index);
  IF (code >= 48) & (code <= 57) THEN
     target := target + char (code);
  END_IF;
END_FOR;
```

**To insert a FOR**

- From the Toolbox, drag the **FOR** element into the language container.

## IF-THEN-ELSIF-ELSE-END_IF Statement

**Name:** **IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF**

**Meaning:** executes one of several lists of ST statements
selection is made according to the value of a Boolean expression

**Syntax:** **IF** <**Boolean_expression**> **THEN**
<statement> ;
<statement> ;
...
**ELSIF** <**Boolean_expression**> **THEN**
<statement> ;
<statement> ;
...
**ELSE**
<statement> ;
<statement> ;
...
**END_IF**;

The ELSE and ELSIF statements are optional. If the ELSE statement is not written, no instruction is executed when the condition is FALSE. You can use the ELSIF statement more than once. The ELSE statement, if used, must appear only once at the end of the 'IF, ELSIF...' sequence.

### Example

```
(* ST program using IF statement *)

IF manual AND not (alarm) THEN
  level := manual_level;
  bx126 := bi12 OR bi45;
ELSIF over_mode THEN
  level := max_level;
ELSE
level := (lv16 * 100) / scale;
END_IF;
```

```
(* IF structure without ELSE *)
If overflow THEN
  alarm_level := true;
END_IF;
```

**To insert an IF-THEN-ELSIF-ELSE-END_IF**

- From the Toolbox, drag the **IF THEN ELSE** element into the language container.

## REPEAT Statement

**Name:** **REPEAT ... UNTIL ... END_REPEAT**

**Meaning:** iteration structure for a group of ST statements
the "continue" condition is evaluated AFTER any iteration

**Syntax:** **REPEAT**
<statement> ;
<statement> ;
...
**UNTIL <Boolean_condition>**
**END_REPEAT ;**

**Warning:** Because the virtual machine is a synchronous system, input variables are not refreshed during REPEAT iterations. The change of state of an input variable cannot be used to describe the ending condition of a REPEAT statement.

### Example

```
(* ST program using REPEAT statement *)

(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

str := ''; (* empty string *)
nbchar := 0;
IF ComIsReady ( ) THEN
  REPEAT
     str := str + ComGetChar ( );
     nbchar := nbchar + 1;
  UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )
  END_REPEAT;
END_IF;
```

**To insert a REPEAT**

- From the Toolbox, drag the **REPEAT** element into the language container.

## RETURN Statement

**Name:** **RETURN**

**Meaning:** terminates the execution of the current program

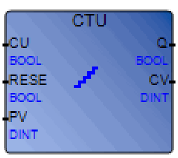
**Syntax:** **RETURN** ;

**Operands:** (none)

In an SFC action block, the RETURN statement indicates the end of the execution of that block only.

### Example

(* FBD specification of the program: programmable counter *)




```
(* ST implementation of the program, using RETURN statement *)

If NOT (CU) then
  Q := false;
  CV := 0;
  RETURN; (* terminates the program *)
end_if;

if RESET then
  CV := 0;
else
  if (CV < PV) then
     CV := CV + 1;
  end_if;
end_if;
Q := (CV >= PV);
```

**To insert a RETURN**

- In the language container, type **RETURN**.

## WHILE Statement

**Name:** **WHILE ... DO ... END_WHILE**

**Meaning:** iteration structure for a group of ST statements
the "continue" condition is evaluated BEFORE any iteration

**Syntax:** **WHILE <Boolean_expression> DO**
<statement> ;
<statement> ;
...
**END_WHILE ;**

**Warning:** Since the virtual machine is a synchronous system, input variables are not refreshed during WHILE iterations. The change of state of an input variable cannot be used to describe the condition of a WHILE statement.

### Example

```
(* ST program using WHILE statement *)

(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

str := ''; (* empty string *)
nbchar := 0;

WHILE ((nbchar < 16) & ComIsReady ( )) DO
  str := str + ComGetChar ( );
  nbchar := nbchar + 1;
END_WHILE;
```

**To insert a WHILE**

- From the Toolbox, drag the **WHILE** element into the language container.

## ST Extensions

The following statements and functions are extensions of the ST language:

| | |
|---|---|
| TSTART | starts a timer |
| TSTOP | stops a timer |

The following statements and functions are available to control the execution of SFC child programs. You can use these within action blocks written in ST for SFC steps.

| | |
|---|---|
| GSTART | starts an SFC program or function block |
| GFREEZE | freezes an SFC program |
| GKILL | terminates an SFC program |
| GSTATUS | gets current status of an SFC program |
| GRST | restarts a frozen SFC program or function block |

**Warning:** These functions are not part of the IEC 61131-3 standard.

Simple equivalents for the GSTART and GKILL statements are available using the following syntax in an SFC step:

- child_name with the S qualifier (* equivalent to GSTART(child_name); *)
- child_name with the R qualifier (* equivalent to GKILL(child_name); *)

The following fields enable accessing the status of an SFC step or child (from its parent):

| | |
|---|---|
| StepName.x | Boolean value that represents the **activity of the Step** |
| StepName.t | time elapsed since the last activation of the step: **activity duration** ("**StepName**" represents the name of the SFC step) |
| ChildName.__S1.x | Boolean value that represents the **activity of the child** |
| ChildName.__S1.t | time elapsed since the last activation of the step: **activity duration** ("**ChildName**" represents the name of the SFC child) |

## TSTART Statement

**Name:** **TSTART**

**Meaning:** Starts the counting of a timer variable. The timer value is not modified by the TSTART command, i.e., the counting starts from the current value of the timer.

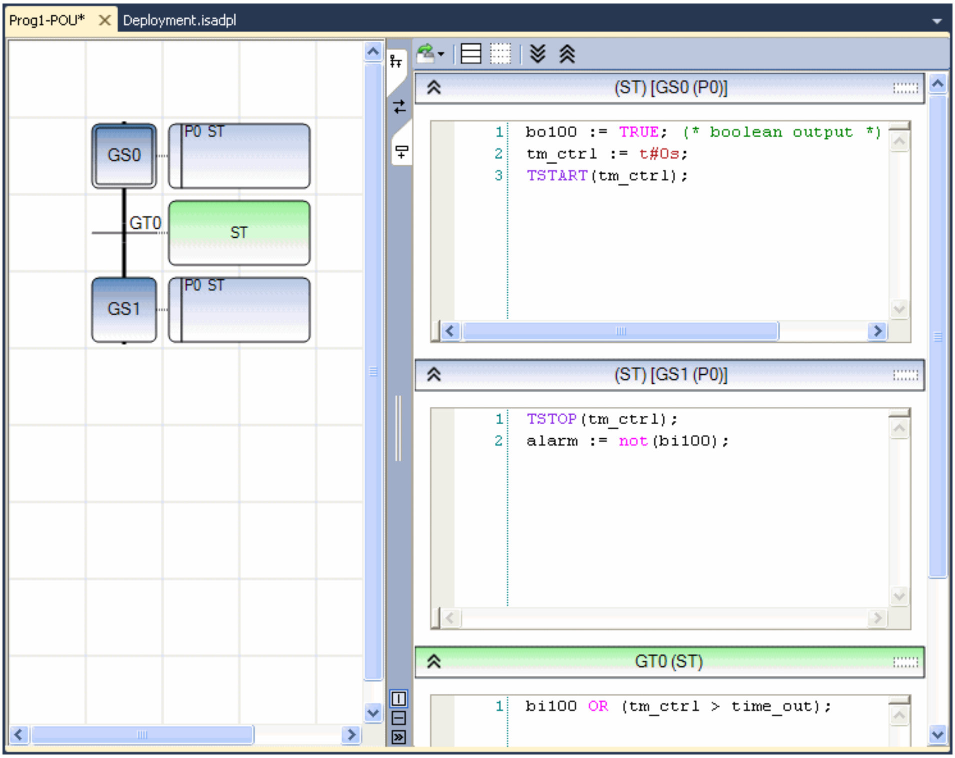
**Syntax:** **TSTART** ( <*timer_variable*> );

**Operands:** Any inactive timer variable

**Return value:** (none)

### Example

(* SFC program using TSTART and TSTOP statements)




**To insert a TSTART Statement**

- In the language container, type **TSTART**.

# TSTOP Statement

**Name:** **TSTOP**

**Meaning:** Stops updating a timer variable. The timer value is not modified by the TSTOP command.

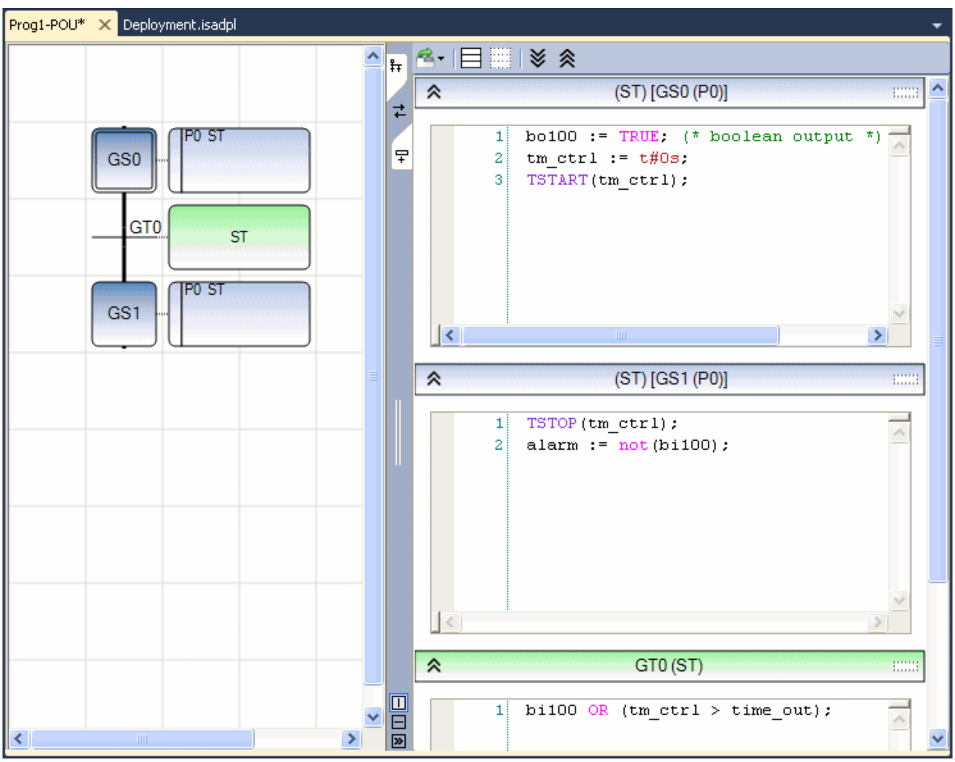
**Syntax:** **TSTOP** ( <*timer_variable*> );

**Operands:** Any active timer variable

**Return value:** (none)

## Example

(* SFC program using TSTART and TSTOP statements)



**To insert a TSTOP Statement**

- In the language container, type **TSTOP**.

## GSTART Statement in SFC Action

**Name:** **GSTART**

**Meaning:** Starts an SFC child program or function block by placing a token into each of its initial steps. The abbreviated syntax is equivalent to an SFC Child action block having the S qualifier. The extended syntax only applies to SFC child function blocks.

**Syntax:** **GSTART** ( <*child_name*> );
or
**GSTART** ( <*child_name,step_name,input1,input2,...inputn*> )
where
*child_name* represents the name of the SFC child POU
*step_name* represents the name of the active step. *step_name* must be preceded by two underscore characters (e.g., __S1)
*input1,input2,...inputn* indicate the values of the input parameters of the SFC child POU

**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** (none)

Children of the child program are not automatically started by the GSTART statement. Since GSTART is not part of the IEC 61131-3 standard, it is preferable to use the S qualifier attached to the child name.

**To insert a GSTART**

- In the language container, type **GSTART**.

## GFREEZE Statement in SFC Action

**Name:** **GFREEZE**

**Meaning:** freezes a child SFC (program or function block); suspends its execution. The suspended SFC POU can then be restarted using the GRST statement.

**Syntax:** **GFREEZE (** <*child_name*> **);**
where
*child_name* represents the name of the SFC child POU

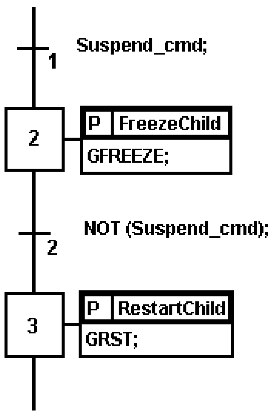
**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** (none)

Children of the child program are automatically frozen along with the specified program.

GFREEZE is not part of the IEC 61131-3 standard.

### Example




**To insert a GFREEZE**

- In the language container, type **GFREEZE**.

## GKILL Statement in SFC Action

**Name:** **GKILL**

**Meaning:** Terminates a child SFC program by removing the Tokens currently existing in its Steps. The syntax is equivalent to an SFC Child action block having the R qualifier.

**Syntax:** **GKILL** ( <*child_name*> );
where
*child_name* represents the name of the SFC child POU

**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** (none)

Children of the child program are automatically terminated with the specified program.

Since GKILL is not part of the IEC 61131-3 standard, it is preferable to use the R qualifier attached to the child name.

**To insert a GKILL**

- In the language container, type **GKILL**.

## GSTATUS Statement in SFC Action

**Name:** **GSTATUS**

**Meaning:** returns the current status of an SFC program

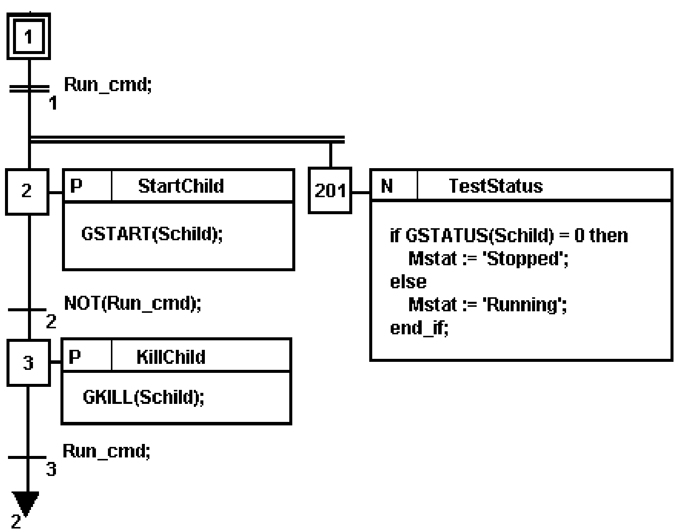
**Syntax:** **<var> := GSTATUS (** <*child_name*> **);**
where
*child_name* represents the name of the SFC child POU

**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** 0 = Program is inactive (killed)
1 = Program is active (started)
2 = Program is frozen

GSTATUS is not part of the IEC 61131-3 standard.

### Example



**To insert a GSTATUS**

- In the language container, type **GSTATUS**.

## GRST Statement in SFC Action

**Name:** **GRST**

**Meaning:** restarts a child SFC program frozen by the GFREEZE statement: all the tokens removed by GFREEZE are restored. The extended syntax only applies to SFC child function blocks.

**Syntax:** **GRST** ( <*child_name*> );
or
**GRST** ( <*child_name,input1,input2,...inputn*> );
where
*child_name* represents the name of the SFC child POU
*input1,input2,...inputn* indicate the value of the input parameter of the SFC child POU

**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** (none)

The GRST statement automatically restarts children of the child program.

GRST is not part of the IEC 61131-3 standard.

**To insert a GRST**

- In the language container, type **GRST**.

## ST Keyboard Shortcuts

The following keyboard shortcuts are available for use with the ST language. Some shortcuts do not apply or may differ while debugging.

| | |
|---|---|
| Ctrl+A | Selects the entire document (not available while debugging) |
| Ctrl+C | Copies the selected text to the clipboard (not available while debugging) |
| Ctrl+Insert | Copies the selected text to the clipboard (not available while debugging) |
| Ctrl+V | Pastes text saved on the clipboard to the insertion point (not available while debugging) |
| Shift+Insert | Pastes text saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl+X | Cuts the selected text to the clipboard (not available while debugging) |
| Shift+Delete | Cuts the selected text to the clipboard (not available while debugging) |
| Ctrl+L | Cuts the current line to the clipboard (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Shift+Z | Redoes the previous command (not available while debugging) |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Insert | Toggles between the overwrite/insert typing mode |
| Shift+Enter | Inserts a line break. While debugging, when the insertion point is on a variable it opens the Write Logical Value dialog box. |
| Ctrl+Enter | Inserts a line above the current line. While debugging, when the insertion point is on a variable it opens the Write Logical Value dialog box. |

| | |
|---|---|
| Ctrl+Shift+Enter | Inserts a line below the current line. While debugging, when the insertion point is on a variable it opens the Write Logical Value dialog box. |
| Ctrl+Shift+T | Transposes the current and previous word (not available while debugging) |
| Ctrl+Shift+Alt+T | Transposes the current and next line (not available while debugging) |
| Ctrl+Space | Displays a drop-down combo-box listing available items such as variables, operators, functions, and function blocks. You can filter displayed items by typing letters, digits, and underscore characters. (not available while debugging) |
| Ctrl+Shift+Space | Displays a drop-down combo-box listing available items such as variables, operators, functions, and function blocks. You can filter displayed items by typing letters, digits, and underscore characters. (not available while debugging) |
| Ctrl+Shift+U | Changes the selected text into uppercase (not available while debugging) |
| Ctrl+U | Changes the selected text into lowercase (not available while debugging) |
| Up Arrow | Moves up lines and characters |
| Down Arrow | Moves down lines and characters |
| Left Arrow | Moves left across lines and characters |
| Right Arrow | Moves right across lines and characters |
| Ctrl+Left Arrow | Moves to the previous statement or word |
| Ctrl+Right Arrow | Moves to the next statement or word |
| Home | Jumps to the start of the line |
| End | Jumps to the end of the line |
| Ctrl+Home | Jumps to the start of the document |
| Ctrl+End | Jumps to the end of the document |
| Page Up | Jumps to the top of the visible code |
| Page Down | Jumps to the bottom of the visible code |
| Ctrl+Page Up | Jumps to the top of the visible code |

| | |
|---|---|
| Ctrl+Page Down | Jumps to the bottom of the visible code |
| Ctrl+Up Arrow | Scrolls up |
| Ctrl+Down Arrow | Scrolls down |
| Shift+Up Arrow | Selects up |
| Shift+Down Arrow | Selects down |
| Shift+Left Arrow | Selects left |
| Shift+Right Arrow | Selects right |
| Ctrl+Shift+Left Arrow | Selects to the previous statement or word |
| Ctrl+Shift+Right Arrow | Selects to the next statement or word |
| Shift+Home | Selects from the insertion point until the start of the line |
| Shift+End | Selects from the insertion point until the end of the line |
| Ctrl+Shift+Home | Selects from the insertion point until the start of the document |
| Ctrl+Shift+End | Selects from the insertion point until the end of the document |
| Ctrl+Shift+Page Up | Selects from the insertion point until the top of the visible code |
| Ctrl+Shift+Page Down | Selects from the insertion point until the end of the visible code |
| Ctrl+Shift+W | Selects the next word |
| Shift+Alt+Up Arrow | Selects the current and previous lines |
| Shift+Alt+Down Arrow | Selects the current and next lines |
| Shift+Alt+Left Arrow | Selects left on the current line |
| Shift+Alt+Right Arrow | Selects right on the current line |
| Ctrl+Shift+Alt+Left Arrow | Selects available columns in lines of code from the left to right |
| Ctrl+Shift+Alt+Right Arrow | Selects available columns in lines of code from the right to left |
| Escape | Deselects the selected text |
| Ctrl+I | Opens the Variable Selector. While debugging, opens the Variable Monitoring dialog box. |
| Ctrl+Shift+I | Opens the Variable Selector. While debugging, opens the Variable Monitoring dialog box. |
| Ctrl+R | Opens the Block Selector. When the insertion point is on a variable during debugging, it is selected. |

| | |
|---|---|
| Ctrl+Alt+R | Opens the Block Selector. When the insertion point is on a variable during debugging, it is selected. |
| Ctrl+Shift+Alt+R | Opens the Block Selector. When the insertion point is on a variable during debugging, it is selected. |
| Delete | Removes the character on the right (not available while debugging) |
| Ctrl+Shift+L | Removes the current line (not available while debugging) |
| Ctrl+Delete | Removes the next word in the current line (not available while debugging) |
| Ctrl+Backspace | Removes the previous word in the current line (not available while debugging) |
| Backspace | Removes the character on the left (not available while debugging) |
| Shift+Backspace | Removes the character on the left (not available while debugging) |

# Language Reference

The language reference includes information about the usage and limitations of various project elements and other aspects:

- Programs
- Functions
- Function Blocks
- Execution Rules
- Reserved Keywords
- Variables
- Directly Represented Variables
- Defined Words
- Data Types

# Programs

Programs, also known as POUs, are logical programming units describing operations between variables of a process. Programs describe either sequential or cyclic operations. Cyclic programs are executed at each target system cycle. Sequential programs, representing sequential operations, are grouped together. The execution of sequential programs has a dynamic behavior.

Programs before and after sequential programs describe cyclic operations. Cyclic programs are not time-dependent. Cyclic programs are systematically executed at the beginning of each run-time cycle. Main sequential programs (at the top of the hierarchy) are executed according to their respective dynamic behavior.

| | |
|---|---|
| Begin | Cyclic operations (FDB, LD, ST) |
| Sequential | Sequential operations (SFC, SFC child) |
| End | Cyclic operations (FDB, LD, ST) |

Programs located at the beginning of a cycle (before sequential programs) typically describe preliminary operations on input devices to build high level filtered variables. Sequential programs frequently use these variables. Programs located at the end of the cycle (after sequential programs) typically describe security operations on the variables operated on by sequential programs, before sending values to output devices.

Programs are described using the available graphic or literal languages. You specify the programming language when creating a program; you cannot change the programming language for an existing program.

POUs defined as programs are executed on the target system respecting the order shown in the Programs section.

Programs are linked together in a hierarchical tree. Those placed at the top of the hierarchy are activated by the system. Child-programs (lower level of the hierarchy) are activated by their parent.

POUs (programs, functions, and function blocks) within a project and dependency libraries must have unique names. These names can have up to eight (8) characters and must begin with a letter followed by letters, digits, and single underscores.

Projects can contain up to 255 programs.

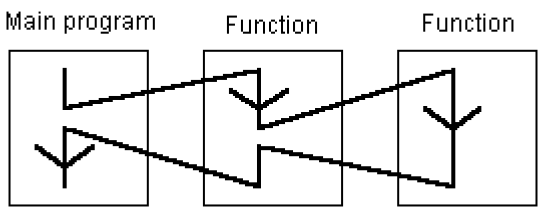
**See Also**

Execution Rules

# Functions

Functions are POUs having one or more input parameters and one output parameter. A function can be called by a program, a function or a function block. A function has no instance meaning that local data is not stored and is usually lost from one call to the other.

The execution of a function is driven by its parent program. Therefore, the execution of the parent program is suspended until the function ends:




Any POU of any section can call one or more functions. A function can have local variables.

**ISaGRAF** does not support recursivity during function calls. When a function of the *Functions* section is called by itself or one of its called functions, a build error occurs. Furthermore, functions do not store the local values of their local variables. Since functions are not instantiated, these cannot call function blocks.

The interface of a function must be explicitly defined with a type and a unique name for each of its calling (input) parameters or return (output) parameter. Functions can have up to 31 calling parameters and one return parameter. Return parameters can only have Boolean, Real, or Time data types.

POUs (programs, functions, and function blocks) within a project and dependency libraries must have unique names. Function names can have up to eight (8) characters and must begin with a letter followed by letters, digits, and single underscores. Functions can have a maximum of 32 parameters (31 inputs and one output). Parameter names have a maximum of 32 characters and must begin with a letter followed by letters, digits, and single underscores.

## Function Blocks

Function blocks are POUs having multiple input and output parameters. These are instantiated meaning local variables of a function block are copied for each instance. When calling a function block in a program, you actually call the instance of the block where the same code is called but the data used is that which has been allocated to the instance. The values of the variables of an instance are stored from one cycle to the other.

Function blocks can be called by any POU in the project. Function blocks can call functions or other function blocks.

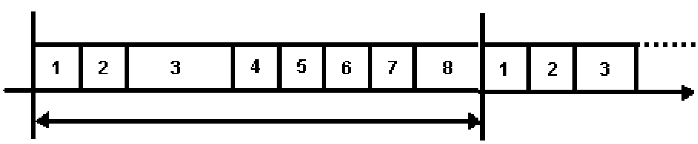
The interface of a function block must be explicitly defined with a type and a unique name for each of its calling (input) parameters or return (output) parameters. Function blocks can have more than one output parameter. The value of a return parameter for a function block differs for the various programming languages.

POUs (programs, functions, and function blocks) within a project and dependency libraries must have unique names. Function block names can have up to eight (8) characters and must begin with a letter followed by letters, digits, and single underscores. Function blocks can have a maximum of 32 parameters. Parameter names have a maximum of 32 characters and must begin with a letter followed by letters, digits, and single underscores.

# Execution Rules

The execution of a control application is a synchronous system where a clock triggers all operations for a device. The basic duration of the clock is called the cycle timing for a device.

1. Scan input variables.
2. Process "Begin" section programs.
3. Process "Sequential" section programs according to execution rules.
4. Process "End" section rules.
5. Process Modbus messages.
6. Update output devices.
7. Save retained values.
8. Sleep until next cycle.




When a cycle time is specified, a device waits until this time has elapsed before starting the execution of a new cycle. The POUs execution time varies depending on the size of the application. When a cycle exceeds the specified time, the loop continues to execute the cycle but sets an overrun flag. In such a case, the application no longer runs in real time.

When a cycle time is not specified, a device performs all programs then restarts a new cycle without waiting.

# Reserved Keywords

Reserved keywords are unavailable for use as names of POUs or variables.

| | |
|---|---|
| **A** | ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN, |
| **B** | BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE, |
| **C** | CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS, |
| **D** | DATE, DATE_AND_TIME, DATE_AND_TIME_TO_DATE, DATE_AND_TIME_TO_TIME_OF_DATE, DELETE, DINT, DIV, DO, DT, DWORD, |
| **E** | ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT, |
| **F** | FALSE, FEDGE, FIND, FOR, FUNCTION, |
| **G** | GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT, |
| **I** | IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME, |
| **J** | JMP, JMPC, JMPCN, JMPN, JMPNC, |
| **L** | LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD, |
| **M** | MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX, |
| **N** | NE, NOT, |
| **O** | OF, ON, OPERATE, OR, OR_MASK, ORN, |
| **P** | PROGRAM |
| **R** | R, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL, REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REDGE, REPEAT, REPLACE, RESOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR, |

| | |
|---|---|
| **S** | S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD, STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME, STRUCT, SUB, SYS_ERR_READ, SYS_ERR_TEST, SYS_INITALL, SYS_INITANA, SYS_INITBOO, SYS_INITTMR, SYS_RESTALL, SYS_RESTANA, SYS_RESTBOO, SYS_RESTTMR, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO, SYS_SAVTMR, SYS_TALLOWED, SYS_TCURRENT, SYS_TMAXIMUM, SYS_TOVERFLOW, SYS_TRESET, SYS_TWRITE, SYSTEM, |
| **T** | TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL, TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TMR, TO, TOD, TRUE, TSTART, TSTOP, TYPE, |
| **U** | UDINT, UINT, ULINT, UNTIL, USINT, |
| **V** | VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, VAR_OUTPUT |
| **W** | WHILE, WITH, WORD |
| **X** | XOR, XOR_MASK, XORN |

# Variables

The scope of variables can be local to a POU or global to a device. Local variables are available for use within one POU only. Global variables are available for use within any POU of the device.

- Name, limited to 32 characters beginning with a letter followed by letters, digits, and single underscore characters. These names cannot have two consecutive underscore characters.
- Data Type, possible values are Boolean, Double Integer, Real, Time, Message, function blocks, and one-dimensional arrays. Arrays are only available for the BOOL, DINT, REAL, and TIME data types; these are not available for the MESSAGE type.
- Logical Value, available when online. The displayed value differs depending on the direction of the variable: inputs are locked values and outputs are updated by the running TIC code.
- Physical Value, available when online. The displayed value differs depending on the direction of the variable: inputs are updated by the field value and outputs are locked.
- String Size, for message type variables, indicates the maximum length. String capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string.
- Dimension, the size (number of elements) of an array. Arrays can ony have one dimension. Arrays can have a maximum of 255 elements. For example, [6] represents a one-dimensional array containing elements from 0 to 5.
- Wiring, (read-only cell) generated by the I/O wiring tool indicating the I/O channel to which the variable is wired. You can only wire global variables. Uses the syntax of Directly Represented Variables.
- Attribute, property of a variable indicating its read and write access rights. Possible values are read, write, and read-write.
- Direction, for I/O wiring, wired variables can only be global. The direction of a variable affects the logical value and physical value.

- Modbus Address, modbus address of the variable. The format is hexadecimal and the value ranges from 0001 to FFFF.
- Retained, the indication of whether the value of the variable is saved by the virtual machine at each cycle. For details on retaining, i.e., backing up, variables, refer to the SYSTEM operator. Possible values are Yes or No.
- Initial Value, value held by a variable when the virtual machine starts the execution of the device. The initial value of a variable can be the default value, a value given by the user when the variable is defined or the value of the retain variable after the virtual machine has stopped. You can set initial values for POU variables and global variables.
- Unit, string identifying the physical unit at debug time. Free-format text.
- Comment, user-defined free format text

Although function block instances are declared using variables, these variables do not follow rules applying to elementary or derived type variables. These variables can only have the var direction and the read-write attribute.

## Directly Represented Variables

The system enables the use of directly represented variables in the source of programs to represent a free channel. Free channels are those not linked to a declared I/O variable. The identifier of a directly represented variable always begins with the "%" character.

The naming conventions of a directly represented variable for a channel of a single board. "***s***" is the slot number of the board. "***c***" is the number of the Channel:

| | |
|---|---|
| **%IXs.c** | free channel of a Boolean input board |
| **%IDs.c** | free channel of an integer input board |
| **%ISs.c** | free channel of a message input board |
| **%QXs.c** | free channel of a Boolean output board |
| **%QDs.c** | free channel of an integer output board |
| **%QSs.c** | free channel of a message output board |

The naming conventions of a directly represented variable for a channel of a complex equipment. "s" is the slot number of the equipment. "b" is the index of the single board within the complex equipment. "c" is the number of the channel:

| | |
|---|---|
| **%IXs.b.c** | free channel of a Boolean input board |
| **%IDs.b.c** | free channel of an integer input board |
| **%ISs.b.c** | free channel of a message input board |
| **%QXs.b.c** | free channel of a Boolean output board |
| **%QDs.b.c** | free channel of an integer output board |
| **%QSs.b.c** | free channel of a message output board |

### Example

%QX1.6 6th channel of the board #1 (Boolean output)
%ID2.1.7 7th channel of the board #1 in the equipment #2 (integer input)

# Defined Words

**ISaGRAF** supports the use of identifier names, called defined words. When building, defined words are replaced by the variables and expressions these represent. Defined words can have the following:

- common scope, i.e., available for use in any project on the computer
- global scope, i.e., available for use in any POU of a project
- local scope, i.e., available for use in only one POU of a project

For POUs, a defined word can replace literal expressions, boolean expressions, reserved keywords, or complex ST expressions.

The following are examples of defined words:

Defined Words

| Name | Equivalent | Comment |
|---|---|---|
| OK | (auto_mode AND NOT (alarm)) | |
| PI | 3.14159 | |
| YES | TRUE | |

When such an equivalence is defined, its identifier is available anywhere in an ST program to replace the attached expression. The following ST programming example uses defined words:

```
If OK Then
angle := PI / 2.0;
isdone := YES;
End_if;
```

When the same identifier is defined twice with different ST equivalencies, the last defined expression is used:

| | |
|---|---|
| Define: | OPEN is FALSE |
| | OPEN is TRUE |
| means: | OPEN is TRUE |

Programs can contain up to 255 defined words. Names of defined words can have up to 32 characters and must begin with a letter followed by letters, digits, and single underscore characters. The last character can be either a letter or a digit.

The definition of a defined word cannot contain a defined word. Note the invalid definition (with strikethrough mark) in the following defined word examples:

| | |
|---|---|
| PI is 3.14159 | PI2 is 6.28318 |
| ~~PI2 is PI*2~~ | |

# Data Types

Any literal, expression, or variable used in a POU (written in any language) must be characterized by a data type. Data type coherence must be followed in graphic operations and literal statements. You can program objects using the following elementary IEC 61131-3 types:

- BOOL: logic (true or false) value
- DINT - Integer: integer value-32 bit
- REAL - Real: real (floating) value 32-bit
- TIME: time values less than one day; these value types cannot store dates (32 bit)
- MESSAGE: character string having a defined *size*, representing the maximum number of characters the string can contain. For example, to define `MyString` as a string containing 10 characters, enter `MyString(10)`.

For global and local variables other than MESSAGE type, you can create arrays having one dimension. Upon creation of such arrays, you can choose to retain the values and specify initial values. The following example shows the MyVar variable of type BOOLEAN having a dimension defined as follows:

```
[10]
```

```
FOR i = 0 TO 9 DO
MyVar[i] := FALSE;
END_FOR;
```

## Boolean Data Type

Boolean variables (BOOL) can take one of the Boolean values: **TRUE** or **FALSE**. Boolean variables are typically used in Boolean expressions. Boolean variables can have one of the following attributes:

- Internal: memory variable updated by the program
- Constant: read-only memory variable with an initial value
- Input: variable connected to an input device (refreshed by the system)
- Output: variable connected to an output device

For Boolean literal expressions, **ISaGRAF** targets evaluate all parts of such expressions. Whereas, the IEC 61131-3 standard states that Boolean expressions may be evaluated only to the extent necessary to determine the resultant value. In the following example according to the IEC 61131-3 standard, if B is zero then the first expression `(B <> 0)` is false and the second expression `(A/B > 0)` is not performed.

```
if ((B <> 0) and (A/B > 0)) then
GREATER := true;
else
GREATER := false;
end_if;
```

Boolean literal expressions are the following:

- TRUE is equivalent to the integer value 1
- FALSE is equivalent to the integer value 0

## Double Integer Data Type

Double integer variables are 32-bit signed integer values ranging from -2147483647 to +2147483647. Double Integer variables can have one of the following attributes:

- Internal: memory variable updated by the program
- Constant: read-only memory variable with an initial value
- Input: variable connected to an input device (refreshed by the system)
- Output: variable connected to an output device

**Warning:** A double integer expression cannot contain integer and real variables or literal expressions.

A bit of an integer variable can be accessed using the following syntax:

```
MyVar.i
If MyVar is an Integer.
MyVar.i is a Boolean. "i" must be a literal value from 0 to 31.
```

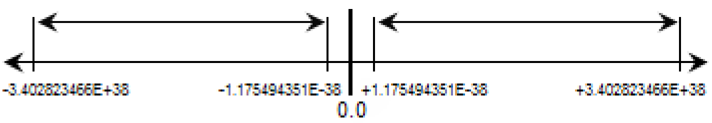
Integer literal values represent signed long integer (32-bit) values ranging from -2147483647 to +2147483647. Integer literals may be expressed with one of the following bases. Integer literals must begin with a prefix identifying the base used:

| **Base** | **Prefix** | **Example** |
|---|---|---|
| DECIMAL | (none) | -908 |
| HEXADECIMAL | "16#" | 16#1A2B3C4D |
| OCTAL | "8#" | 8#1756402 |
| BINARY | "2#" | 2#1101_0001_0101_1101_0001_0010_1011_1001 |

The underscore character ('_') may be used to separate groups of digits. The underscore character has no particular significance other than to improve literal value readability.

## Real Data Type

Real variables are standard IEEE 32-bit floating values (single precision) composed of 1 sign bit + 23 mantissa bits + 8 exponent bits. A real variable has six significant digits. For larger values, the maximum possible value is ±3.402823466E+38 while for smaller values, the minimum possible value is ±1.175494351E-38. Therefore, values greater than ±3.402823466E+38 and greater than 0.0 but less than ±1.175494351E-38 are not supported. The following example shows the value ranges including 0.0 that are supported for real variables:




Real variables can have one of the following attributes:

- Internal: memory variable updated by the program
- Constant: read-only memory variable with an initial value
- Input: variable connected to an input device (refreshed by the system)
- Output: variable connected to an output device

When a real variable is connected to an I/O device, the corresponding I/O driver operates the equivalent integer value.

Real literal values can be written with either decimal or scientific representation. The decimal point ('.') separates the integer and decimal parts. The decimal point differentiates a real literal value from an integer value. The scientific representation uses the 'E' or 'F' letter to separate the mantissa part and the exponent. The exponent part of a real scientific expression must be a signed integer value ranging from -37 to +37.

### Example

| | |
|---|---|
| 3.14159 | -1.0E+12 |
| +1.0 | 1.0E-15 |
| -789.56 | +1.0E-37 |

The expression "123" does not represent a Real literal value. Its correct real representation is "123.0".

## Time Data Type

Time variables are typically used in Time expressions. A Time value represents positive values from 0 to 23h59m59s999ms. Time variables are stored in 32 bit words. The internal representation is a positive number of milliseconds. Time variables can be used with timer function blocks such as TOF and TON. Timer variables can have one of the following attributes:

- Internal: memory variable managed by the program, refreshed by the system
- Constant: read-only memory variable with an initial value

**Warning:** Time variables cannot have the INPUT or OUTPUT attributes.

When a timer is active, its value is automatically increased according to the target system real-time clock. The following ST- language statements can be used to control a timer:

- TSTART, starts automatic refresh of a timer
- TSTOP, stops automatic refresh of a timer

Time literal values represent time values from 0 to 23h59m59s999ms. The lowest allowed unit is a millisecond. Standard time units used in literal values are:

| | |
|---|---|
| Hours | The "h" letter must follow the number of hours |
| Minutes | The "m" letter must follow the number of minutes |
| Seconds | The "s" letter must follow the number of seconds |
| Milliseconds | The "ms" letters must follow the number of milliseconds |

The time literal value must begin with "T#" or "TIME#" prefix. Prefixes and unit letters are not case sensitive. Some units may not appear.

### Example

T#1H450MS 1 hour, 450 milliseconds
time#1H3M 1 hour, 3 minutes

## Message Data Type

Message variables contain character strings. The length of the string can change during process operations. The length of a string variable cannot exceed the capacity (maximum length) specified when the variable is declared. String capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string.

Message variables can contain any character of the standard ASCII table (ASCII code from 0 to 255). The null character (0) can exist in a character string, however, it indicates the end of the string.

Message literal values represent character strings. Characters must be preceded and followed by single quote (') characters. For example:

'THIS IS A MESSAGE'

**Warning:** When placing single quote (‘) characters within a message literal, these characters must be preceded by the dollar ($) character. In the following message literal, note the dollar character preceding the single quote character.

'THIS IS $' A MESSAGE'

A message literal value must be expressed on one line of the program source code. Its length cannot exceed 252 characters, including spaces.

Empty message literal values are represented two single quote (') characters, with no space or tab character between them:

'' (* this is an empty string *)

The dollar ('**$**') special character, followed by other special characters, can be used in a message literal value to represent a non-printable character:

| Sequence | Meaning | ASCII (hex) | Example |
|---|---|---|---|
| $$ | '$' character | 16#24 | 'I paid $$5 for this' |
| $' | apostrophe | 16#27 | 'Enter $'Y$' for YES' |
| $L | line feed | 16#0a | 'next $L line' |

| | | | |
|---|---|---|---|
| $R | carriage return | 16#0d | ' llo $R He' |
| $N | new line | 16#0d0a | 'This is a line$N' |
| $P | new page | 16#0c | 'lastline $P first line' |
| $T | tabulation | 16#09 | 'name$Tsize$Tdate' |
| $hh (*) | any character | 16#hh | 'ABCD = $41$42$43$44' |

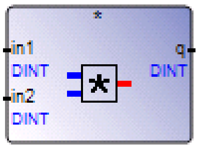
(*) "hh" is the hexadecimal value of the ASCII code for the expressed character.

# Operators

The following are Operators of the IEC 61131-3 languages:

| | | |
|---|---|---|
| **Arithmetic operations** | Addition | Adds two or more variables |
| | Division | Divides two variables |
| | Multiplication | Multiplies two or more variables |
| | Subtraction | Subtracts a variable from another |
| | 1 GAIN | Assigns one variable into another |
| | NEG | Integer negation |
| **Boolean operations** | AND | Boolean AND |
| | OR | Boolean OR |
| | XOR | Boolean exclusive OR |
| **Comparator** | Less Than | Tests if one value is less than another |
| | Less Than or Equal | Tests if one value is less than or equal to another |
| | Greater Than | Tests if one value is greater than another |
| | Greater Than or Equal | Tests if one value is greater than or equal to another |
| | Equal | Tests if one value is equal to another |
| | Not Equal | Tests if one value is not equal to another |
| **Concatenation** | CAT | Concatenates multiple messages into one |
| **Data conversion** | BOO | Converts to Boolean |
| | ANA | Converts to real |
| | REAL | Converts to real |
| | MSG | Converts to message |
| | OPERATE | Varies depending on the implementation of the treated I/O |
| | TMR | Converts to time |
| | SYSTEM | Accesses the system parameters |

# Multiplication




**Note:** For this operator, the number of inputs can be extended to more than two.

Arguments:

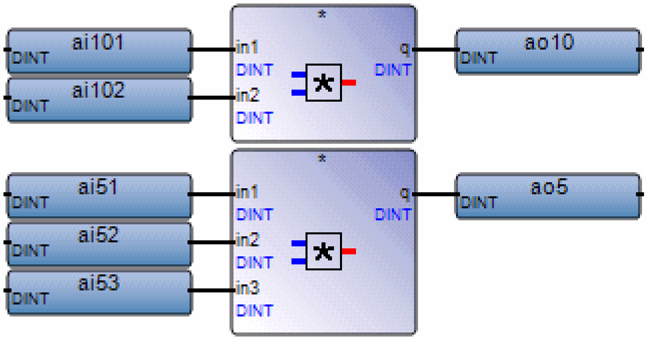
| | | |
|---|---|---|
| (inputs) | DINT - REAL | can be INTEGER or REAL<br>(all inputs must have the same format) |
| output | DINT - REAL | **multiplication** of the input terms |

Description:

Multiplication of two or more integer or real variables.

## Example

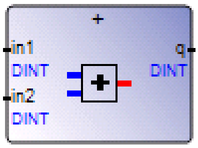
(* FBD example with Multiplication Operators *)




(* ST equivalence *)

```
ao10 := ai101 * ai102;
```

# Addition




**Note:** For this Operator, the number of inputs can be extended to more than two.

Arguments:

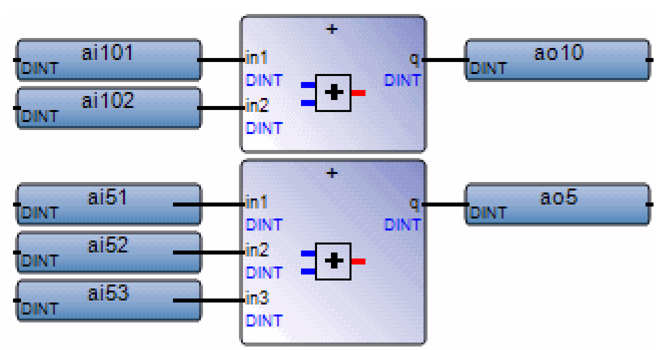
| | | |
|---|---|---|
| (inputs) | DINT - REAL | can be INTEGER or REAL<br>(all inputs must have the same format) |
| output | DINT - REAL | **addition** of the input terms |

Description:

Addition of two or more integer or real variables.

### Example

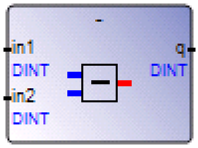
(* FBD example with Addition Operators *)




(* ST equivalence: *)

```
ao10 := ai101 + ai102;
```

# Subtraction




Arguments:

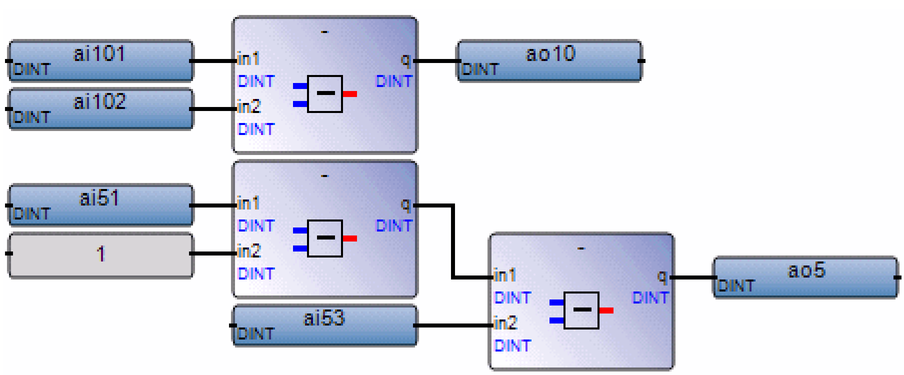
| | | |
|---|---|---|
| IN1 | DINT - REAL | can be integer or real format |
| IN2 | DINT - REAL | (IN1 and IN2 must have the same format) |
| Q | DINT - REAL | subtraction (first - second) |

Description:

Subtraction of two integer or real variables (first - second).

### Example

(* FBD example with Subtraction Operators *)

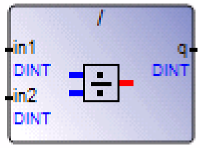



(* ST equivalence: *)

```
ao10 := ai101 - ai102;
```

```
ao5 := (ai51 - 1) - ai53;
```

# Division




Arguments:

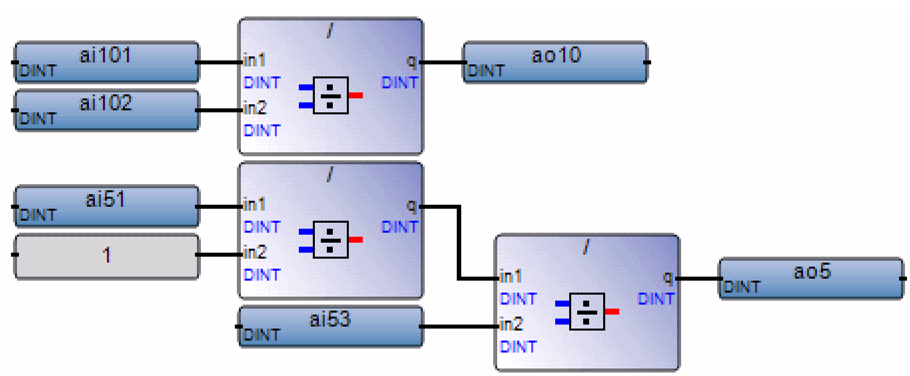
| | | |
|---|---|---|
| IN1 | DINT - REAL | can be integer or real format (operand) |
| IN2 | DINT - REAL | non-zero integer or real value (divisor)<br>(IN1 and IN2 must have the same format) |
| Q | DINT - REAL | integer or real division of IN1 by IN2 |

Description:

Division of two integer or real variables (the first divided by the second).

### Example

(* FBD example with Division Operators *)



(* ST Equivalence: *)

```
ao10 := ai101 / ai102;

ao5 := (ai5 / 2) / ai53;
```

# 1 GAIN


Arguments:

| | | |
|---|---|---|
| IN | DINT - BOOL - MESSAGE - REAL - TIME | |
| Q | DINT - BOOL - MESSAGE - REAL - TIME | IN and Q must have the same format |

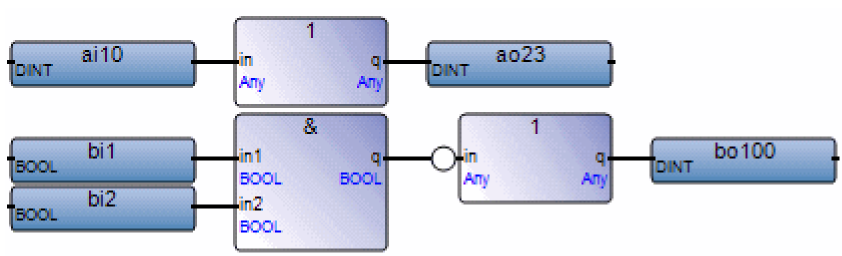
Description:

The assignment of one variable into another

This Block is very useful to directly link a diagram input and a diagram output. It can also be used (with a Boolean negation line) to invert the state of a line connected to a diagram output.

### Example

(* FBD example with assignment Operators *)




(* ST equivalence: *)

```
ao23 := ai10;

bo100 := NOT (bi1 AND bi2);
```

# AND


**Note:** For this Operator, the number of inputs can be extended to more than two.

Arguments:

| | | |
|---|---|---|
| (inputs) | BOOL | |
| output | BOOL | Boolean AND of the input terms |

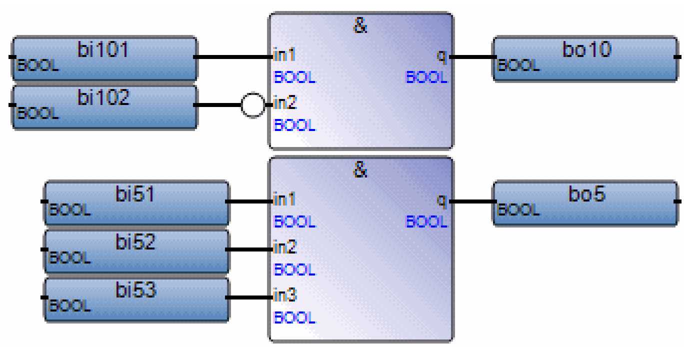
Description:

Boolean AND between two or more terms.

In the text editor, the '&' character can be used as well as typing AND.

### Example

(* FBD example with "AND" Operators *)




(* ST equivalence 1: *)

```
bo10 := bi101 AND NOT (bi102);
```

(* ST equivalence 2: *)

```
bo10 := bi101 & NOT (bi102);
```

# BOO


Arguments:

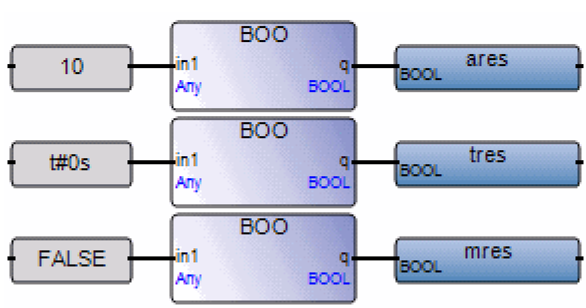
| | | |
|---|---|---|
| IN | DINT- MESSAGE - REAL - TIME | A non-boolean value |
| Q | BOOL | TRUE for non-zero numerical value<br>FALSE for zero numerical value<br>TRUE for 'TRUE' message<br>FALSE for 'FALSE' message |

Description:

Converts a non-boolean variable to a boolean variable.

### Example

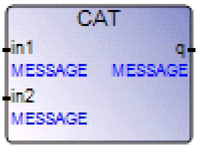
(* FBD example with "BOO" operators *)




(* ST equivalence: *)

```
ares := BOO (10);(* ares is TRUE *)

tres := BOO (t#0s);(* tres is FALSE *)

mres := BOO ('false');(* mres is FALSE *)
```

# CAT




Arguments:

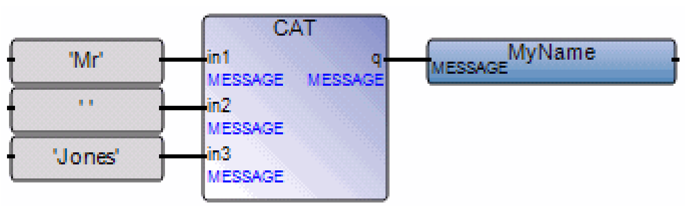
| | | |
|---|---|---|
| (inputs) | MESSAGE | The number of inputs can be extended to more than two. However, the addition of all message lengths must not exceed output message capacity. |
| output | MESSAGE | Concatenation of the input messages |

Description:

Concatenates multiple messages into one message.

## Example

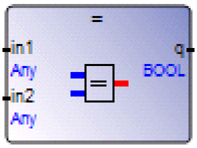
(* FBD example with "CAT" Operator *)




(* ST equivalence: *)

```
myname := ('Mr' + ' ') + 'Jones';
(* means: MyName := 'Mr Jones' *)
```

# Equal




Arguments:

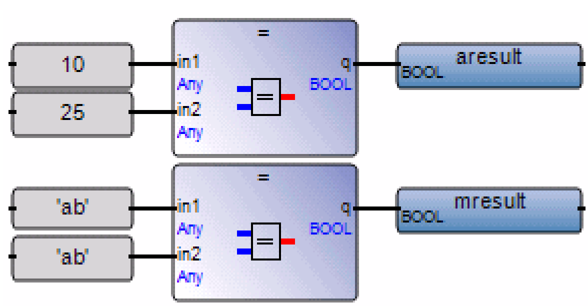
| | | |
|---|---|---|
| IN1 | DINT - BOOL - MESSAGE - REAL | Both inputs must have the same format. |
| IN2 | DINT - BOOL - MESSAGE - REAL | |
| Q | BOOL | TRUE if IN1 = IN2 |

Description

Test if one value is EQUAL TO another one (on integer, real, bool, and message variables)

### Example

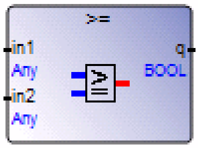
(* FBD example with "Is Equal to" Operators *)




(* ST Equivalence: *)

```
aresult := (10 = 25); (* aresult is FALSE *)
mresult := ('ab' = 'ab'); (* mresult is TRUE *)
```

# Greater Than or Equal




Arguments:

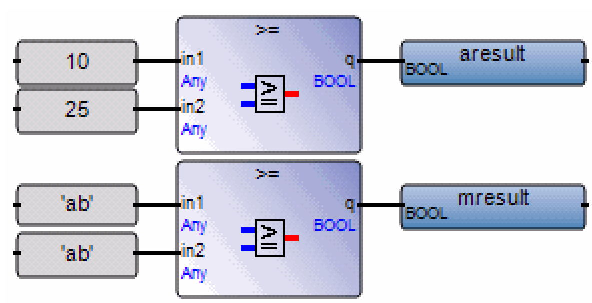
| | | |
|---|---|---|
| IN1 | DINT - BOOL - MESSAGE - REAL | Both inputs must have the same type. |
| IN2 | DINT - BOOL - MESSAGE - REAL | |
| Q | BOOL | TRUE if IN1 >= IN2 |

Description:

Test if one value is GREATER THAN or EQUAL TO another one (on integer, real, bool, and message variables)

### Example

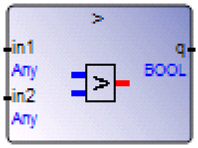
(* FBD example with "Greater or Equal to" Operators *)




(* ST Equivalence: *)

```
aresult := (10 >= 25); (* aresult is FALSE *)
mresult := ('ab' >= 'ab'); (* mresult is TRUE *)
```

# Greater Than




Arguments:

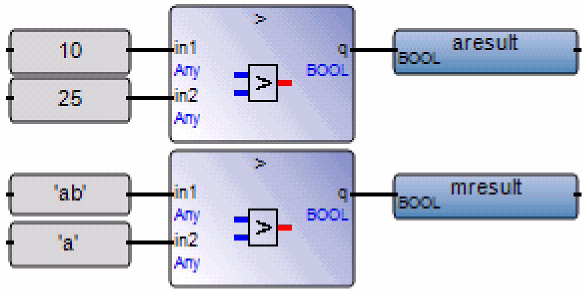
| | | |
|---|---|---|
| IN1 | DINT - BOOL - MESSAGE - REAL - TIME | Both inputs must have the same type |
| IN2 | DINT - BOOL - MESSAGE - REAL - TIME | |
| Q | BOOL | TRUE if IN1 > IN2 |

Description:

Test if one value is GREATER THAN another one (on integer, real, bool, time, and message variables)

## Example

(* FBD example with "Greater than" Operators *)




(* ST Equivalence: *)

```
aresult := (10 > 25); (* aresult is FALSE *)

mresult := ('ab' > 'a'); (* mresult is TRUE *)
```

# ANA


Arguments:

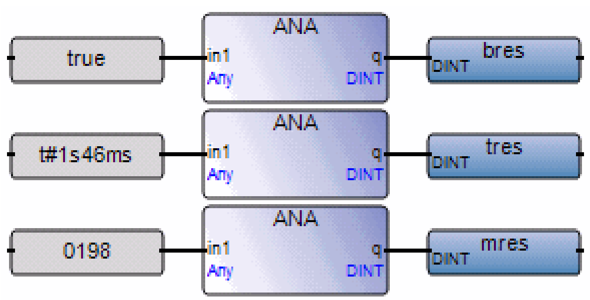
| | | |
|---|---|---|
| IN | BOOL - MESSAGE - REAL - TIME | A non-integer value |
| Q | DINT | 0 if IN is FALSE / 1 if IN is TRUE<br>Number of milliseconds for a timer<br>Integer part for real<br>Decimal number represented by a string |

Description:

Converts a non-integer variable to an integer variable.

### Example

(* FBD example with "ANA" operators *)




(* ST equivalence: *)

```
bres := ANA (true);(* bres is 1 *)

tres := ANA (t#1s46ms);(* tres is 1046 *)

mres := ANA ('0198');(* mres is 198 *)
```

# REAL


Arguments:

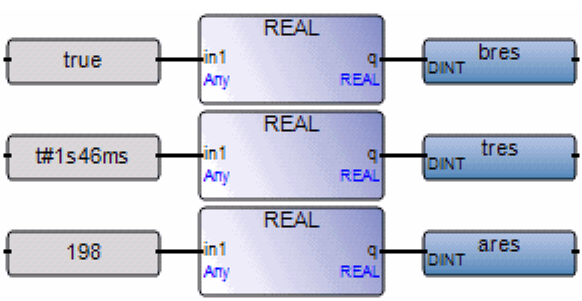
| | | |
|---|---|---|
| IN | DINT - BOOL - MESSAGE - TIME | A non-real value (no message) |
| Q | REAL | 0.0 if IN is FALSE / 1.0 if IN is TRUE<br>Number of milliseconds for a timer<br>Equivalent number for integer |

Description:

Converts a non-real variable to a real variable.

## Example

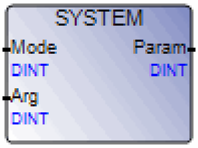
(* FBD example with "REAL" operators *)




(* ST Equivalence: *)

```
bres := REAL (true);                    (* bres is 1.0 *)
tres := REAL (t#1s46ms);                (* tres is 1046.0 *)
ares := REAL (198);                     (* ares is 198.0 *)
```

# SYSTEM




Arguments:

| | | |
|---|---|---|
| MODE | DINT | Identifies the system parameter and the access mode |
| ARG | DINT | New value for a "write" access |
| PARAM | DINT | Value of the accessed parameter |

Description:

Accesses the system parameters to enable performing the following tasks:

- Reading various cycle timing information and changing cycle timing
- Resetting timing counters
- Checking for and reading run-time errors
- Backing up, saving, and restoring variables

The following are the available commands (pre-defined keywords) and expected arguments for the SYSTEM operator:

| **Command** | **Meaning** | **Argument** | **Value** | **Return Value** |
|---|---|---|---|---|
| SYS_TALLOWED | reads allowed cycle timing | 0 | 1 | allowed cycle timing |
| SYS_TCURRENT | reads current cycle timing | 0 | 2 | current cycle timing |
| SYS_TMAXIMUM | reads maximum cycle timing | 0 | 3 | maximum detected timing |
| SYS_TOVERFLOW | reads cycle timing overflows | 0 | 4 | number of timing overflows |

| Command | Meaning | Argument | Value | Return Value |
|---|---|---|---|---|
| SYS_TWRITE | changes cycle timing | new allowed cycle timing | 5 | written time |
| SYS_TRESET | resets timing counters | 0 | 6 | 0 |
| SYS_ERR_TEST | checks for run time errors | 0 | 16 | 0 if no error detected |
| SYS_ERR_READ | reads oldest run time error | 0 | 17 | oldest error code |
| SYS_INITBOO | backs up init Boolean | memory address | 32 | next free address |
| SYS_SAVBOO | saves Booleans | 0 | 33 | zero if OK |
| SYS_RESTBOO | restores Booleans | 0 | 34 | zero if OK |
| SYS_INITANA | backs up init analog | memory address | 36 | next free address |
| SYS_SAVANA | saves analogs | 0 | 37 | zero if OK |
| SYS_RESTANA | restores analogs | 0 | 38 | zero if OK |
| SYS_INITTMR | backs up init timer | memory address | 40 | next free address |
| SYS_SAVTMR | saves timers | 0 | 41 | zero if OK |
| SYS_RESTTMR | restores timers | 0 | 42 | zero if OK |
| SYS_INITALL | backs up init all types | memory address | 44 | next free address |
| SYS_SAVALL | saves all types | 0 | 45 | zero if OK |
| SYS_RESTALL | restores all types | 0 | 46 | zero if OK |

When backing up variables for a specific type or for all types, you need to define the memory backup location using the following syntax:

<*new_address*> := SYSTEM(*SYS_INITxxx*,<*address*>);

where:

<*address*> is the memory backup address location (16# value for Hexadecimal format). The location must be an even address or the operation fails.

*SYS_INITxxx* can be one of the following:
SYS_INITBOO to define memory backup location for all Boolean variables.
SYS_INITANA to define memory backup location for all analog variables.
SYS_INITTMR to define memory backup location for all timer variables.
SYS_INITALL to define memory backup location for all Boolean, analog, and timer variables.

<*new_address*> gets the next free address, i.e., <address> + size of backed up variables (in bytes) according to SYS_INITxxx. This enables verifying the size of the required memory backup. If the operation fails, <*new_address*> gets a zero value.

After having defined the backup memory location, you can perform backups of the variables at any time during the application. The backup is performed once only at the end of the current cycle. If the hardware delivers a Boolean input or a C function to inform of a power failure and allows at least one cycle delay before closing down, the backup may only be performed after detecting the power failure.

<*error*> :=SYSTEM(*SYS_SAVxxx*,0);

where:

*SYS_SAVxxx* can be one of the following:
SYS_SAVBOO to ask for all Boolean variables backup.
SYS_SAVANA to ask for all analog variables backup.
SYS_SAVTMR to ask for all timer variables backup.
SYS_SAVALL to ask for all Boolean, analog and timer variables backup.

<*error*> gets an error status other than zero when the operation fails (SYS_INITxxx is not called).

You can restore variables at any time during the application. The restoration is performed once only at the end of the current cycle. To ensure that the backed up data is valid, an analog variable should be set to a constant value used as a signature.

<*error*> := SYSTEM(*SYS_RESTxxx*,0);

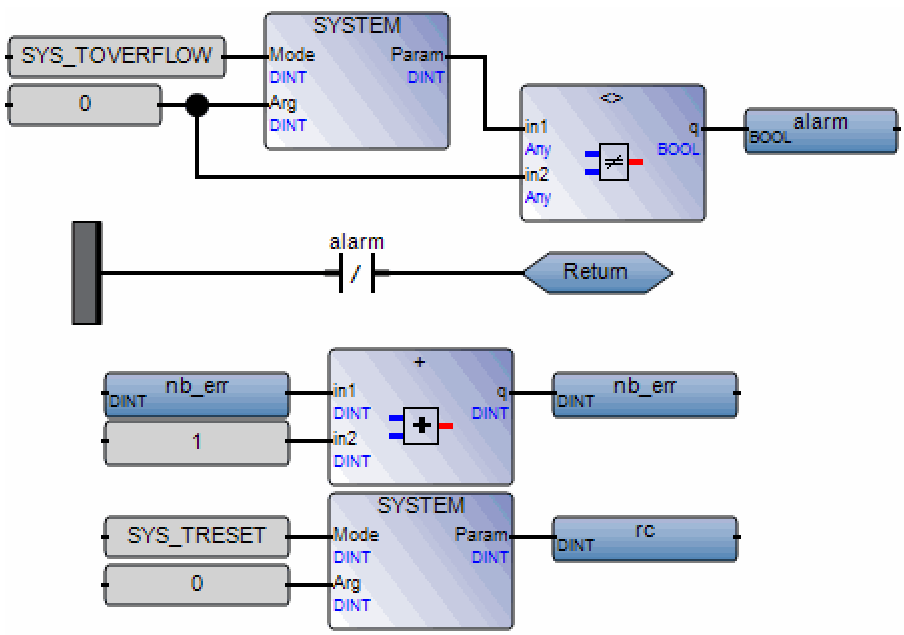
where:

*SYS_RESTxxx* can be one of the following:
SYS_RESTBOO to restore all Boolean variables.
SYS_RESTANA to restore all analog variables.
SYS_RESTTMR to restore all timer variables.
SYS_RESTALL to restore all Boolean, analog and timer variables.

<*error*> gets an error status other than zero when the operation fails (*SYS_INITxxx* is not performed).

### Example

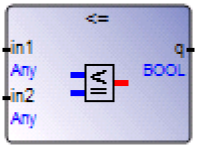
(* FBD example with "SYSTEM" operators *)



(* ST Equivalence: *)

```
alarm := (SYSTEM (SYS_TOVERFLOW, 0) <> 0);
If (alarm) Then
  nb_err := nb_err + 1;
  rc := SYSTEM (SYS_TRESET, 0);
End_If;
```

# Less Than or Equal




Arguments:

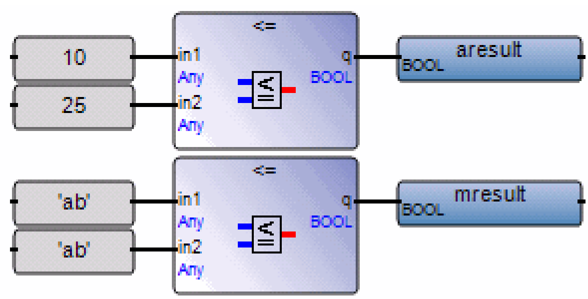
| | | |
|---|---|---|
| IN1 | DINT - BOOL - MESSAGE - REAL | Both inputs must have the same type. |
| IN2 | DINT - BOOL - MESSAGE - REAL | |
| Q | BOOL | TRUE if IN1 <= IN2 |

Description:

Tests if one value is LESS THAN or EQUAL TO another one (on integer, real, bool, and message variables)

## Example

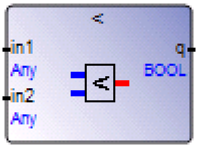
(* FBD example with "Less or equal to" Operators *)




(* ST Equivalence: *)

```
aresult := (10 <= 25); (* aresult is TRUE *)
mresult := ('ab' <= 'ab'); (* mresult is TRUE *)
```

# Less Than




Arguments:

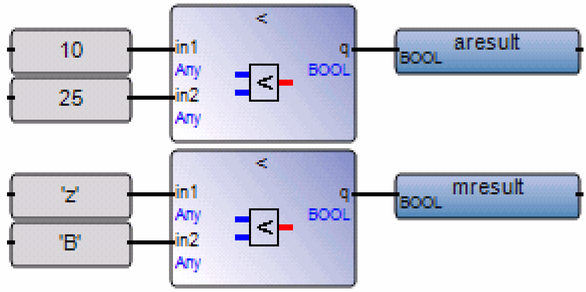
| | | |
|---|---|---|
| IN1 | DINT - BOOL - MESSAGE - REAL - TIME | Both inputs must have the same type |
| IN2 | DINT - BOOL - MESSAGE - REAL - TIME | |
| Q | BOOL | TRUE if IN1 < IN2 |

Description:

Test if one value is LESS THAN another one (on integer, real, bool, time, and message variables)

## Example

(* FBD example with "Less than" Operators *)




(* ST Equivalence: *)

```
aresult := (10 < 25); (* aresult is TRUE *)
mresult := ('z' < 'B'); (* mresult is FALSE *)
```

# MSG


Arguments:

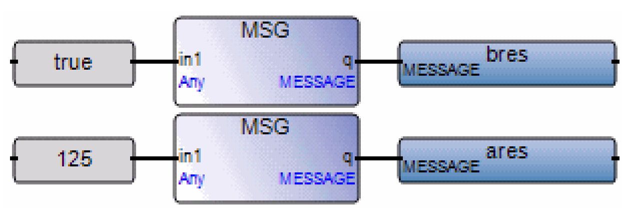
| | | |
|---|---|---|
| IN | DINT - BOOL - REAL - TIME | A non-string value |
| Q | MESSAGE | "false' or 'true' if IN is a boolean value<br>decimal representation if IN is an integer or real |

Description:

Converts an integer, real, boolean, or time variable to a string variable.

## Example

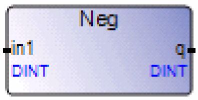
(* FBD example with "Convert to Message" blocks *)




(* ST Equivalence: *)

| | |
|---|---|
| bres := MSG (TRUE); | (* bres is 'TRUE' *) |
| ares := MSG (125); | (* ares is '125' *) |

## NEG




Arguments:

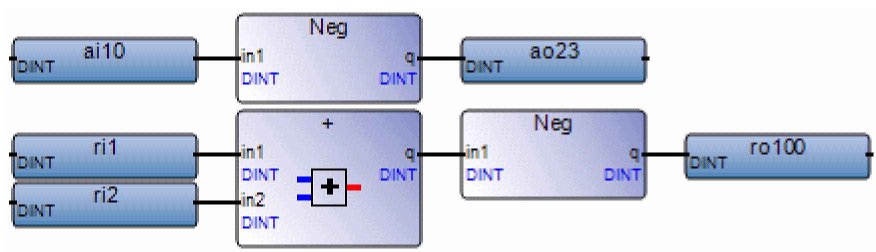
| | | |
|---|---|---|
| IN | DINT - REAL | Input and output must have the same format |
| Q | DINT - REAL | |

Description:

Assignment of the negation of a variable.

### Example

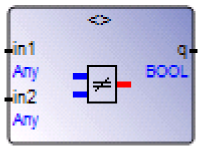
(* FBD example with Negation Operators *)




(* ST equivalence: *)

```
ao23 := - (ai10);

ro100 := - (ri1 + ri2);
```

# Not Equal




Arguments:

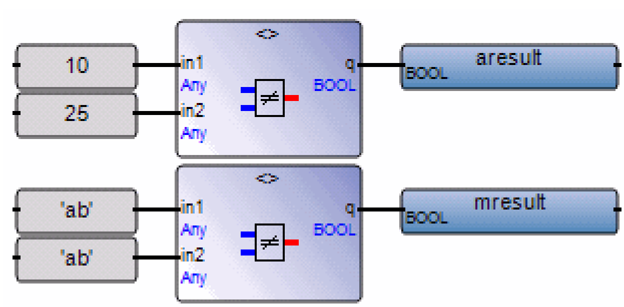
| | | |
|---|---|---|
| IN1 | DINT - BOOL - MESSAGE - REAL | both inputs must have the same type |
| IN2 | DINT - BOOL - MESSAGE - REAL | |
| Q | BOOL | TRUE if first <> second |

Description:

Test if one value is NOT EQUAL TO another one (on integer, real, boolean, and message variables)

### Example

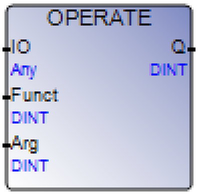
(* FBD example with "Is Not Equal to" Operators *)




(* ST Equivalence: *)

```
aresult := (10 <> 25); (* aresult is TRUE *)
mresult := ('ab' <> 'ab'); (* mresult is FALSE *)
```

# OPERATE




Arguments:

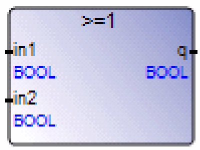
| | | |
|---|---|---|
| IO | DINT - BOOL - MESSAGE - REAL - TIME | Input or output variable |
| FUNCT | DINT | Action to be performed |
| ARG | DINT | Argument for I/O action |
| Q | DINT | Return check |

Description:

Accesses an IO channel

**Note:** The meaning of OPERATE arguments differs from one I/O interface implementation to another.

# OR




**Note:** For this Operator, the number of inputs can be extended to more than two.

Arguments:

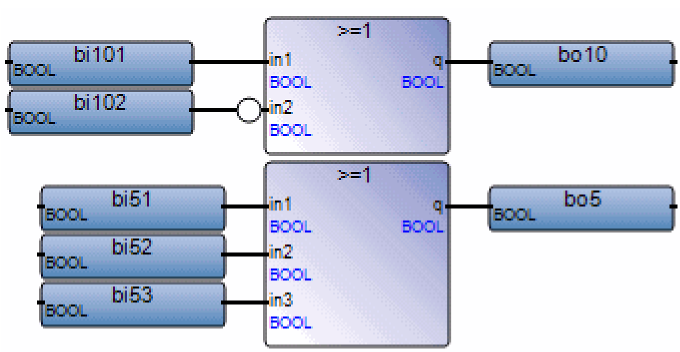
| | | |
|---|---|---|
| (inputs) | BOOL | |
| output | BOOL | Boolean **OR** of the input terms |

Description:

Boolean OR of two or more terms.

### Example

(* FBD example with "OR" Operators *)




(* ST equivalence: *)

```
bo10 := bi101 OR NOT (bi102);

bo5 := (bi51 OR bi52) OR bi53;
```

# TMR


Arguments:

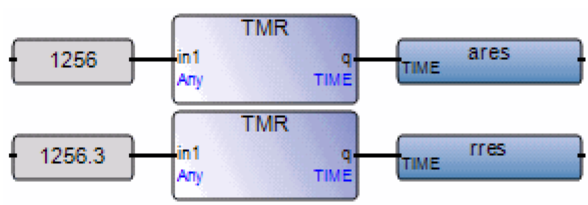
| | | |
|---|---|---|
| IN | DINT - REAL | A non-TIME value<br>IN (or integer part of IN if it is real)<br>is the number of milliseconds |
| Q | TIME | Time value represented by IN |

Description:

Converts an integer or real variable to a time one.

### Example

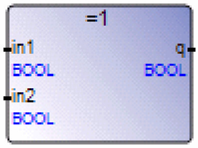
(* FBD example with "Convert to Timer" Operators *)




(* ST Equivalence: *)

```
ares := TMR (1256);                 (* ares := t#1s256ms *)
rres := TMR (1256.3);               (*rres := t#1s256ms *)
```

# XOR




Arguments:

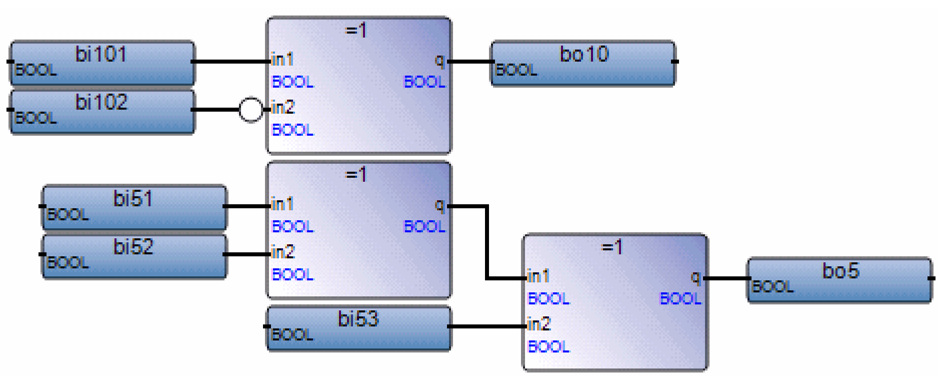
| | | |
|---|---|---|
| IN1 | BOOL | |
| IN2 | BOOL | |
| Q | BOOL | Boolean **exclusive OR** of the two input terms |

Description:

Boolean exclusive OR between two terms.

## Example

(* FBD example with "XOR" operators *)




(* ST equivalence: *)

```
bo10 := bi101 XOR NOT (bi102);
bo5 := (bi51 XOR bi52) XOR bi53;
```

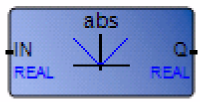
# Functions

The following are the functions supported by the system:

| | | |
|---|---|---|
| **Arithmetic Operations** | ABS | Absolute value of a real value |
| | EXPT, POW | Exponent, power calculation of real values |
| | LOG | Logarithm of a real value |
| | MOD | Modulo |
| | SQRT | Square root of a real value |
| | RAND | Random value |
| | TRUNC | Truncate decimal part of a real value |
| | ACOS, ASIN, ATAN | Arc cosine, Arc sine, Arc tangent of a real value |
| | COS, SIN, TAN | Cosine, Sine, Tangent of a real value |
| **Array manipulation** | ARCREATE | Creates an array of integers |
| | ARREAD | Reads an element in an array of integers |
| | ARWRITE | Stores (writes) a value in an array of integers |
| **Binary operations** | AND_MASK | Integer bit-to-bit AND mask |
| | OR_MASK | Integer bit-to-bit OR mask |
| | XOR_MASK | Integer bit-to-bit Exclusive OR mask |
| | NOT_MASK | Integer bit-to-bit negation |
| | ROL, ROR | Rotate Left, Rotate Right an integer value |
| | SHL, SHR | Shift Left, Shift Right an integer value |
| **Boolean operations** | ODD | Odd parity |

| | | |
|---|---|---|
| **Data manipulation** | MIN, MAX, LIMIT | Minimum, Maximum, Limit |
| | MUX4, MUX8 | Multiplexer (4 or 8 entries) |
| | SEL | Binary selector |
| **File management (for ISaGRAF 3 configurations only)** | F_CLOSE | Closes a binary file |
| | F_EOF | Tests if end of a file has been reached |
| | F_ROPEN | Opens a binary file in read mode |
| | F_WOPEN | Opens a binary file in write mode |
| | FA_READ | Reads integer and real variables from a binary file |
| | FA_WRITE | Writes integer and real variables to a binary file |
| | FM_READ | Reads MESSAGE variables from a binary file |
| | FM_WRITE | Writes MESSAGE variables to a binary file |
| **String manipulation** | ASCII | Character -> ASCII code |
| | CHAR | ASCII code -> Character |
| | MLEN | Get string length |
| | DELETE, INSERT | Delete sub-string, Insert string |
| | FIND, REPLACE | Find sub-string, Replace sub-string |
| | LEFT, MID, RIGHT | Extract left, middle or right of a string |
| **Time operations** | DAY_TIME | Gives date or time of the day |

# ABS




Arguments:

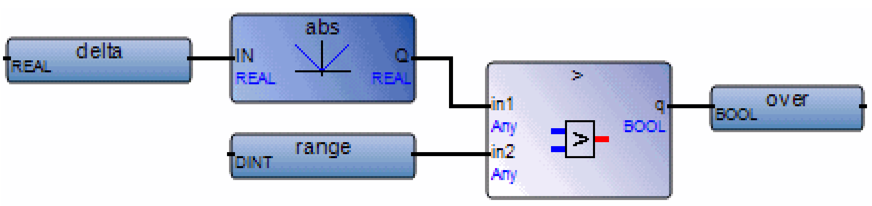
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any signed real value |
| ABS | Q | REAL | Absolute value (always positive) |

Description:

Gives the absolute (positive) value of a real value.

### Example

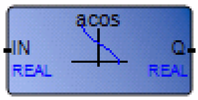
(* FBD Program using "ABS" Function *)




(* ST Equivalence: *)

```
over := (ABS (delta) > range);
```

# ACOS




Arguments:

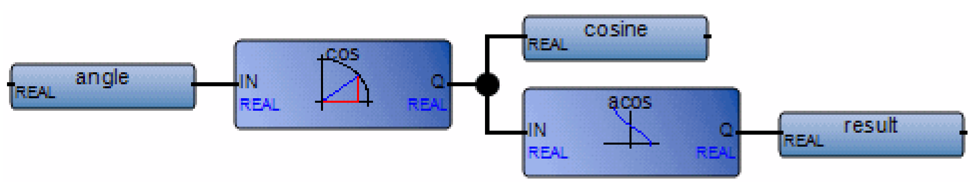
| | | | |
|---|---|---|---|
| IN | IN | REAL | Must be in set [-1.0 .. +1.0] |
| ACOS | Q | REAL | Arc-cosine of the input value (in set [0.0 .. PI])<br>= 0.0 for invalid input |

Description:

Calculates the Arc cosine of a real value.

## Example

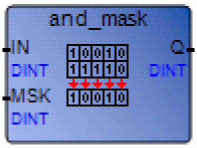
(* FBD Program using "COS" and "ACOS" Functions *)




(* ST Equivalence: *)

```
cosine := COS (angle);

result := ACOS (cosine); (* result is equal to angle *)
```

# AND_MASK




Arguments:

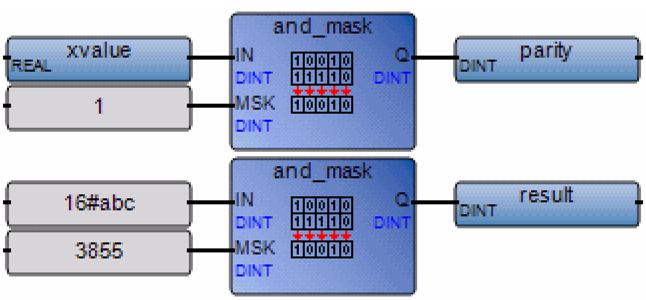
| | | | |
|---|---|---|---|
| IN | IN | DINT | Must have integer format |
| MSK | MSK | DINT | Must have integer format |
| AND_MASK | Q | DINT | Bit-to-bit logical **AND** between IN and MSK |

Description:

Integer AND bit-to-bit mask.

## Example

(* FBD example with AND_MASK Operators *)




(* ST Equivalence: *)

```
parity := AND_MASK (xvalue, 1);  (* 1 if xvalue is odd *)
result := AND_MASK (16#abc, 16#f0f); (* equals 16#a0c *)
```

# ARCREATE


Arguments:

| | | |
|---|---|---|
| ID | DINT | Identifier of the array (must be in set [0..15]) |
| SIZE | DINT | Number of elements in the array |
| OK | DINT | execution status :<br>1 = if array has been successfully created<br>2 = invalid array identifier or array already created<br>3 = invalid size<br>4 = not enough memory |

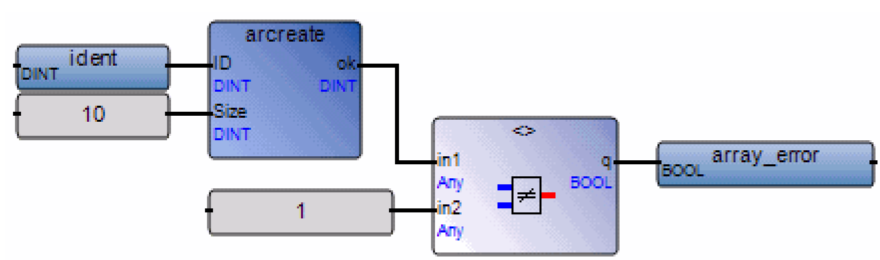
Description:

Creates an array of integers.

**Warning:** There are at most 16 arrays in an application. Arrays contain integer analog values. As dynamic memory allocation is performed, this function may cause a system error if the array size is too close to the size of the available memory.

## Example

(* FBD Program creating an array of integers*)



(* ST Equivalence: *)

```
array_error := (ARCREATE (ident, 16) <> 1));
```

# ARREAD


Arguments:

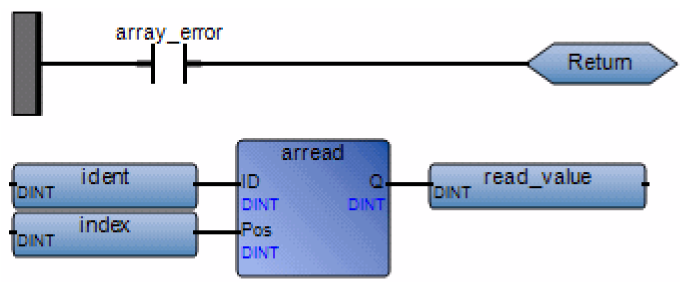
| | | |
|---|---|---|
| ID | DINT | Identifier of the array (must be in set [0..15]) |
| POS | DINT | Position of the element in the array must be in set [0 .. size-1] |
| Q | DINT | value of the element read<br>0 if the arguments are not valid |

Description:

Reads an element in an array of integers.

## Example

(* FBD program using an array management function*)




(* ST Equivalence: *)

```
If (array_error) Then Return; End_if;
read_value := ARREAD (ident, index);
(* array_error comes from the ARCREATE call *)
```

# ARWRITE


Arguments:

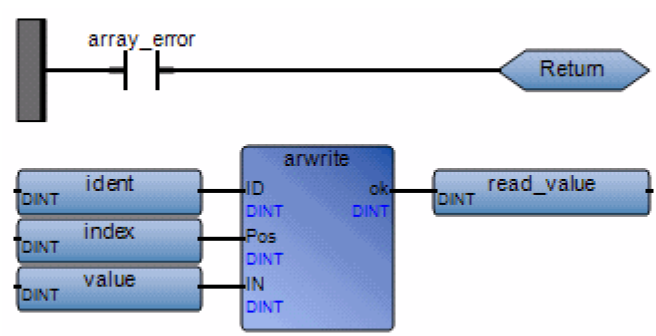
| | | |
|---|---|---|
| ID | DINT | Identifier of the array (must be in set [0..15]) |
| POS | DINT | Position of the element in the array; must be in set [0 .. size-1] |
| IN | DINT | New value for the element |
| OK | DINT | Execution status:<br>1 = writing has succeeded<br>2 = invalid array identifier<br>3 = invalid index |

Description:

Stores (writes) a value in an array of integers.

## Example

(* FBD program using an array management function*)



(* ST Equivalence: *)

```
If (array_error) Then Return; End_if;
write_status := ARWRITE (Ident, Index, value);
(* array_error comes from the ARCREATE call *)
```

# ASCII


Arguments:

| | | | |
|---|---|---|---|
| IN | IN | MESSAGE | Any non-empty string |
| Pos | Pos | DINT | Position of the selected character in set [1.. len] (len is the length of the IN message) |
| ASCII | Code | DINT | Code of the selected character (in set [0 .. 255]) returns 0 is Pos is out of the string |

Description:

Gives the ASCII code of one character in a message string.

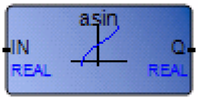
## Example

(* FBD Program using "ASCII" Function *)


(* ST Equivalence: *)

```
FirstChr := ASCII (message_input, 1);
(* FirstChr is the ASCII code of the first character of the string *)
```

# ASIN




Arguments:

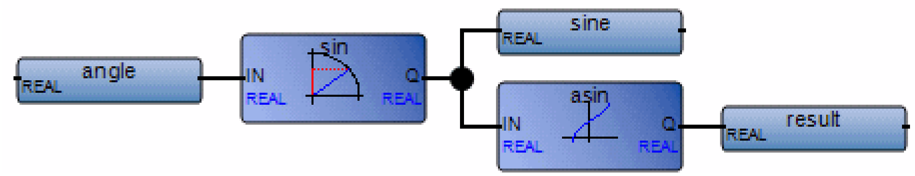
| | | | |
|---|---|---|---|
| IN | IN | REAL | Must be in set [-1.0 .. +1.0] |
| ASIN | Q | REAL | Arc-sine of the input value (in set [-PI/2 .. +PI/2])<br>= 0.0 for invalid input |

Description:

Calculates the Arc sine of a real value.

## Example

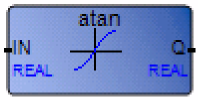
(* FBD Program using "SIN" and "ASIN" Functions *)




(* ST Equivalence: *)

```
sine := SIN (angle);
result := ASIN (sine); (* result is equal to angle *)
```

# ATAN




Arguments:

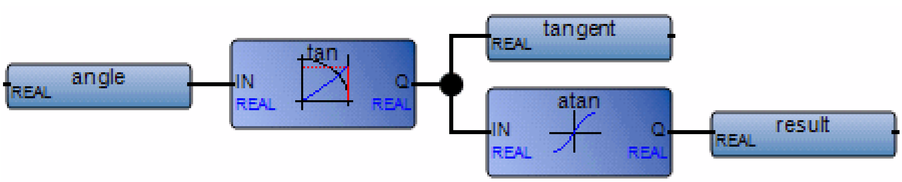
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any real value |
| ATAN | Q | REAL | Arc-tangent of the input value (in set [-PI/2 .. +PI/2])<br>= 0.0 for invalid input |

Description:

Calculates the arc tangent of a real value.

## Example

(* FBD Program using "TAN" and "ATAN" Function *)




(* ST Equivalence: *)

```
tangent := TAN (angle);
result := ATAN (tangent); (* result is equal to angle*)
```

# CHAR


Arguments:

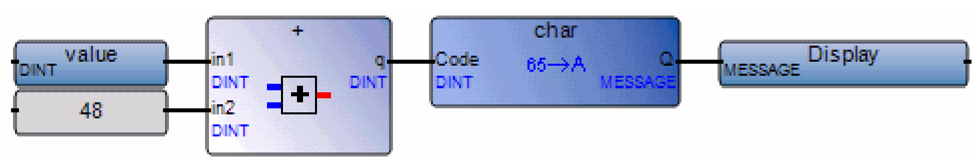
| | | | |
|---|---|---|---|
| Code | Code | DINT | Code in set [0 .. 255] |
| CHAR | Q | MESSAGE | One character string<br>the character has the ASCII code given in input Code<br>(ASCII code is used modulo 256) |

Description:

Gives a one character message string from a given ASCII code.

### Example

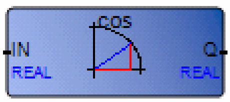
(* FBD Program using "CHAR" Function *)




(* ST Equivalence: *)

```
Display := CHAR ( value + 48 );
(* value is in set [0..9] *)
(* 48 is the ascii code of '0' *)
(* result is one character string from '0' to '9' *)
```

# COS




Arguments:

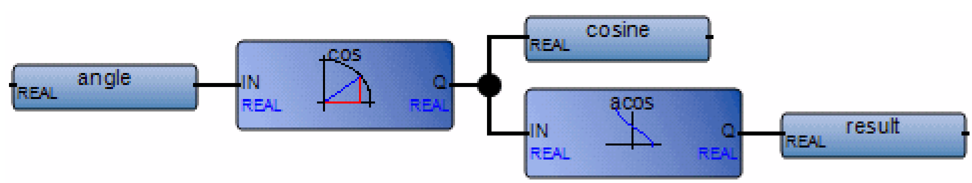
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any REAL value |
| COS | Q | REAL | Cosine of the input value (in set [-1.0 .. +1.0]) |

Description:

Calculates the cosine of a real value.

## Example

(* FBD Program using "COS" and "ACOS" Functions *)




(* ST Equivalence: *)

```
cosine := COS (angle);
result := ACOS (cosine); (* result is equal to angle *)
```

# DAY_TIME


Arguments:

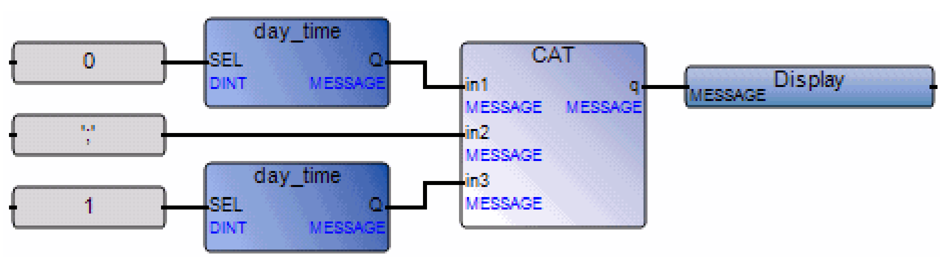
| | | |
|---|---|---|
| SEL | DINT | output selection<br>0= get current date<br>1= get current time<br>2= get day of week |
| Q | MESSAGE | time/date expressed on a character string<br>"YYYY/MM/DD' if SEL = 0<br>"HH:MM:SS' if SEL = 1<br>day name if SEL = 2 (ex: 'Monday') |

Description:

Gives date or time of the day as a message string.

### Example

(* FBD Program using "DAY_TIME" function *)




(* ST Equivalence: *)

```
Display := Day_Time (0) + ' ; ' + Day_Time (1);
(* Display text format is: 'YYYY/MM/DD ; HH:MM:SS' *)
```

# DELETE


Arguments:

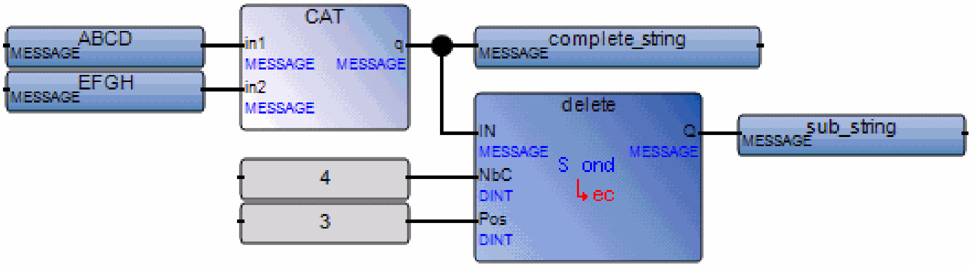
| | | | |
|---|---|---|---|
| IN | IN | MESSAGE | Any non-empty string |
| NbC | NbC | DINT | Number of characters to be deleted |
| Pos | Pos | DINT | Position of the first deleted character<br>(first character of the string has position 1) |
| DELETE | Q | MESSAGE | modified string<br>empty string if Pos < 1<br>initial string if Pos > IN string length<br>initial string if NbC <= 0 |

Description:

Deletes a part of a message string.

### Example

(* FBD Program using "DELETE" Function *)



(* ST Equivalence: *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string is 'ABCDEFGH' *)
sub_string := DELETE (complete_string, 4, 3); (* sub_string is 'ABGH'*)
```

# EXPT


Arguments:

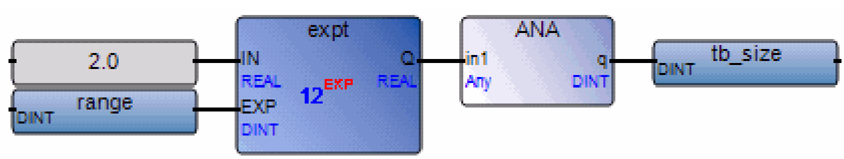
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any signed real value |
| EXP | EXP | DINT | Integer exponent |
| EXPT | Q | REAL | (IN $^{EXP}$) |

Description:

Gives the real result of the operation: (base $^{exponent}$) 'base' being the first argument and 'exponent' the second one.

## Example

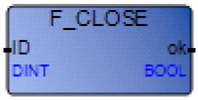
(* FBD Program using "EXPT" Function *)




(* ST Equivalence: *)

```
tb_size := ANY_TO_DINT (EXPT (2.0, range) );
```

# F_CLOSE




Arguments:

| | | |
|---|---|---|
| ID | DINT | File number returned by F_ROPEN or F_WOPEN |
| OK | BOOL | return status<br>TRUE  if file close is OK<br>FALSE if an error occurred |

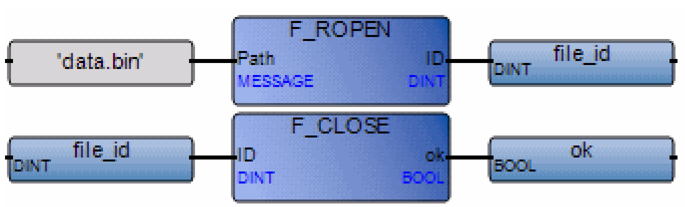
Description:

Closes a binary file open with functions F_ROPEN or F_WOPEN.

This function is not included in the **ISaGRAF** simulator.

## Example

(* FBD program using file management blocks *)




(* ST Equivalence: *)

```
file_id := F_ROPEN('data.bin');

ok := F_CLOSE(file_id);
```

# F_EOF


Arguments:

| | | |
|---|---|---|
| ID | DINT | File number returned by F_ROPEN or F_WOPEN |
| OK | BOOL | End of file indicator.<br>TRUE  if end of file has been reached at the last read or write procedure call.<br>With FM_READ, the last message read from a file may not be correct, if the last character is not a string terminator. |

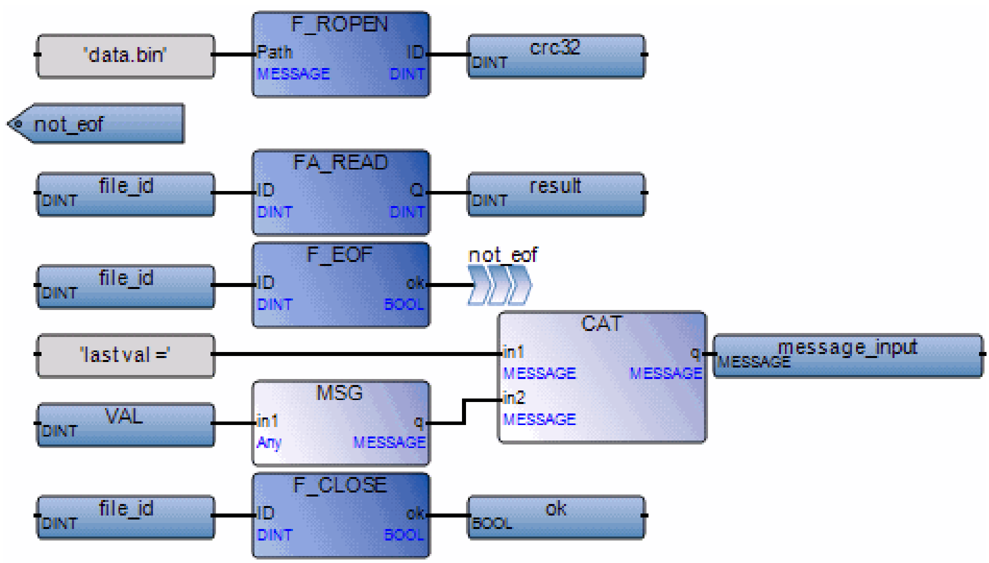
Description:

Tests if end of file has been reached.

This function is not included in the **ISaGRAF** simulator.

## Example

(* FBD program using file management blocks *)




(* ST Equivalence: *)

```
file_id := F_ROPEN('data.bin');
WHILE not(F_EOF(file_id))
VAL := FA_READ(file_id);
END_WHILE;
message_input:= 'last val = ' + msg(VAL);
ok := F_CLOSE(file_id);
```

# F_ROPEN


Arguments:

| | | |
|---|---|---|
| PATH | MESSAGE | May include the access path to the file using the \ or / symbol to specify a directory. To ease application portability, / or \ is equivalent. |
| ID | DINT | File number<br>0 if an error occurs: file does not exist |

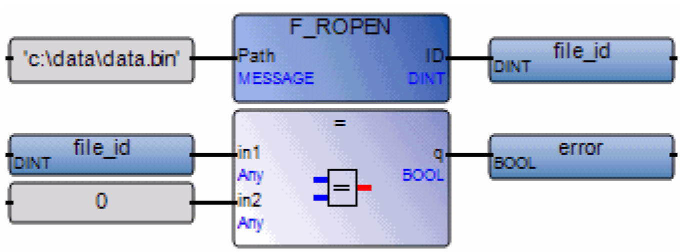
Description:

Opens a binary file in read mode. To be used with FA_READ, FM_READ, and F_CLOSE.

This function is not included in the **ISaGRAF** simulator.

## Example

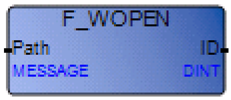
(* FBD program using file management blocks *)




(* ST Equivalence: *)

```
file_id := F_ROPEN('c:\data \data.bin');

error := (file_id=0);
```

# F_WOPEN




Arguments:

| | | |
|---|---|---|
| PATH | MESSAGE | May include the access path to the file using the \ or / symbol to specify a directory. To ease application portability, / or \ is equivalent. |
| ID | DINT | File number<br>0 if an error occurs. If the file already exists, it is overwritten |

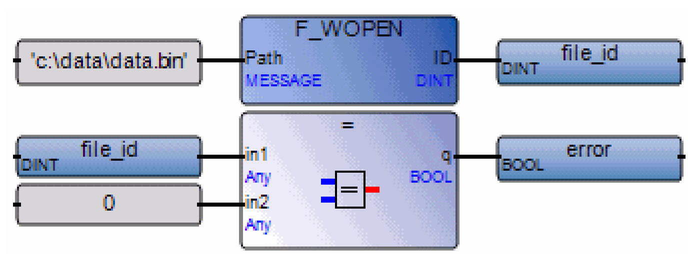
Description:

Opens a binary file in write mode. To be used with FA_WRITE, FM_WRITE, and F_CLOSE.

This function is not included in the **ISaGRAF** simulator.

### Example

(* FBD program using file management blocks *)




(* ST Equivalence: *)

```
file_id := F_WOPEN('c:\data\data.bin');
error := (file_id=0);
```

# FA_READ


Arguments:

| | | |
|---|---|---|
| ID | DINT | File number: returned by F_ROPEN |
| Q | DINT | Integer value read from file |

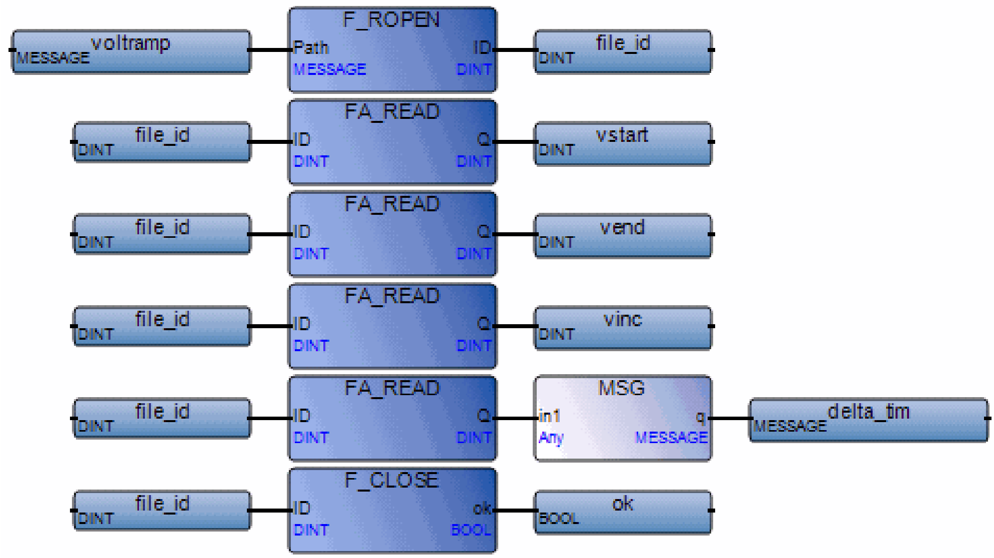
Description:

Reads integer variables from a binary file. To be used with F_ROPEN and F_CLOSE. This procedure makes a sequential access to the file, from the previous position. The first call after F_ROPEN reads the first four bytes of the file, each call pushes the reading pointer. To check if the end of file is reached, use F_EOF.

This function is not included in the **ISaGRAF** simulator.

### Example

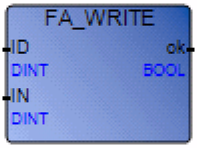
(* FBD program using file management blocks *)




(* ST Equivalence: *)

```
file_id := F_ROPEN('voltramp.bin');
vstart := FA_READ(file_id);
vend := FA_READ(file_id);
vinc := FA_READ(file_id);
delta_tim := tmr(FA_READ(file_id));
ok := F_CLOSE(file_id);
```

# FA_WRITE




Arguments:

| | | |
|---|---|---|
| ID | DINT | File number: returned by F_WOPEN |
| IN | DINT | Integer value to be written in the file |
| OK | BOOL | Execution status: TRUE if ok |

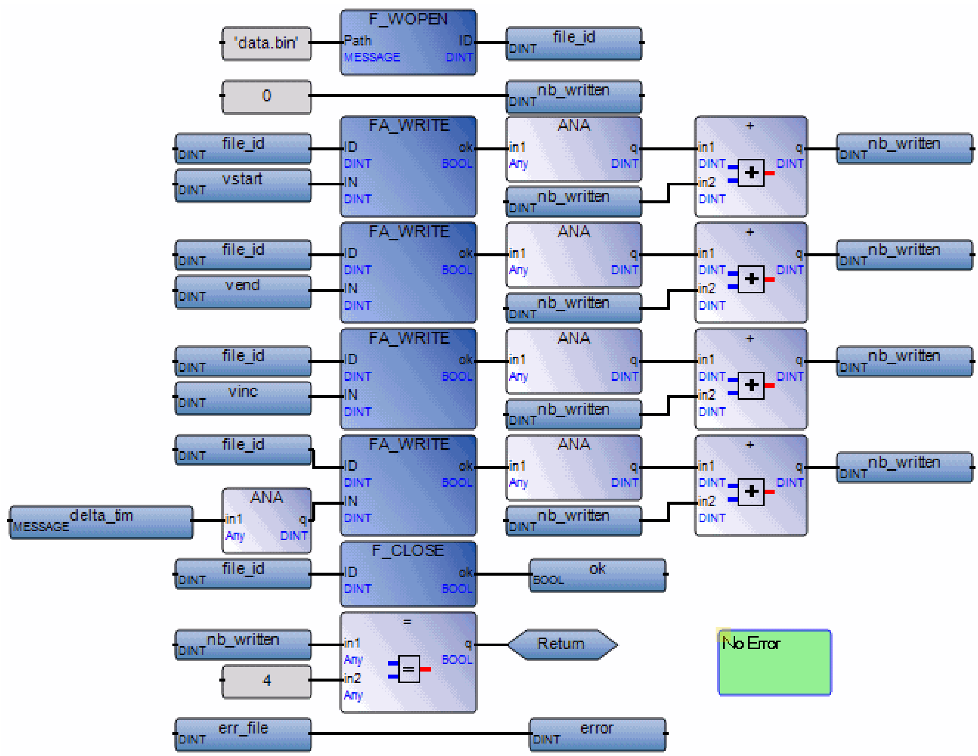
Description:

Writes integer variables to a binary file. This procedure makes a sequential access to the file, from the previous position. The first call after F_WOPEN writes the first four bytes of the file, each call pushes the writing pointer.

This function is not included in the **ISaGRAF** simulator.

### Example

(* FBD program using file management blocks*)




(* ST Equivalence: *)

```
file_id := F_WOPEN('voltramp.bin');
nb_written  := 0;
nb_written := nb_written + dint(FA_WRITE(file_id,vstart));
nb_written := nb_written + dint(FA_WRITE(file_id,vend));
nb_written := nb_written + dint(FA_WRITE(file_id,vinc));
nb_written := nb_written + dint(FA_WRITE(file_id,dint(delta_tim)));
ok := F_CLOSE(file_id);
IF ( nb_written <> 4) THEN
  ERROR := ERR_FILE;
```

```
END_IF;
```

# FM_READ


Arguments:

| | | |
|---|---|---|
| ID | DINT | file number: returned by F_ROPEN |
| Q | MESSAGE | message value read from file |

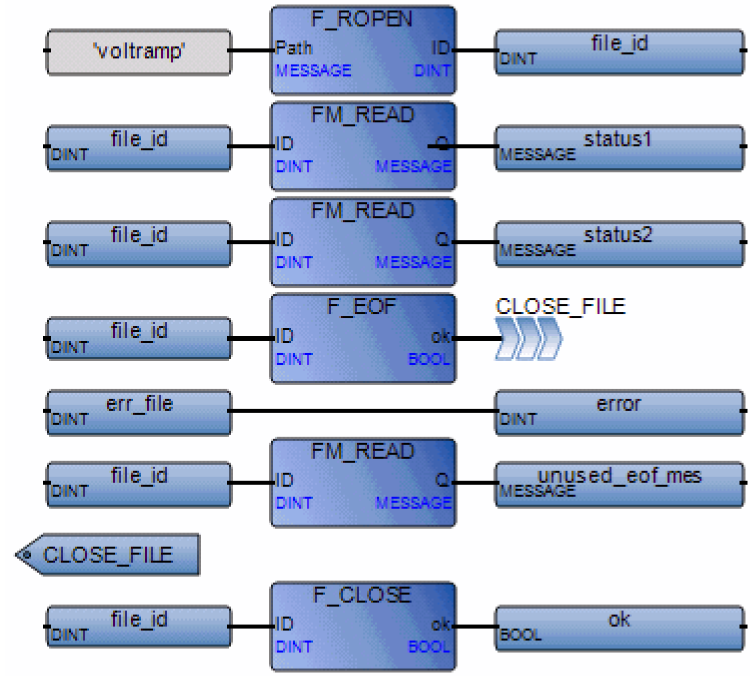
Description:

Reads message variables from a binary file. To be used with F_ROPEN and F_CLOSE. This procedure makes a sequential access to the file, from the previous position. The first call after F_ROPEN reads the first string of the file, each call pushes the reading pointer. A string is a terminated by null (0), end of line ('\n') or return ('\r'); To check if the end of file is reached, use F_EOF.

This function is not included in the **ISaGRAF** simulator.

### Example

(* FBD program using file management blocks *)




(* ST Equivalence: *)

```
file_id := F_ROPEN('voltramp.bin');
status1 := FM_READ(file_id);
status2 := FM_READ(file_id);
IF (F_EOF(file_id)) THEN
  error := err_file;
  unused_eof_mes := FM_READ(file_id);
END_IF;
ok := F_CLOSE(file_id);
```

# FM_WRITE


Arguments:

| | | |
|---|---|---|
| ID | DINT | File number: returned by F_WOPEN |
| IN | MESSAGE | Message value to be written in the file |
| OK | BOOL | Execution status: TRUE if successful |

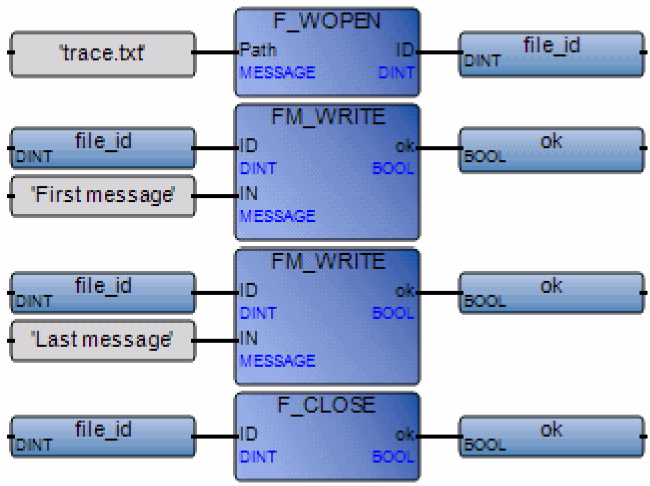
Description:

Writes message variables to a binary file. To be used with F_WOPEN and F_CLOSE. A message is written in the file as a null terminated string. This procedure makes a sequential access to the file, from the previous position. The first call after F_WOPEN writes the first string to the file, each call pushes the writing pointer.

This function is not included in the **ISaGRAF** simulator.

### Example

(* FBD program using file management blocks*)




(* ST Equivalence: *)

```
file_id := F_WOPEN('trace.txt');
ok := FM_WRITE(file_id,'First message');
ok := FM_WRITE(file_id,'Last message');
ok := F_CLOSE(file_id);
```

# FIND


Arguments:

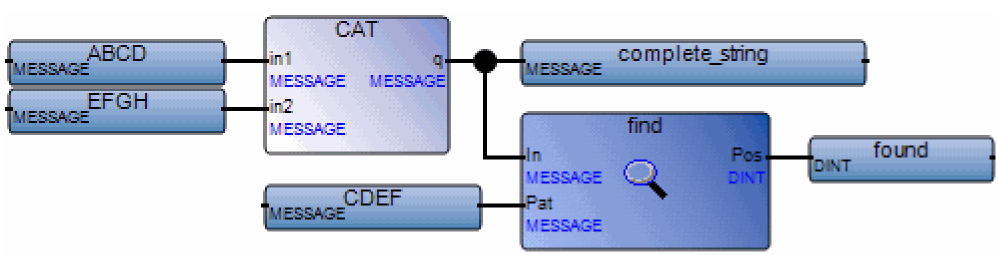
| | | | |
|---|---|---|---|
| In | In | MESSAGE | Any message string |
| Pat | Pat | MESSAGE | Any non-empty string (Pattern) |
| FIND | Pos | DINT | = 0 if sub string Pat not found<br>= position of the first character of the first occurrence of the sub-string Pat<br>(first position is 1)<br>this function is **case sensitive** |

Description:

Finds a sub-string in a message string. Gives the position in the string of the sub-string.

### Example

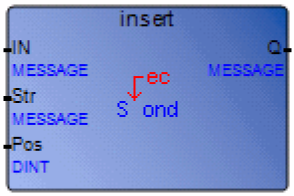
(* FBD Program using "FIND" Function *)




(* ST Equivalence: *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string is 'ABCDEFGH' *)
found := FIND (complete_string, 'CDEF'); (* found is 3 *)
```

# INSERT




Arguments:

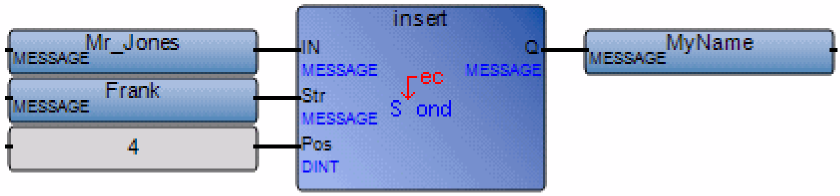
| | | | |
|---|---|---|---|
| IN | IN | MESSAGE | Initial string |
| Str | Str | MESSAGE | String to be inserted |
| Pos | Pos | DINT | Position of the insertion<br>the insertion is done before the position<br>(first valid position is 1) |
| INSERT | Q | MESSAGE | Modified string<br>empty string if Pos <= 0<br>concatenation of both strings if Pos is greater than the length of the IN string |

Description:

Inserts a sub-string in a message string at a given position.

### Example

(* FBD Program using "INSERT" Function *)




(* ST Equivalence: *)

```
MyName := INSERT ('Mr JONES', 'Frank ', 4);
(* MyName is 'Mr Frank JONES' *)
```

# LEFT


Arguments:

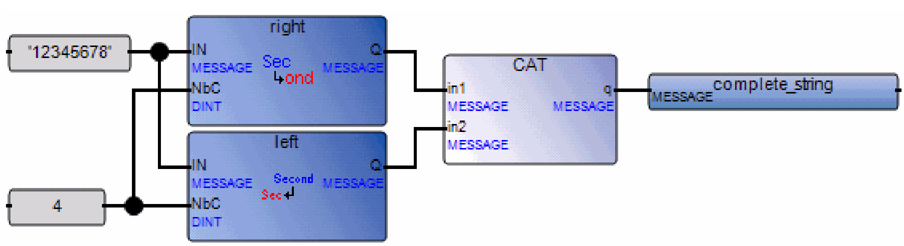
| | | | |
|---|---|---|---|
| IN | IN | MESSAGE | Any non-empty string |
| NbC | NbC | DINT | Number of characters to be extracted. This number cannot be greater than the length of the IN string. |
| LEFT | Q | MESSAGE | Left part of the IN string (its length = NbC)<br>empty string if NbC <= 0<br>complete IN string if NbC >= IN string length |

Description:

Extracts the left part of a message string. The number of characters to be extracted is given.

## Example

(* FBD Program using "LEFT" and "RIGHT" Functions *)




(* ST Equivalence: *)

```
complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);
```

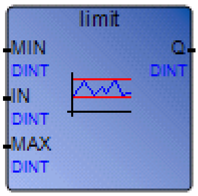
(* complete_string is '56781234'

the value issued from RIGHT call is '5678'

the value issued from LEFT call is '1234'

*)

# LIMIT




Arguments:

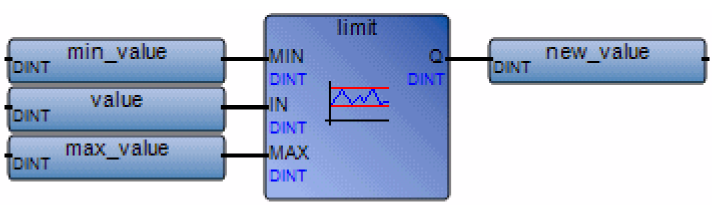
| | | | |
|---|---|---|---|
| MIN | MIN | DINT | Minimum allowed value |
| IN | IN | DINT | Any signed integer value |
| MAX | MAX | DINT | Maximum allowed value |
| LIMIT | Q | DINT | Input value bounded to allowed range |

Description:

Limits an integer value into a given interval. Whether it keeps its value if it is between minimum and maximum, or it is changed to maximum if it is above, or it is changed to minimum if it is below.

## Example

(* FBD Program using "LIMIT" Function *)

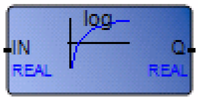



(* ST Equivalence: *)

```
new_value := LIMIT (min_value, value, max_value);
```

(* bounds the value to the [min_value..max_value] set *)

# LOG




Arguments:

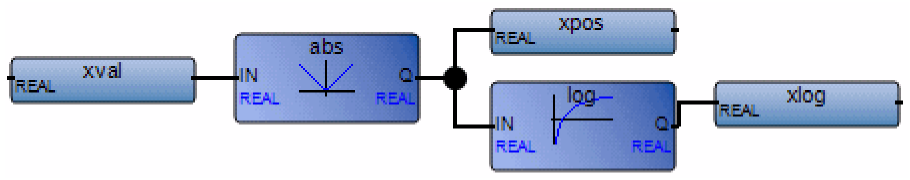
| | | | |
|---|---|---|---|
| IN | IN | REAL | Must be greater than zero |
| LOG | Q | REAL | Logarithm (base 10) of the input value |

Description:

Calculates the logarithm (base 10) of a real value.

### Example

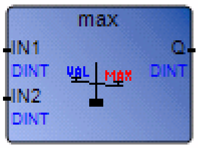
(* FBD Program using "LOG" Function *)




(* ST Equivalence: *)

```
xpos := ABS (xval);
xlog := LOG (xpos);
```

# MAX




Arguments:

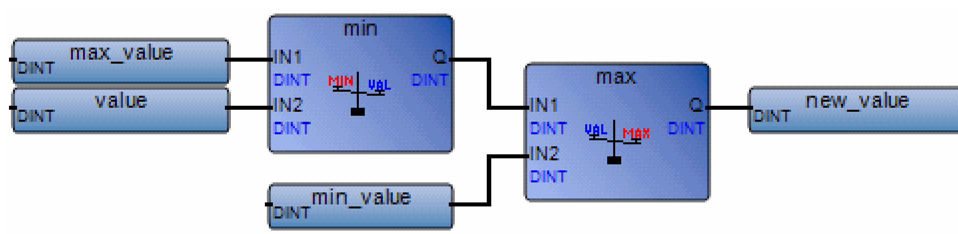
| | | | |
|---|---|---|---|
| IN1 | IN1 | DINT | Any signed integer value |
| IN2 | IN2 | DINT | (cannot be REAL) |
| MAX | Q | DINT | Maximum of both input values |

Description:

Gives the maximum of two integer values.

## Example

(* FBD Program using "MIN" and "MAX" Function *)

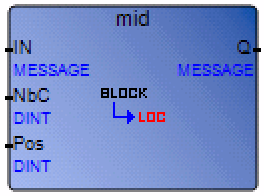



(* ST Equivalence: *)

```
new_value := MAX (MIN (max_value, value), min_value);
```

(* bounds the value to the [min_value..max_value] set *)

# MID




Arguments:

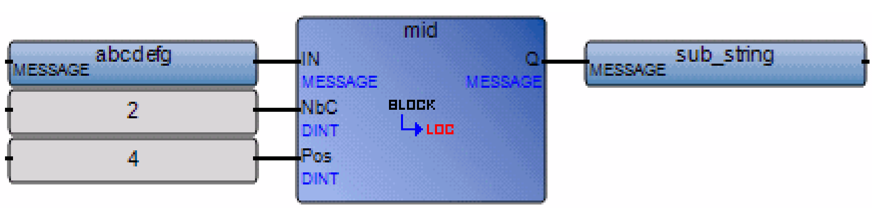
| | | | |
|---|---|---|---|
| IN | IN | MESSAGE | Any non-empty string |
| NbC | NbC | DINT | Number of characters to be extracted cannot be greater than the length of the IN string |
| Pos | Pos | DINT | Position of the sub-string<br>the sub-string first character will be the one pointed to by Pos<br>(first valid position is 1) |
| MID | Q | MESSAGE | Middle part of the string (its length = NbC)<br>empty string if parameters are not valid |

Description:

Extracts a part of a message string. The number of characters to be extracted and the position of the first character are given.

### Example

(* FBD Program using "MID" Function *)

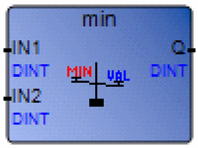



(* ST Equivalence: *)

```
sub_string := MID ('abcdefgh', 2, 4);
```

(* sub_string is 'de' *)

# MIN




Arguments:

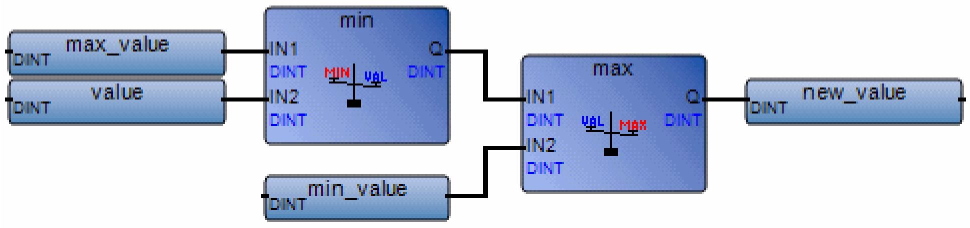
| | | | |
|---|---|---|---|
| IN1 | IN1 | DINT | Any signed integer value |
| IN2 | IN2 | DINT | (cannot be REAL) |
| MIN | Q | DINT | Minimum of both input values |

Description:

Gives the minimum of two integer values.

### Example

(* FBD Program using "MIN" and "MAX" Function *)

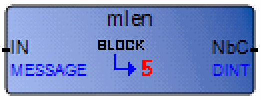



(* ST Equivalence: *)

```
new_value := MAX (MIN (max_value, value), min_value);
```

(* bounds the value to the [min_value..max_value] set *)

## MLEN




Arguments:

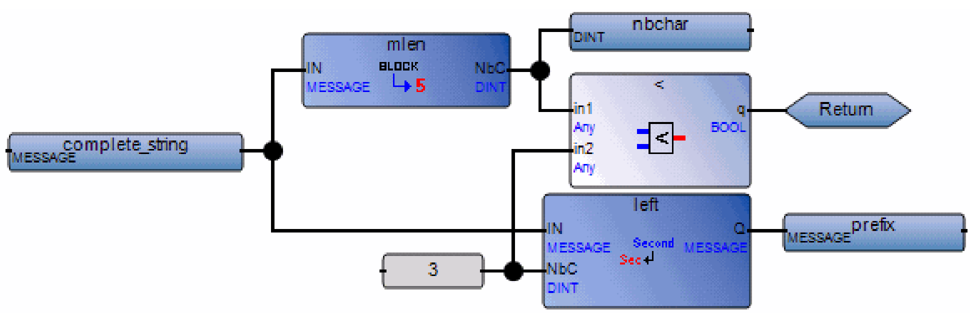
| | | | |
|---|---|---|---|
| IN | IN | MESSAGE | Any message string |
| MLEN | NbC | DINT | Number of characters in the IN string |

Description:

Calculates the length of a message string.

### Example

(* FBD Program using "MLEN" Function *)



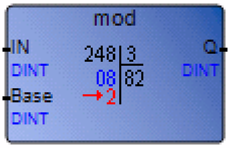

(* ST Equivalence: *)

```
nbchar := MLEN (complete_string);
If (nbchar < 3) Then Return; End_if;
prefix := LEFT (complete_string, 3);
```

(* This program extracts the three characters on the left of the string and places the result in the prefix string variable.

Nothing is done if the string length is less than three characters. *)

# MOD




Arguments:

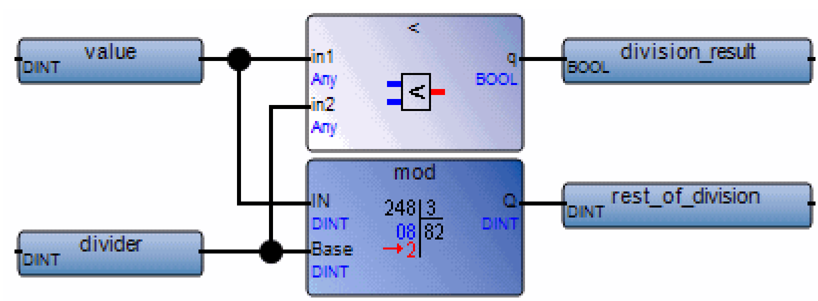
| | | | |
|---|---|---|---|
| IN | IN | DINT | Any signed integer value |
| Base | Base | DINT | Must be greater than zero |
| | Q | DINT | Modulo calculation (input MOD base) returns -1 if Base <= 0 |

Description:

Calculates the modulo of an integer value.

### Example

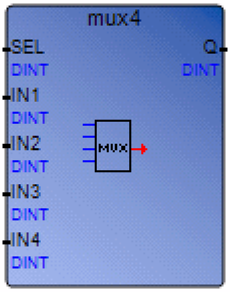
(* FBD Program using "MOD" Function *)




(* ST Equivalence: *)

```
division_result := (value / divider); (* integer division *)
rest_of_division := MOD (value, divider); (* rest of the division *)
```

## MUX4




Arguments:

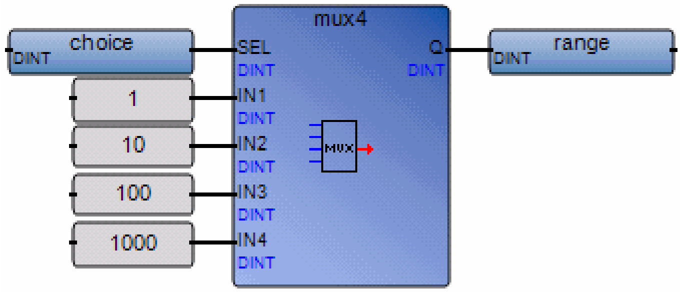
| | | | |
|---|---|---|---|
| SEL | SEL | DINT | Selector integer value (must be in set [0..3]) |
| IN1...IN4 | IN1..IN4 | DINT | Any integer values |
| MUX4 | Q | DINT | = value1 if SEL = 0<br>= value2 if SEL = 1<br>= value3 if SEL = 2<br>= value4 if SEL = 3<br>= 0 for all other values of the selector |

Description:

Multiplexer with four entries: selects a value between four integer values.

### Example

(* FBD Program using "MUX4" Function *)

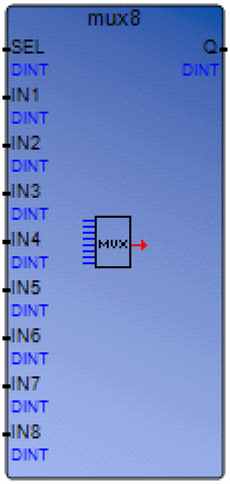



(* ST Equivalence: *)

```
range := MUX4 (choice, 1, 10, 100, 1000);
```

(* select from 4 predefined ranges, for example, if choice is 1, range will be 10 *)

# MUX8




Arguments:

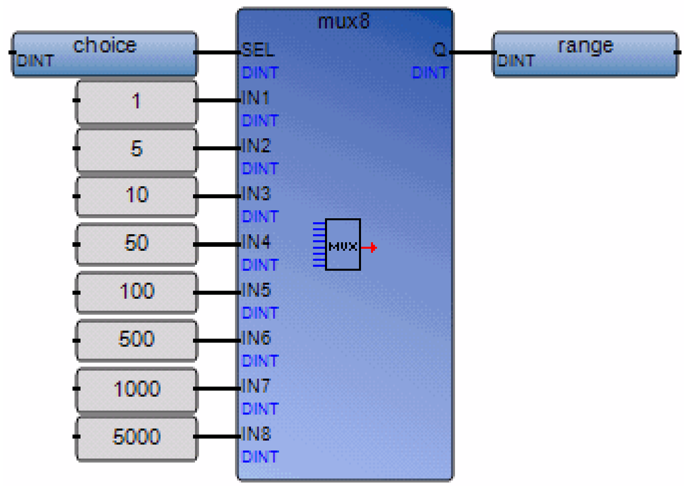
| | | | |
|---|---|---|---|
| SEL | SEL | DINT | Selector integer value (must be in set [0..7]) |
| IN1...IN8 | IN1..IN8 | DINT | Any integer values |
| MUX8 | Q | DINT | = value1 if selector = 0<br>= value2 if selector = 1<br>...<br>= value8 if selector = 7<br>= 0 for all other values of the selector |

Description:

Multiplexer with eight entries: selects a value between eight integer values.

### Example

(* FBD Program using "MUX8" Function *)




(* ST Equivalence: *)

```
range := MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);
```

(* select from 8 predefined ranges, for example, if choice is 3, range will be 50 *)

# NOT_MASK


Arguments:

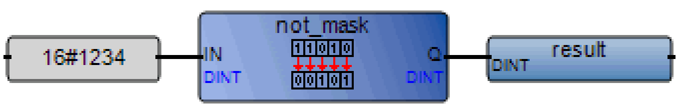
| | | | |
|---|---|---|---|
| IN | IN | DINT | Must have integer format |
| NOT_MASK | Q | DINT | Bit-to-bit negation on 32 bits of IN |

Description:

Integer bit-to-bit negation mask.

## Example

(* FBD example with NOT_MASK Operators *)

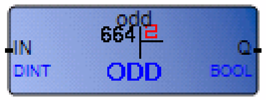



(*ST equivalence: *)

```
result := NOT_MASK (16#1234);
```

(* result is 16#FFFF_EDCB *)

# ODD




Arguments:

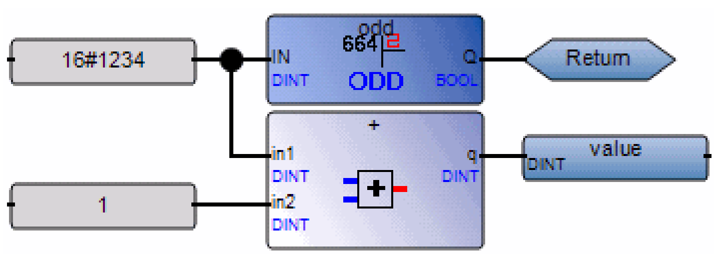
| | | | |
|---|---|---|---|
| IN | IN | DINT | Any signed integer value |
| Odd | Q | BOOL | TRUE if input value is odd<br>FALSE if input value is even |

Description:

Tests the parity of an integer: result is odd or even.

## Example

(* FBD Program using "ODD" Function *)

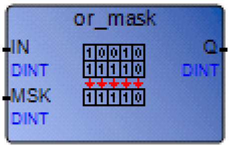



(* ST Equivalence: *)

```
If Not (ODD (value)) Then Return; End_if;

value := value + 1;
```

(* makes value always even *)

# OR_MASK




Arguments:

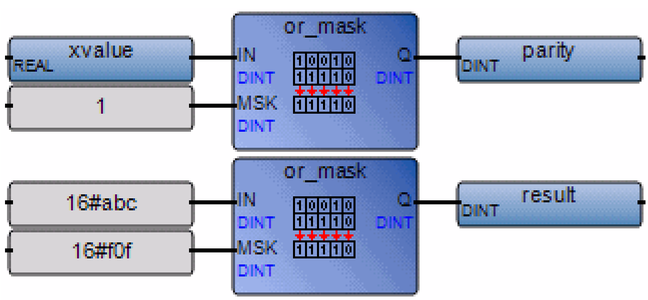
| | | | |
|---|---|---|---|
| IN | IN | DINT | Must have integer format |
| MSK | MSK | DINT | Must have integer format |
| OR_MASK | Q | DINT | Bit-to-bit logical **OR** between IN and MSK |

Description:

Integer OR bit-to-bit mask.

### Example

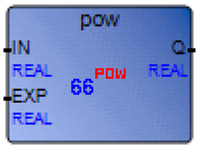
(* FBD example with OR_MASK Operators *)




(* ST Equivalence: *)

```
parity := OR_MASK (xvalue, 1); (* makes value always odd *)
result := OR_MASK (16#abc, 16#f0f); (* equals 16#fbf *)
```

# POW




Arguments:

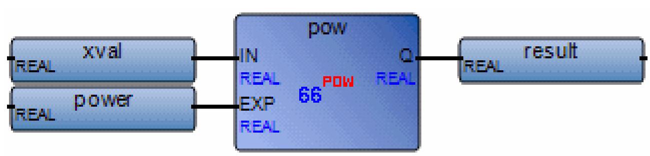
| | | | |
|---|---|---|---|
| IN | IN | REAL | Real number to be raised |
| EXP | EXP | REAL | Power (exponent) |
| POW | Q | REAL | ($IN^{EXP}$)<br>1.0 if IN is not 0.0 and EXP is 0.0<br>0.0 if IN is 0.0 and EXP is negative<br>0.0 if both IN and EXP are 0.0<br>0.0 if IN is negative and EXP does not correspond to an integer |

Description:

Gives the real result of the operation: ($base^{exponent}$) 'base' being the first argument and 'exponent' the second one. The exponent is a real value.

### Example

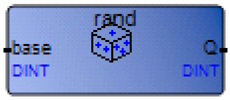
(* FBD Program using "POW" Function *)




(* ST Equivalence: *)

```
result := POW (xval, power);
```

# RAND




Arguments:

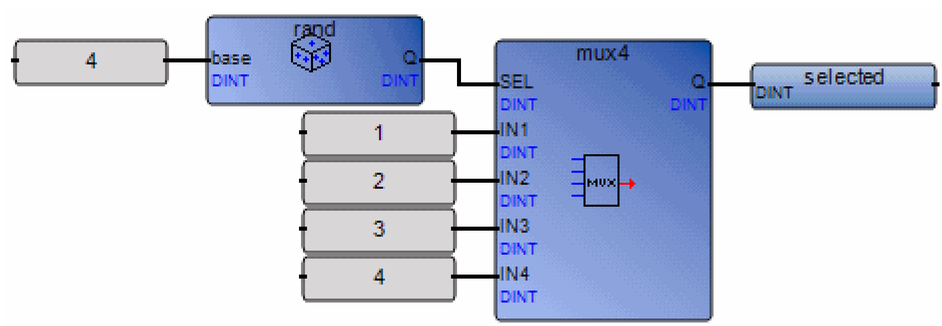
| | | | |
|---|---|---|---|
| base | base | DINT | Defines the allowed set of number |
| RAND | Q | DINT | Random value in set [0..base-1] |

Description:

Gives a random integer value in a given range.

### Example

(* FBD Program using "RAND" function *)

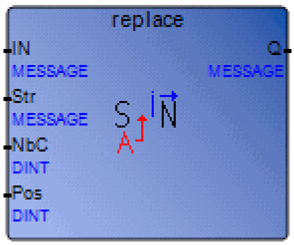



(* ST Equivalence: *)

```
selected := MUX4 ( RAND (4), 1, 4, 8, 16 );
(*
random selection of 1 of 4 pre-defined values
the value issued of RAND call is in set [0..3],
so 'selected' issued from MUX4, will get 'randomly' the value
```

```
1 if 0 is issued from RAND,
or 4 if 1 is issued from RAND,
or 8 if 2 is issued from RAND,
or 16 if 3 is issued from RAND,
*)
```

# REPLACE




Arguments:

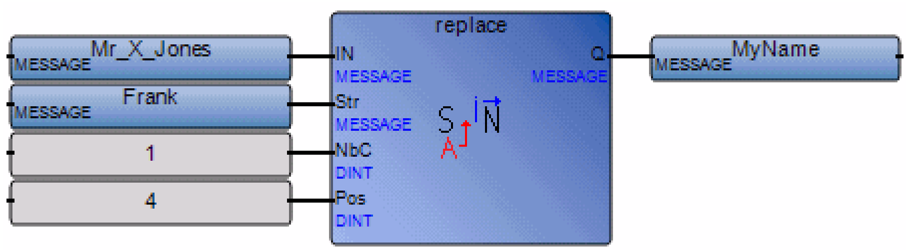
| | | | |
|---|---|---|---|
| IN | IN | MESSAGE | Any string |
| Str | Str | MESSAGE | String to be inserted (to replace NbC chars) |
| NbC | NbC | DINT | Number of characters to be deleted |
| Pos | Pos | DINT | Position of the first modified character<br>(first valid position is 1) |
| REPLACE | Q | MESSAGE | Modified string:<br>- NbC characters are deleted at position Pos<br>- then substring Str is inserted at this position<br>returns empty string if Pos <= 0<br>returns strings concatenation (IN+Str) if Pos is greater than the length of the IN string<br>returns initial string IN if NbC <= 0 |

Description:

Replaces a part of a message string by a new set of characters.

### Example

(* FBD program using "REPLACE" function *)




(* ST Equivalence: *)

```
MyName := REPLACE ('Mr X JONES, 'Frank', 1, 4);
```

(* MyName is 'Mr Frank JONES' *)

# RIGHT


Arguments:

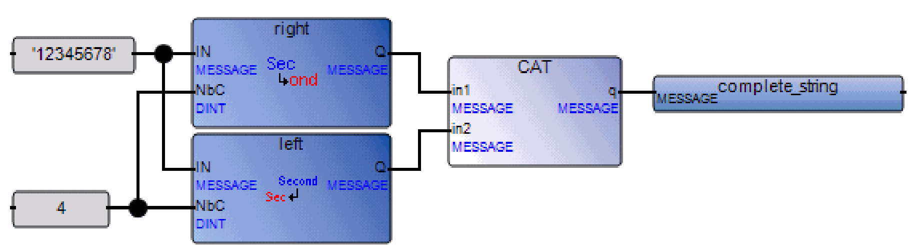
| | | | |
|---|---|---|---|
| IN | IN | MESSAGE | Any non-empty string |
| NbC | NbC | DINT | Number of characters to be extracted. This number cannot be greater than the length of the IN string. |
| RIGHT | Q | MESSAGE | Right part of the string (length = NbC)<br>empty string if NbC <= 0<br>complete string if NbC >= string length |

Description:

Extracts the right part of a message string. The number of characters to be extracted is given.

## Example

(* FBD Program using "LEFT" and "RIGHT" Functions *)(* ST Equivalence: *)




```
complete_string := RIGHT ('12345678', 4) + LEFT ('12345678', 4);
```

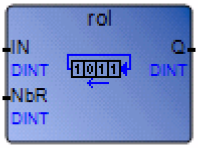
(* complete_string is '56781234'

the value issued from RIGHT call is '5678'

the value issued from LEFT call is '1234'

*)

# ROL

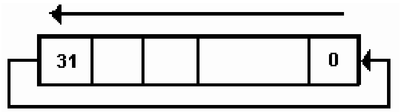



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DINT | Any integer value |
| NbR | NbR | DINT | Number of 1 bit rotations (in set [1..31]) |
| ROL | Q | DINT | Left rotated value<br>no effect if NbR <= 0 |

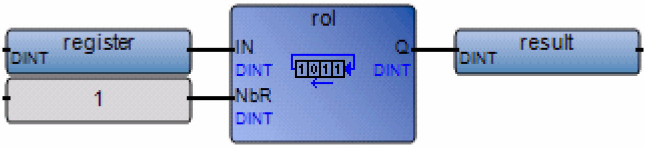
Description:

Make the bits of an integer rotate to the left. Rotation is made on 32 bits:




## Example

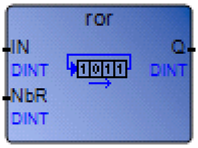
(* FBD Program using "ROL" Function *)




(* ST Equivalence: *)

```
result := ROL (register, 1);
(* register = 2#0100_1101_0011_0101*)
(* result = 2#1001_1010_0110_1010*)
```

# ROR

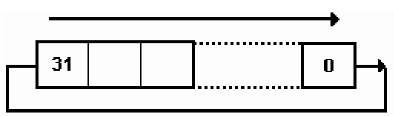



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DINT | Any integer value |
| NbR | NbR | DINT | Number of 1 bit rotations (in set [1..31]) |
| ROR | Q | DINT | Right rotated value<br>no effect if NbR <= 0 |

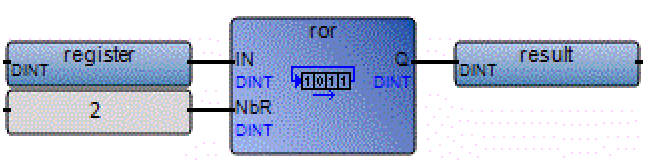
Description:

Make the bits of an integer rotate to the right. Rotation is made on 32 bits:




## Example

(* FBD Program using "ROR" Function *)




```
(* ST Equivalence: *)

result := ROR (register, 2);

(* register = 2#0011_0011_0010_1011_0011_0010_1001_1001 *)

(* result = 2#0100_1100_1100_1010_1100_1100_1010_0110 *)
```

# SEL


Arguments:

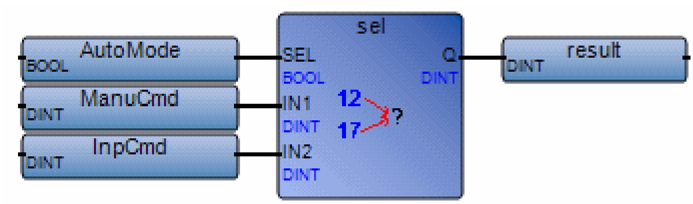
| | | | |
|---|---|---|---|
| SEL | SEL | BOOL | Indicates the chosen value |
| IN1,IN2 | IN1, IN2 | DINT | Any integer values |
| SEL | Q | DINT | = IN1 if SEL is FALSE<br>= IN2 if SEL is TRUE |

Description:

Binary selector: selects a value between two integer values.

### Example

(* FBD Program using "SEL" Function *)

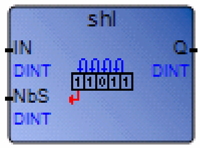



(* ST Equivalence: *)

```
ProCmd := SEL (AutoMode, ManuCmd, InpCmd);
```

(* process command selection *)

# SHL

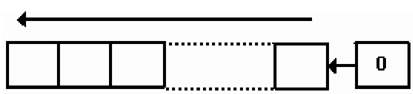



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DINT | Any integer value |
| NbS | NbS | DINT | Number of 1 bit shifts (in set [1..31]) |
| SHL | Q | DINT | Left shifted value<br>no effect if NbS <= 0<br>0 replaces the least significant bit |

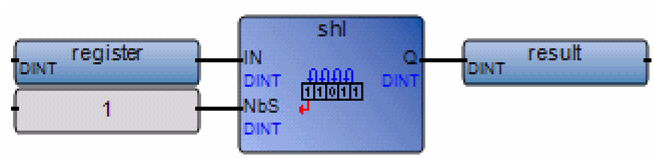
Description:

Shifts the 32 bits of an integer to the left and places a 0 in the least significant bit.




## Example

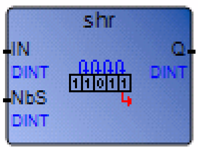
(* FBD Program using "SHL" Function *)




(* ST Equivalence: *)

```
result := SHL (register,1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1001_1010_0110_1010 *)
```

# SHR

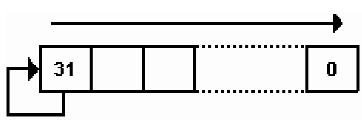



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DINT | Any integer value |
| NbS | NbS | DINT | Number of 1 bit shifts (in set [1..31]) |
| SHR | Q | DINT | Right shifted value<br>no effect if NbS <= 0<br>the leftmost bit is replicated if NbS >=1 |

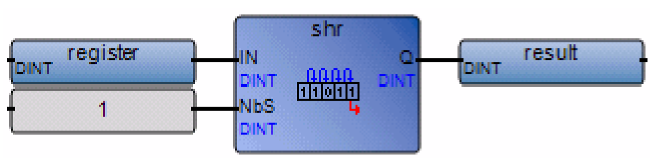
Description:

Shifts the 32 bits of an integer to the right and replicates the leftmost bit (significant bit) to fill the vacant bits.




## Example

(* FBD Program using "SHR" Function *)

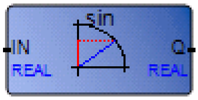



(* ST Equivalence: *)

```
result := SHR (register,1);
(* register = 2#1100_1101_0011_0101 *)
```

```
(* result = 2#1110_0110_1001_1010 *)
```

# SIN




Arguments:

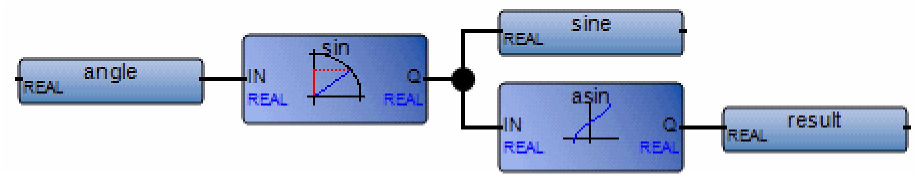
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any REAL value |
| SIN | Q | REAL | Sine of the input value (in set [-1.0 .. +1.0]) |

Description:

Calculates the Sine of a real value.

### Example

(* FBD Program using "SIN" and "ASIN" Functions *)




(* ST Equivalence: *)

```
sine := SIN (angle);
result := ASIN (sine); (* result is equal to angle *)
```

# SQRT


Arguments:

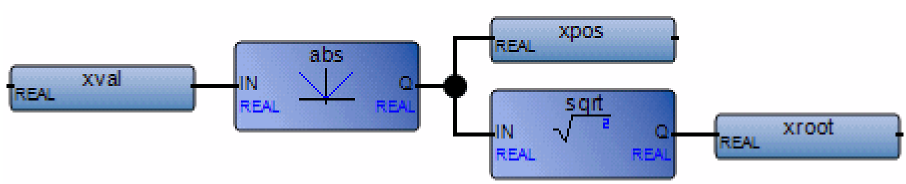
| | | | |
|---|---|---|---|
| IN | IN | REAL | Must be greater than or equal to zero |
| SQRT | Q | REAL | Square root of the input value |

Description:

Calculates the square root of a real value.

### Example

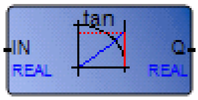
(* FBD Program using "SQRT" Function *)




(* ST Equivalence: *)

```
xpos  := ABS (xval);
xroot  := SQRT (xpos);
```

# TAN




Arguments:

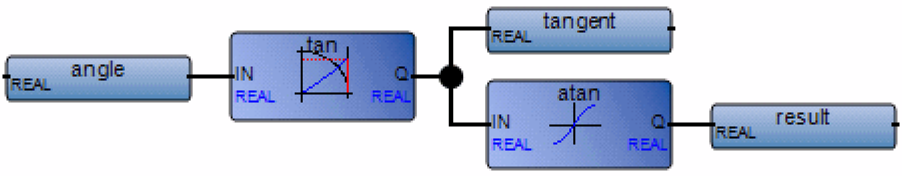
| | | | |
|---|---|---|---|
| IN | IN | REAL | Cannot be equal to PI/2 modulo PI |
| TAN | Q | REAL | Tangent of the input value<br>= 1E+38 for invalid input |

Description:

Calculates the Tangent of a real value.

### Example

(* FBD Program using "TAN" and "ATAN" Functions *)




(* ST Equivalence: *)

```
tangent := TAN (angle);
result := ATAN (tangent); (* result is equal to angle*)
```

# TRUNC


Arguments:

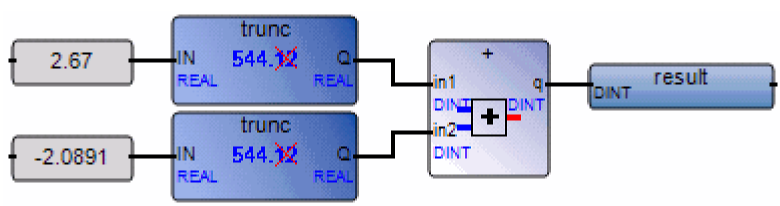
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any REAL value |
| TRUNC | Q | REAL | If IN>0, biggest integer less or equal to the input<br>If IN<0, least integer greater or equal to the input |

Description:

Truncates a real value to have just the integer part.

## Example

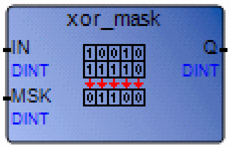
(* FBD Program using "TRUNC" Function *)




(* ST Equivalence: *)

```
result := TRUNC (+2.67) + TRUNC (-2.0891);
(* means: result := 2.0 + (-2.0) := 0.0; *)
```

# XOR_MASK




Arguments:

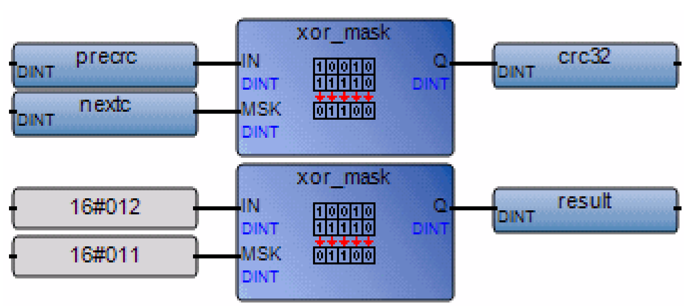
| | | | |
|---|---|---|---|
| IN | IN | DINT | Must have integer format |
| MSK | MSK | DINT | Must have integer format |
| XOR_MASK | Q | DINT | Bit-to-bit logical **Exclusive OR** between IN and MSK |

Description:

Integer exclusive OR bit-to-bit mask

## Example

(* FBD example with XOR_MASK functions *)




(* ST Equivalence: *)

```
crc32 := XOR_MASK (prevcrc, nextc);

result := XOR_MASK (16#012, 16#011); (* equals 16#003 *)
```

# Function Blocks

The following function blocks are supported:

| | | |
|---|---|---|
| **Alarms management** | LIM_ALRM | High/low limit alarm with hysteresis |
| **Boolean operations** | SR | Set dominant bistable |
| | RS | Reset dominant bistable |
| | R_TRIG | Rising edge detection |
| | F_TRIG | Falling edge detection |
| **Comparator** | CMP | Full comparison function block |
| **Counters** | CTU | Up counter |
| | CTD | Down counter |
| | CTUD | Up-down counter |
| **Data manipulation** | AVERAGE | Running average over N samples |
| **Process control** | DERIVATE | Differentiation according to time |
| | HYSTER | Boolean hysteresis on difference of reals |
| | INTEGRAL | Integration over time |
| | STACKINT | Stack of integer |
| **Semaphore manipulation** | SEMA | Manipulates a software semaphore |
| **Signal generation** | BLINK | Blinking Boolean signal |
| | SIG_GEN | Signal generator |
| **Time operations** | TON | On-delay timing |
| | TOF | Off-delay timing |
| | TP | Pulse timing |

**Note:** When new function blocks are created, they can be called from any language.

# AVERAGE


Arguments:

| | | |
|---|---|---|
| RUN | BOOL | TRUE=run / FALSE=reset |
| XIN | REAL | Any real Variable |
| N | DINT | Application defined number of samples |
| XOUT | REAL | Running average of XIN value |

Note: When setting or changing the value for N, you need to set RUN to FALSE, then set it back to TRUE.

Description:

Stores a value at each cycle and calculates the average value of all stored values. Only the latest N values are stored.

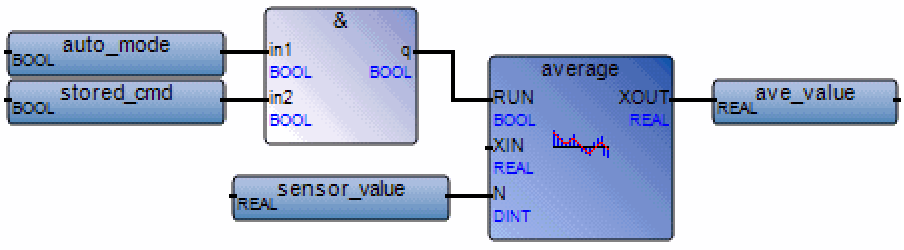
The maximum number of samples N is 128. When N exceeds 128, the number of samples is truncated to 128.

If the "RUN" command is FALSE (reset mode), the output value is equal to the input value.

Upon reaching the maximum N of stored values, the first stored value is overwritten with the latest value.

### Example

(* FBD program using the AVERAGE block: *)




(* ST Equivalence: AVERAGE1 instance of AVERAGE block *)

```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);

ave_value := AVERAGE1.XOUT;
```

# BLINK

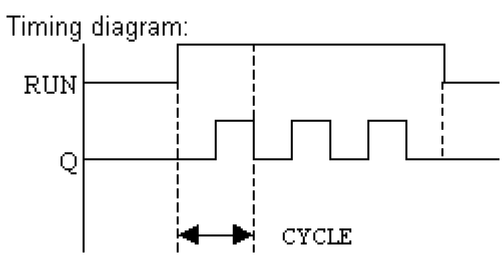

Arguments:

| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=blinking / FALSE=reset the output to false |
| CYCLE | TIME | Blinking period. Possible values range from 0ms to 23h59m59s999ms. |
| Q | BOOL | Output blinking signal |

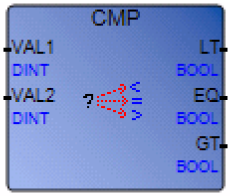
Description:

Generates a blinking signal.

Timing diagram:



# CMP




Arguments:

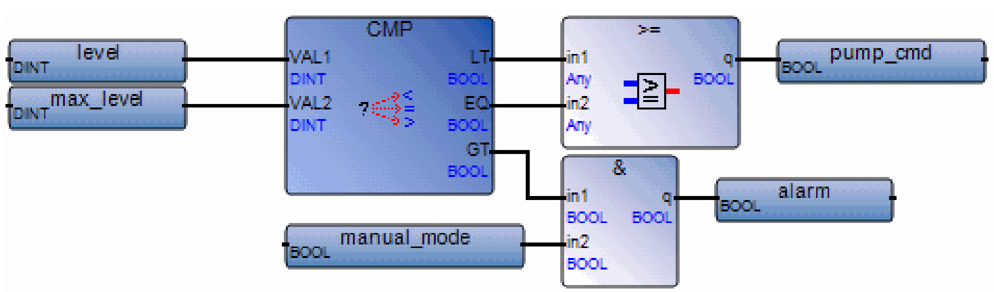
| | | |
|---|---|---|
| VAL1 | DINT | Any signed integer value |
| VAL2 | DINT | Any signed integer value |
| LT | BOOL | TRUE if val1 is less than val2 |
| EQ | BOOL | TRUE if val1 is equal to val2 |
| GT | BOOL | TRUE if val1 is greater than val2 |

Description:

Compare two values: tell if they are equal, or if the first is less or greater than the second one.

### Example

(* FBD program using the CMP block *)

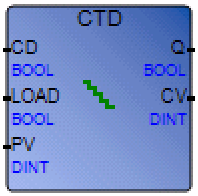



(* ST Equivalence: We suppose CMP1 is an instance of CMP block *)

```
CMP1(level, max_level);
```

```
pump_cmd := CMP1.LT OR CMP1.EQ;
alarm := CMP1.GT AND NOT(manual_mode);
```

# CTD




Arguments:

| | | |
|---|---|---|
| CD | BOOL | Counting input<br>(down-counting when CD is TRUE) |
| LOAD | BOOL | Load command (dominant)<br>(CV = PV when LOAD is TRUE) |
| PV | DINT | Programmed initial value |
| Q | BOOL | Underflow: TRUE when CV <= 0 |
| CV | DINT | Counter result |

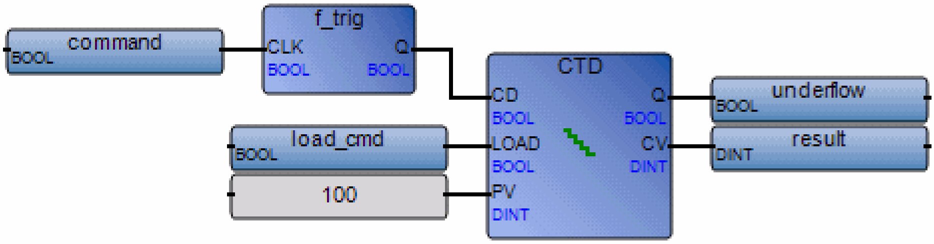
**Warning:** The CTD block does not detect the rising edges or falling edges of the counting input (CD). The block must be associated with an "R_TRIG" or "F_TRIG" block to create a pulse counter.

Description:

Count (integer) from a given value down to 0 1 by 1

### Example

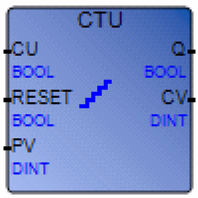
(* FBD program using the CTD block *)




(* ST Equivalence: We suppose F_TRIG1 is an instance of F_TRIG block and CTD1 is an instance of CTD block*)

```
F_TRIG1(command);

CTD1(F_TRIG1.Q,load_cmd,100);

underflow := CTD1.Q;

result := CTD1.CV;
```

# CTU




Arguments:

| | | |
|---|---|---|
| CU | BOOL | Counting input (counting when CU is TRUE) |
| RESET | BOOL | Reset command (dominant) |
| PV | DINT | Programmed maximum value |
| Q | BOOL | Overflow: TRUE when CV >= PV |
| CV | DINT | Counter result |

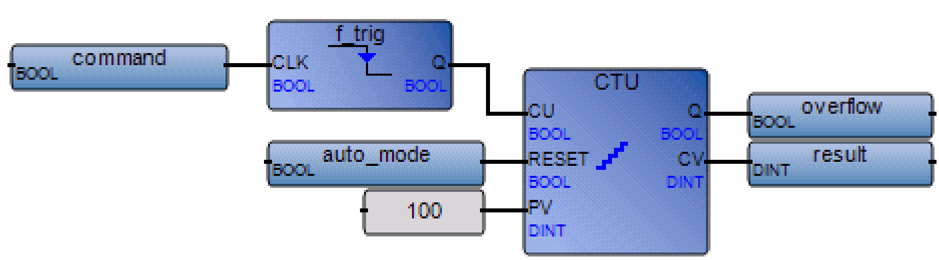
**Warning:** The CTU block does not detect the rising edges or falling edges of the counting input (CU). The block must be associated with an "R_TRIG" or "F_TRIG" block to create a pulse counter.

Description:

Count (integer) from 0 up to a given value 1 by 1

## Example

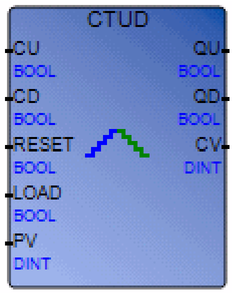
(* FBD program using the CTU block *)



(* ST Equivalence: We suppose R_TRIG1 is an instance of R_TRIG block and CTU1 is an instance of CTU block*)

```
R_TRIG1(command);

CTU1(R_TRIG1.Q,NOT(auto_mode),100);

overflow := CTU1.Q;

result := CTU1.CV;
```

# CTUD




Arguments:

| | | |
|---|---|---|
| CU | BOOL | Up-counting (when CU is TRUE) |
| CD | BOOL | Down-counting (when CD is TRUE) |
| RESET | BOOL | Reset command (dominant)<br>(CV = 0 when RESET is TRUE) |
| LOAD | BOOL | Load command (CV = PV when LOAD is TRUE) |
| PV | DINT | Programmed maximum value |
| QU | BOOL | Overflow: TRUE when CV >= PV |
| QD | BOOL | Underflow: TRUE when CV <= 0 |
| CV | DINT | Counter result |

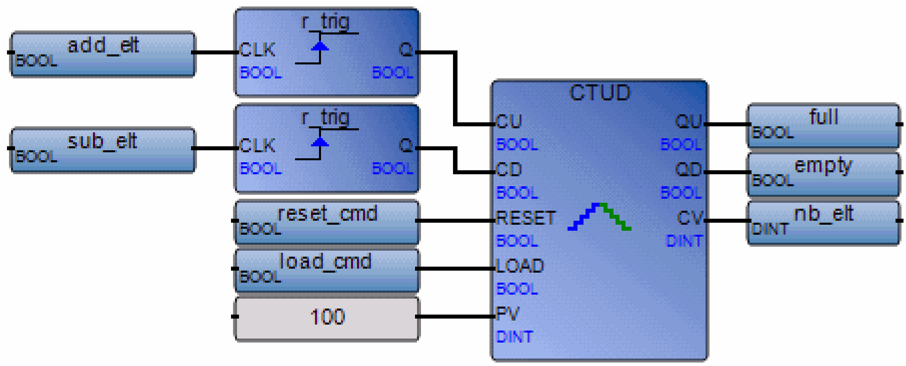
**Warning:** The CTUD block does not detect the rising edges and falling edges of the counting inputs (CU and CD). The block must be associated with an R_TRIG or F_TRIG block to create a pulse counter.

Description:

Count (integer) from 0 up to a given value 1 by 1 or from a given value down to 0 1 by 1

## Example

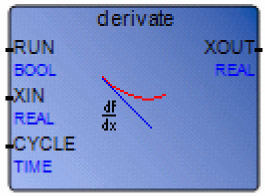
(* FBD program using the CTUD block *)




(* ST Equivalence: We suppose R_TRIG1 and R_TRIG2 are two instances of R_TRIG block and CTUD1 is an instance of CTUD block*)

```
R_TRIG1(add_elt);

R_TRIG2(sub_elt);

CTUD1(R_TRIG1.Q, R_TRIG2.Q, reset_cmd, load_cmd,100);

full := CTUD1.QU;

empty := CTUD1.QD;

nb_elt := CTUD1.CV;
```

# DERIVATE




Arguments:

| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=normal / FALSE=reset |
| XIN | REAL | Input: any real value |
| CYCLE | TIME | Sampling period. Possible values range from 0ms to 23h59m59s999ms. |
| XOUT | REAL | Differentiated output |

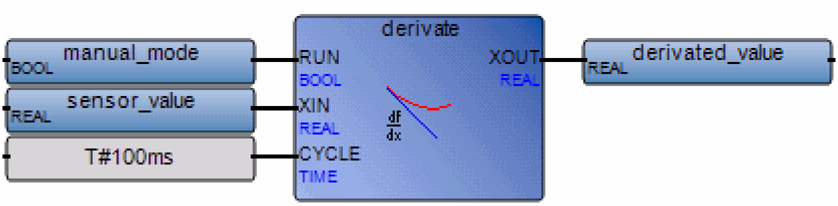
Description:

Differentiation of a real value.

If the "CYCLE" parameter value is less than the real duration of the cycle time in the virtual machine, the sampling period will use the real duration of the cycle time.

## Example

(* FBD program using the DERIVATE block: *)




(* ST Equivalence: DERIVATE1 instance of DERIVATE block *)

```
DERIVATE1(manual_mode, sensor_value, t#100ms);
```

```
derivated_value := DERIVATE1.XOUT;
```

# F_TRIG


Arguments:

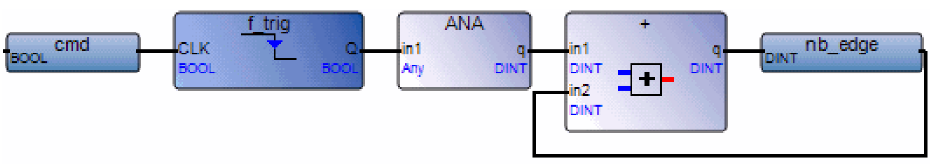
| | | |
|---|---|---|
| CLK | BOOL | Any Boolean Variable |
| Q | BOOL | TRUE when CLK changes from TRUE to FALSE<br>FALSE if all other cases |

Description:

Detects a falling edge of a Boolean variable

## Example

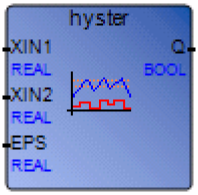
(* FBD program using the F_TRIG block *)




(* ST Equivalence: We suppose F_TRIG1 is an instance of F_TRIG block *)

```
F_TRIG1(cmd);
nb_edge := ANY_TO_DINT(F_TRIG1.Q) + nb_edge;
```

# HYSTER




Arguments:

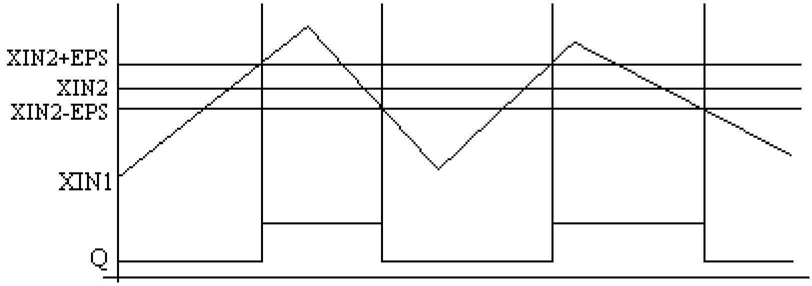
| | | |
|---|---|---|
| XIN1 | REAL | Any real value |
| XIN2 | REAL | To test if XIN1 has overpassed XIN2+EPS |
| EPS | REAL | Hysteresis value (must be greater than zero) |
| Q | BOOL | TRUE if XIN1 has overpassed XIN2+EPS and is not yet below XIN2-EPS |

Description:

Hysteresis on a real value for a high limit.

### Example

Example of a timing diagram:



# INTEGRAL


Arguments:

| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=integrate / FALSE=hold |
| R1 | BOOL | Overriding reset |
| XIN | REAL | Input: any real value |
| X0 | REAL | Initial value |
| CYCLE | TIME | Sampling period. Possible values range from 0ms to 23h59m59s999ms. |
| Q | BOOL | Not R1 |
| XOUT | REAL | Integrated output |

Description:

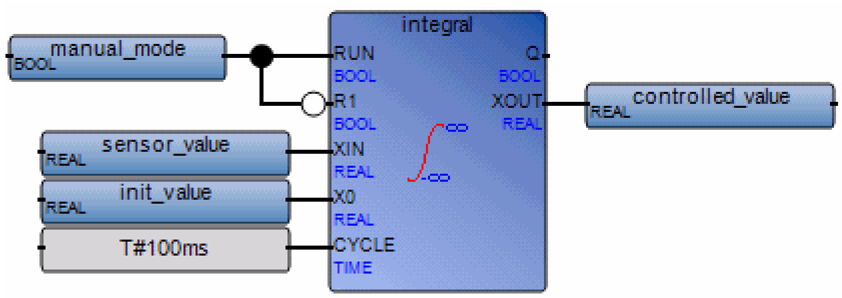
Integration of a real value.

If the "CYCLE" parameter value is less than the real duration of the cycle time in the virtual machine, the sampling period will use the real duration of the cycle time.

When using the Enable EN/ENO option for INTEGRAL blocks in LD POUs, you must reinitialize the internal variables for the R1 input. To reinitialize the R1 input, toggle the value from False to True then back to False.

### Example

(* FBD program using the INTEGRAL block: *)

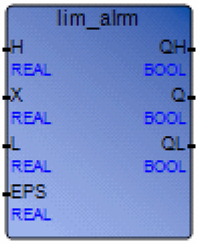



(* ST Equivalence: INTEGRAL1 instance of INTEGRAL block *)

```
INTEGRAL1(manual_mode, NOT(manual_mode), sensor_value, init_value,
t#100ms);
```

```
controlled_value := INTEGRAL1.XOUT;
```

# LIM_ALRM




Arguments:

| | | |
|---|---|---|
| H | REAL | High limit value |
| X | REAL | Input: any real value |
| L | REAL | Low limit value |
| EPS | REAL | Hysteresis value (must be greater than zero) |
| QH | BOOL | "high" alarm: TRUE if X above high limit H |
| Q | BOOL | Alarm output: TRUE if X out of limits |
| QL | BOOL | "low" alarm: TRUE if X below low limit L |

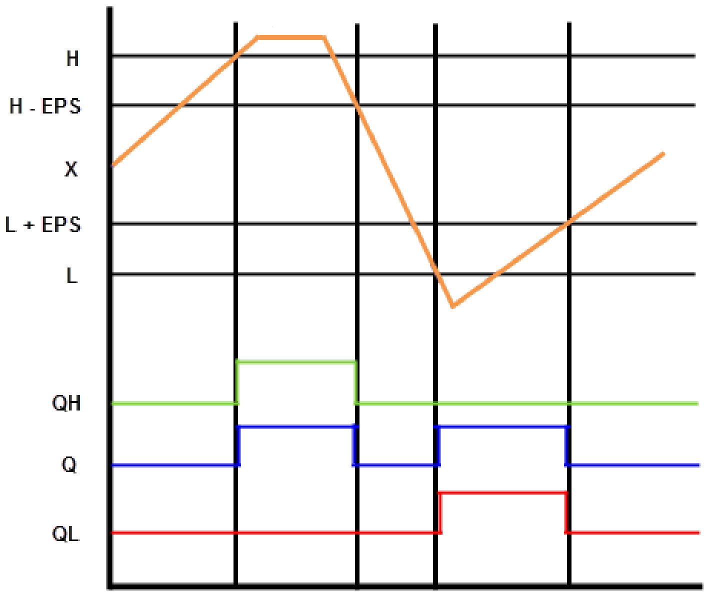
Description:

Hysteresis on a real value for high and low limits.

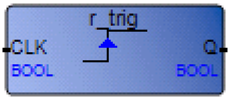
A hysteresis is applied on high and low limits. The hysteresis delta used for either the high or low limit is equal to the EPS parameter.

### Example

Example of timing diagram:


H
H - EPS
X
L + EPS
L
QH
Q
QL

## R_TRIG




Arguments:

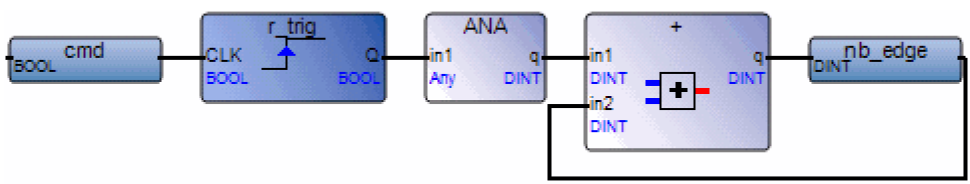
| | | |
|---|---|---|
| CLK | BOOL | Any Boolean Variable |
| Q | BOOL | TRUE when CLK rises from FALSE to TRUE<br>FALSE in all other cases |

Description:

Detects a rising edge of a Boolean variable

### Example

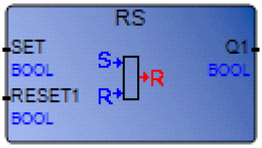
(* FBD program using the R_TRIG block *)




(* ST Equivalence: We suppose R_TRIG1 is an instance of the R_TRIG block *)

```
R_TRIG1(cmd);
nb_edge := ANY_TO_DINT(R_TRIG1.Q) + nb_edge;
```

# RS




Arguments:

| | | |
|---|---|---|
| SET | BOOL | If TRUE, sets Q1 to TRUE |
| RESET1 | BOOL | If TRUE, resets Q1 to FALSE (dominant) |
| Q1 | BOOL | Boolean memory state |

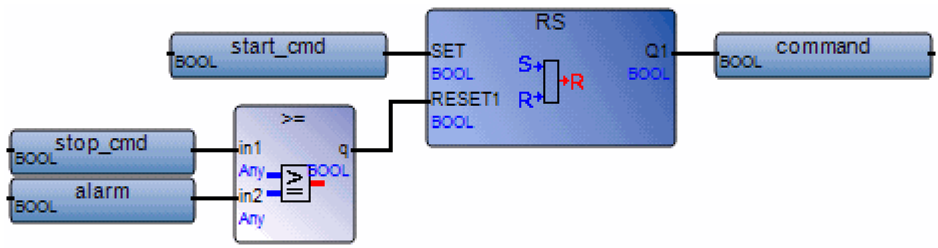
Description:

Reset dominant bistable:

| **Set** | **Reset1** | **Q1** | **Result Q1** |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

### Example

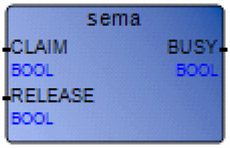
(* FBD Program using the RS block *)




(* ST Equivalence: We suppose RS1 is an instance of RS block *)

```
RS1(start_cmd, (stop_cmd OR alarm));
command := RS1.Q1;
```

# SEMA




**Note:** This operator is only available for **ISaGRAF 3** configurations.

Arguments:

| | | |
|---|---|---|
| CLAIM | BOOL | "test and set" command |
| RELEASE | BOOL | Releases the semaphore |
| BUSY | BOOL | State of the semaphore |

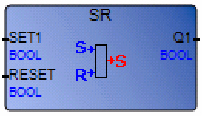
Description:

Manipulates a software semaphore.

(* "x" is a Boolean variable initialized to FALSE *)

```
busy := x;
If claim Then
  x := True;
Else
  If release Then
     busy := False;
     x := False;
  End_if;
End_if;
```

# SR




Arguments:

| | | |
|---|---|---|
| SET1 | BOOL | If TRUE, sets Q1 to TRUE (dominant) |
| RESET | BOOL | If TRUE, resets Q1 to FALSE |
| Q1 | BOOL | Boolean memory state |

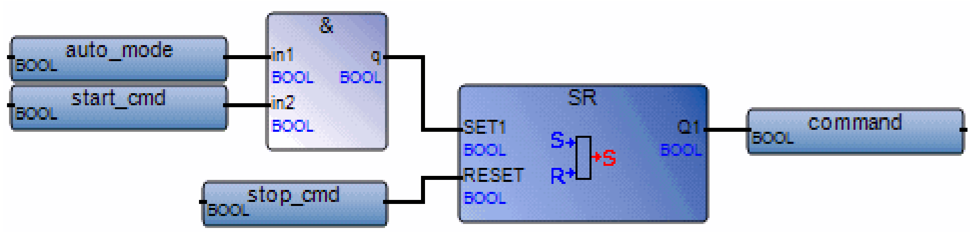
Description:

Set dominant bistable:

| Set1 | Reset | Q1 | Result Q1 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

## Example

(* FBD Program using the SR block *)




(* ST Equivalence: We suppose SR1 is an instance of SR block *)

```
SR1((auto_mode & start_cmd), stop_cmd);
command := SR1.Q1;
```

# SIG_GEN


Arguments:

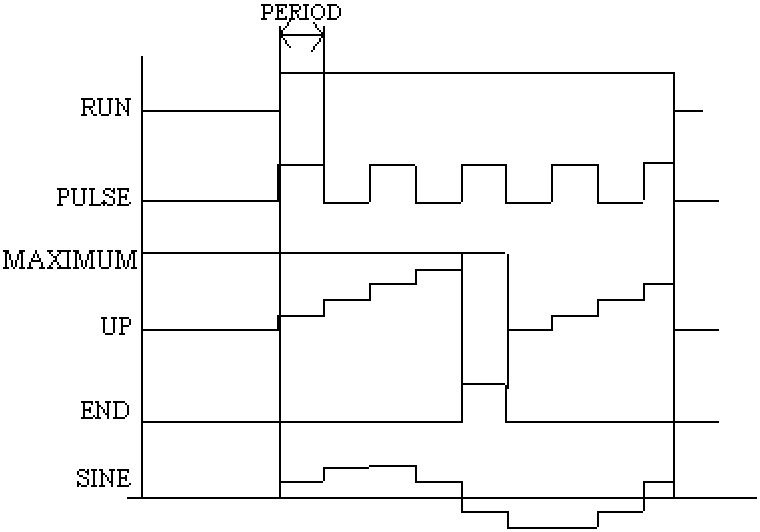
| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=running / FALSE=reset to false |
| PERIOD | TIME | Duration of one sample. Possible values range from 0ms to 23h59m59s999ms. |
| MAXIMUM | DINT | Maximum counting value |
| PULSE | BOOL | Inverted after each sample |
| UP | DINT | Up-counter, increased on each sample |
| END | BOOL | TRUE when up-counting ends |
| SINE | REAL | Sine signal (period = counting duration) |

Description:

Generates various signal: blink on a boolean, a integer counter-up, and real sine wave.

When counting reaches maximum value, it restarts from 0 (zero). So END keeps the TRUE value only during 1 PERIOD.

Timing diagram:


PERIOD
RUN
PULSE
MAXIMUM
UP
END
SINE

# STACKINT


Arguments:

| | | |
|---|---|---|
| PUSH | BOOL | Push command (on rising edge only)<br>add the IN value on the top of the stack |
| POP | BOOL | Pop command (on rising edge only)<br>delete in the stack the last value pushed (top of the stack) |
| R1 | BOOL | Resets the stack to its empty state |
| IN | DINT | Pushed value |
| N | DINT | Application defined stack size |
| EMPTY | BOOL | TRUE if the stack is empty |
| OFLO | BOOL | Overflow: TRUE if the stack is full |
| OUT | DINT | Value at the top of the stack |

Description:

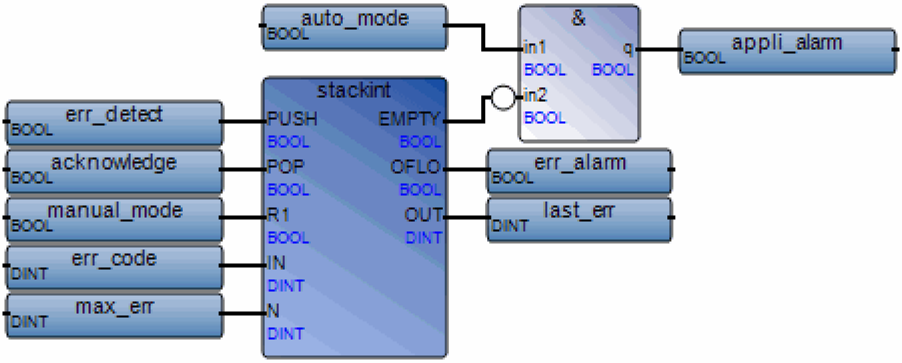
Manage a stack of integer values.

The STACKINT function block includes a rising edge detection for both PUSH and POP commands. The maximum size of the stack is 128. The application defined stack size N cannot be less than 1 or greater than 128.

**Note:** The OFLO value is valid only after a reset (R1 has been set to TRUE at least once and back to FALSE).

### Example

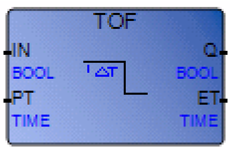
(* FBD program using the STACKINT block: error management *)




(* ST Equivalence: We suppose STACKINT1 is an instance of STACKINT block *)

```
STACKINT1(err_detect, acknowledge, manual_mode, err_code, max_err);
appli_alarm := auto_mode AND NOT(STACKINT1.EMPTY);
err_alarm := STACKINT1.OFLO;
last_error := STACKINT1.OUT;
```

# TOF




Arguments:

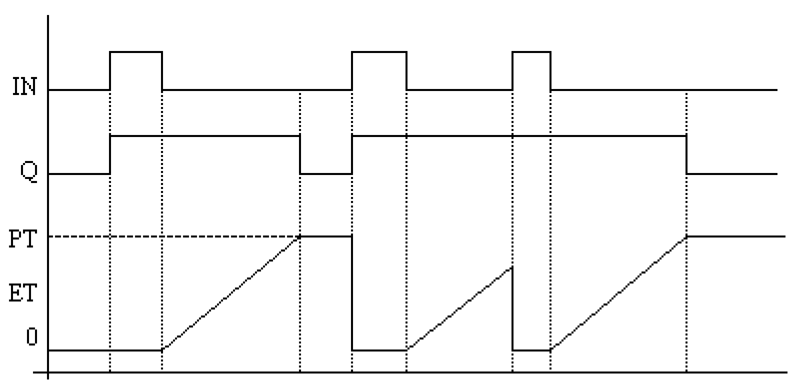
| | | |
|---|---|---|
| IN | BOOL | If falling edge, starts increasing internal timer<br>If rising edge, stops and resets internal timer |
| PT | TIME | Maximum programmed time |
| Q | BOOL | If TRUE: total time is not elapsed |
| ET | TIME | Current elapsed time |

Description:

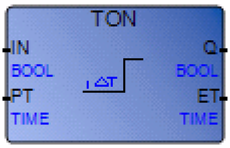
Increase an internal timer up to a given value.

While using the Enable EN/ENO option for LD POUs, execution disregards the TOF function block when EN is FALSE. When EN toggles from FALSE to TRUE, the function block is not reinitialized if IN is TRUE. To reinitialize the TOF function block, make sure IN is FALSE before setting EN to TRUE.

Timing diagram:



# TON




Arguments:

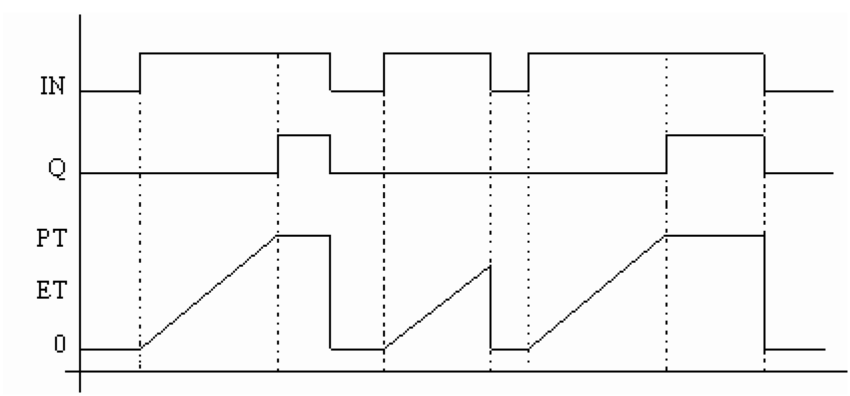
| | | |
|---|---|---|
| IN | BOOL | If rising edge, starts increasing internal timer<br>If falling edge, stops and resets internal timer |
| PT | TIME | Maximum programmed time |
| Q | BOOL | If TRUE, programmed time is elapsed |
| ET | TIME | Current elapsed time. Possible values range from 0ms to 23h59m59s999ms. |

Description:

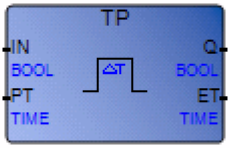
Increase an internal timer up to a given value.

While using the Enable EN/ENO option for LD POUs, execution disregards the TON function block when EN is FALSE. When EN toggles from FALSE to TRUE, the function block is not reinitialized if IN is TRUE. To reinitialize the TON function block, make sure IN is FALSE before setting EN to TRUE.

Timing diagram:



# TP




Arguments:

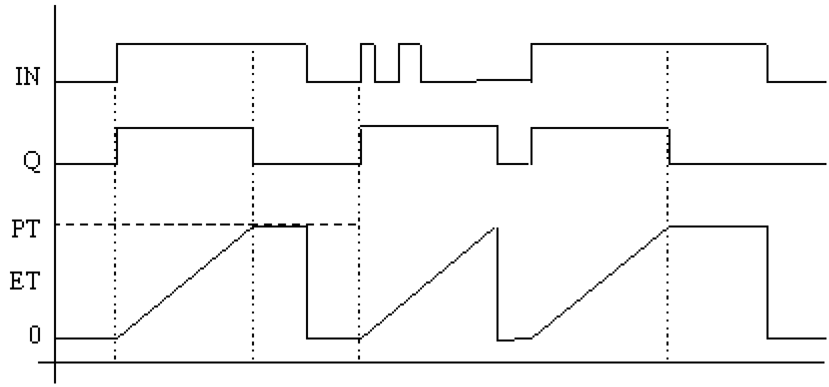
| | | |
|---|---|---|
| IN | BOOL | If rising edge, starts increasing internal timer (if not already increasing)<br>If FALSE and only if timer is elapsed, resets the internal timer<br>Any change on IN during counting has no effect. |
| PT | TIME | Maximum programmed time |
| Q | BOOL | If TRUE: timer is counting |
| ET | TIME | Current elapsed time. Possible values range from 0ms to 23h59m59s999ms. |

Description:

Increase an internal timer up to a given value.

While using the Enable EN/ENO option for LD POUs, execution disregards the TP function block when EN is FALSE. When EN toggles from FALSE to TRUE, the function block is not reinitialized if IN is TRUE. To reinitialize the TP function block, make sure IN is FALSE before setting EN to TRUE.

Timing diagram:


IN
Q
PT
ET
0

# Glossary

The Glossary contains terms used in **ISaGRAF** and their definitions.

To optimize a search for a definition, click one of the following letter groups in which you want to search.

A - C    D - H    I - N    O - R    S - Z

**A - C**

| | |
|---|---|
| **AAM** | Abstract Automation Model. Common interfaces used to access Concrete Automation Model data represented by IEC 61131 elements and concepts, as well as device management. |
| **Access Control** | The use of password-protection to control access to projects, devices, and POUs. For projects, devices, and POUs, access control can also limit access to read mode. |
| **ACP** | Automation Collaborative Platform. A set of software components and services through which plug-ins communicate. |
| **Action** | A collection of operations to perform whose execution differs for each programming language. |
| **Add-in** | Also known as a plug-in, it is a utility, driver, or other software added to a primary application. In the Visual Studio Integrated Development Environment (IDE), an add-in is an Automation-based application that extends the capabilities of the IDE. |
| **Address** | Optional hexadecimal address freely defined for each variable. This address can be used by an external application to access the value of the variable when the application is executed by the virtual machine. |
| **Alias** | The property of a variable indicating a short name for a variable. |
| **Application** | Built project using the Application Builder. |

| | |
|---|---|
| **Application Builder** | An integrated development environment used to build control applications, i.e. the workbench. |
| **Array** | Set of elements of the same type referenced by one or more indexes enclosed in square brackets and separated by commas. The index is an integer. Examples: tabi[2] or tabij[2,4]. |
| **Attribute** | The property of a variable indicating whether a variable is read, write, or read-write. |
| **Boolean (BOOL)** | Basic type that can be used to define a variable, a Parameter (POU) or an I/O board. A Boolean can be TRUE (1) or FALSE (0). |
| **C Function** | Function written with the "C" language, called from POUs, in a synchronous manner. |
| **C Language** | High level literal language used to access particularities of the target device. C language can be used to program functions, function blocks and conversion functions. |
| **CAM** | (Concrete Automation Model) Concrete project model enabling the creation of applications supporting multi-process control. |
| **Cell** | Elementary area of the graphic matrix for graphic languages or for the Dictionary. A cell is defined as one box in the grid. |
| **CFB** | Indicates a C function block |
| **CFU** | Indicates a C function |
| **Channel** | A channel of an I/O board represents a hardware I/O point. A channel is either an input or output. A variable is wired to a channel to be used in POUs. Directly represented variables can also be used in POUs. |
| **Child** | A POU which is activated by its parent. The child POU has only one parent. Only the parent can start or stop the child program. A parent can have more than one child.<br>See also Parent Program. |

| | |
|---|---|
| **Clearing a Transition** | The forcing of the clearing of a transition where one of the previous steps is active. Tokens are moved and actions are executed as for a usual transition clearing. All tokens existing in the preceding steps are removed. A token is created in each of the following steps. |
| **Coil** | A graphic component representing the assignment of an output or an internal variable. |
| **Common Scope** | Scope of a declaration applying to all POUs and common to all projects within a specific installation of the workbench. Only defined words can have common scope. The following file contains all defined words having the common scope: %ALLUSERSPROFILE%\ISaGRAF\6.*x*\CAM ISaGRAF 3\Standard 3.55\COMMON.EQV |
| **Complex Equipment** | Element grouping multiple I/O boards. This provides the means for manufacturers to mix data types and directions. The implementation of the I/O driver for complex equipment corresponds to the implementation of the drivers of all contained I/O boards. OEM parameters enable providing parameters to complex equipment. |
| **Connection** | The link between networks and devices. |
| **Constant Expression** | Literal expression used to describe a constant value. |
| **Contact** | Depending on the type of contact, a graphic component representing the value of an input or an internal variable. |
| **Contextual Menu** | Menu that is displayed under the mouse cursor by right-clicking the mouse. |
| **Conversion Function** | "C" written Function which implements a conversion. Such a conversion can be attached to any input or output channel. The conversion is applied each time the input variable is read or the output variable is written. |
| **CRC** | The virtual machine compares the Cyclic redundancy checking (CRC) values for compiled, running, and stored versions of code to detect possible mismatches. |

**CSV File Format** (Comma Separated Values) A delimited data format having each piece of information separated by commas, where text strings, including comments, are surrounded by quotation marks ("), and each line ending with a carriage return. The CSV file format can be used for importing or exporting variable properties.

**Cycle** The virtual machine executes the programs of a device as a cycle. All programs of the device are executed following the order defined by the user, from the first program to the last and again and again. Before the execution of the first program, inputs are read. After the execution of the last program, the outputs are refreshed.

**Cycle Timing** The amount of time given to each virtual machine cycle. The cycle consists of scanning the physical inputs, executing the POUs of the device, then updating physical outputs. If a cycle is completed within the specified cycle timing period, the system waits until this period has elapsed before starting a new cycle. The cycle time can differ for each cycle when no cycle timing is specified. When the cycle timing is shorter, the virtual machine waits until this time has elapsed. When the cycle time is longer, the virtual machine immediately scans the inputs but signals with the "overflow" that the programmed time has been exceeded. When the trigger cycles property is false or the cycle time is 0, the virtual machine does not wait to start a new cycle.

**Cycle-to-cycle Mode** Execution mode of a device where cycles are executed one by one, according to commands given by the user during debugging. Another execution mode for the virtual machine is real-time.

**Cyclic Program** A time independent program that is executed during each cycle. A cyclic program can be executed before and after sequential programs.

## D - H

| | |
|---|---|
| **Data Type** | Data types are defined for many items in **ISaGRAF** projects:<br>- variables<br>- function or function block parameters<br>- I/O boards<br>See also Standard IEC 61131 Types, User Types. |
| **Database** | The collection of definitions making up an **ISaGRAF** project. |
| **Debugging** | The process of detecting defects in a project that includes cycle-to-cycle debugging, setting and clearing breakpoints. |
| **Declared Instance (of a function block)** | A function block having assigned instances, i.e., declared in the dictionary. A declared instance is only available in ST. |
| **Defined Word** | Equivalent expression for use in POUs. At compiling time the word is replaced by the expression. A defined word cannot use a defined word. |
| **Dependency (on a library)** | The state where a project uses, i.e., depends, on functions or function blocks defined in a library. |
| **Design (mode)** | An editing mode during which the Application Builder is not connected to the device. |
| **Device** | An instance of a target platform in the application builder.<br>See also Target Platform. |
| **Device Management** | Provides the communication infrastructure with the target platform. |
| **Dictionary** | The grid view displaying the variables, function and function block parameters, types, and defined words used in the programs of a project. |
| **Dimension** | The size (number of elements) of an array. For example: [1..3,1..10] - represents a two-dimensional array containing a total of 30 elements. |
| **DINT** | Signed double integer 32-bit format. Basic type that can be used to define a variable, a Parameter (POU) or an I/O Device. |
| **Direction** | Variables and I/O devices have a direction. For the property of a variable, direction indicates whether a variable is an input, output, or internal. The direction of an I/O device can be input or output. |

| | |
|---|---|
| **Directly Represented Variable** | A variable is generally declared before its use in one POU. A directly represented variable is used in a program to represent a channel for an I/O device. Example: %QX1.6, %ID8.2 |
| **Dynamic Behavior** | Continuous and sequential execution of the steps and operations of an SFC program during an execution cycle. |
| **Edge** | See Falling Edge, Rising Edge. |
| **Execution Mode** | The mode in which a device is executed: real-time and cycle-to-cycle. |
| **Expression** | Set of operators and identifiers. |
| **Falling Edge** | A falling edge of a boolean variable corresponds to a change from TRUE (1) to FALSE (0). |
| **FBD** | Function Block Diagram. Programming language. |
| **Function** | POU which has input parameters and one output parameter. A function can be called by a program, a function or a function block. A function has no instance. This signifies that local data is not stored and is generally lost from one call to the other. |
| **Function Block** | POU which has input and output parameters and works on internal data (parameters). A program can call an instance of a function block. A function block instance cannot be called by a function (no internal data for a function). A function block can call another function block (instantiation mechanism is extended to the function blocks called). |
| **Global Scope** | Scope of a declaration applying to all POUs of the current project. |
| **Global Variable** | A variable whose scope is global. |
| **Hidden Parameter** | Input parameters of a function block that are not displayed in programs. |
| **Hierarchy** | Architecture of a project, divided into several POUs. The hierarchy tree represents the links between parent programs and children programs.<br>See also Parent Program. |

## I - N

| | |
|---|---|
| **I/O Board** | An I/O board corresponds to a piece of equipment having inputs or outputs. OEM parameters enable providing parameters to I/O boards. Integrators define I/O boards. |
| **I/O Channel** | See Channel. |
| **I/O Device** | Element grouping several channels of the same data type and direction. These can be either an I/O board or a complex equipment. |
| **I/O Driver** | "C" code which makes the interface between a virtual machine and the devices. The driver can be statically linked to the virtual machine or in a separate DLL (such as for the Windows NT target). |
| **I/O Variable** | Variable connected to a channel of an I/O device. An array can be connected to an I/O device if all elements are connected to contiguous channels, the type of the array must be the same type as the I/O device. |
| **I/O Wiring** | Definition of the links between the variables of the project and the channels of the I/O devices existing on the target platform. |
| **Identifier** | Unique word used to represent a variable or a constant expression in the programming. |
| **IFB** | Indicates an IEC 61131 user-defined function block |
| **IFU** | Indicates an IEC 61131 user-defined function |
| **Initial Situation** | Set of the initial steps which represents the context of the program when it is started. |
| **Initial Step** | A Step that is activated when the program starts. |
| **Initial Value** | Value which has a variable when the virtual machine starts the execution of the application. The initial value of a variable can be the default value, a value given by the user when the variable is defined or the value of the retain variable after the virtual machine has stopped. |
| **Input** | Direction of a variable or an I/O device. An input variable is connected to an input channel of an input device. |

| | |
|---|---|
| **Input Parameter** | Input argument of a function or a function block. These parameters can only be read by function or function block. A parameter is characterized by a type. |
| **Instance (of a Function Block)** | A variable containing a copy of the internal data of a function block persisting from one call to the other. This word is used, by extension, to say that a program calls a function block instance and not the function block itself. |
| **Instruction** | An elementary operation of a program, entered on one line of text. |
| **Internal** | Attribute of a variable, which is not linked to an I/O device. Such a variable is called an internal variable. |
| **Label** | The identifier for an instruction within a program. Labels can also be used for jump operations. |
| **Language Container** | A workspace enabling the development of graphic or textual POUs programmed using one of the available programming languages. Individual language containers can only use one programming language. When editing a container, the toolbox displays the corresponding elements for the specific programming language. The multi-language editor (MLGE) enables the creation of language containers. |
| **LD** | Ladder Diagram. Programming language. |
| **Library** | Special projects made up of devices in which you define functions and function blocks for reuse throughout **ISaGRAF** projects. Libraries also enable you to modularize projects and to isolate functions and function blocks so that these can be validated separately. |
| **Link** | A graphic component connecting elements in a network diagram. |
| **Literal** | A lexical unit that directly represents a value. |
| **Local scope** | Scope of a declaration applying to only one POU. |
| **Maximum time** | Time of the longest cycle since the virtual machine has started the execution of the application. |
| **MESSAGE** | Character string. Basic type available for defining a variable, a parameter (POU) or a device. |

| | |
|---|---|
| **MLGE** | Multi-language Editor. |
| **Modbus** | A communications protocol using programmable logic controllers (PLCs). |
| **Monitoring** | A process by which the user views virtual machine running states, system events, target capability, network card status, and various online statistics in a read format. |
| **MSI** | Windows installers (.msi) used to install applications and files typically used by the end user of the application. |
| **Network** | The term network is used in different contexts:<br>- The means of communication between the target platform and their clients.<br>- For the execution order of graphic programs, a sequence of connected blocks. |
| **Network Driver** | "C" code which makes the interface between the vitual machine network layer and the physical network. |
| **Non-stored Action** | A list of statements, executed at each target cycle, when the corresponding step is active. |

## O - R

| | |
|---|---|
| **OEM** | Original Equipment Manufacturer |
| **OEM Parameter** | Parameters attached to an IO device. A parameter is characterized by a type. An OEM parameter is defined by the designer of the device. It can be a constant, or a variable parameter entered by the user during the I/O connection. |
| **Online Mode** | Mode in which the Application Builder is connected to a target enabling target management, monitoring and debugging. |
| **OPE** | Indicates an operator. |
| **Operator** | Basic logical operation such as arithmetic, boolean, comparator, and data conversion. |
| **Output** | Direction of a variable or an I/O device. An output variable is connected to an output channel of an output device. |

| | |
|---|---|
| **Output Parameter** | Output argument of a function or function block. These parameters can only be written by a function or function block. A function has only one output parameter. A parameter is characterized by a type. |
| **Overflow** | Integer value which corresponds to the number of times the cycle time has been exceeded. Always 0 if cycle time is 0. |
| **Package** | The Target Definition Builder enables OEMs to provide packages containing the drivers of several I/O devices and/or "C" functions and function blocks available for a specific target.<br><br>See also Plug-in |
| **Parameter (POU)** | See Input Parameter, Output Parameter, and Hidden Parameter |
| **Parent Program** | A program which controls other programs, called its children. See also Child. |
| **PLC** | Programmable Logic Controller |
| **Plug-in** | A Visual Studio-based add-in or package integrated into a broad platform enabling the extension of the Workbench or the Automation Collaborative Platform. |
| **POU** | Program Organization Unit: set of instructions that are programs, functions or function blocks. |
| **Power Rail** | Main left and right vertical rails at the extremities of a ladder rung. |
| **Program** | See POU. A program belongs to a project. It is executed by the Virtual Machine, depending on its location (order) in the device. |
| **Project** | Set of programs making up an application. |
| **Project Updater** | A program allowing to convert projects developed using previous versions for use within the latest version. Each time you upgrade to a newer version, you need to update projects. |
| **Pulse Action** | A list of statements executed only once when the corresponding step is activated. |
| **Qualifier** | Determines the way the action of a step is executed. The qualifier can be N, S, R, P0 or P1. |

| | |
|---|---|
| **REAL** | Type of a variable, stored in a floating IEEE single precision 32-bit format. Basic type that can be used to define a variable, a parameter (POU) or a device. |
| **Real I/O Device** | I/O device physically connected to an I/O driver on the target. See also Virtual I/O Device. |
| **Real-time Mode** | The virtual machine normal execution mode of an application where execution cycles are triggered by the cycle timing. Another execution mode for applications is cycle-to-cycle. |
| **Reserved Keyword** | Reserved identifier of the languages unavailable for use as names of POUs or variables. |
| **Retain** | Attribute of a variable. The value of a retain variable is saved by the Virtual Machine at each cycle. The value stored is restored if the Virtual Machine stops and restarts. |
| **Return** | Graphic component of a program representing the conditional end of a program. |
| **Return Parameter** | See Output Parameter. |
| **Rising Edge** | A rising edge of a Boolean variable corresponds to a change from FALSE (0) to TRUE (1). |
| **Rung** | Graphic component of a program representing a group of circuit elements leading to the activation of a coil in an LD diagram. A rung is situated between left and right power rails. |
| **Run-time Error** | Application error detected by the virtual machine. |

## S - Z

| | |
|---|---|
| **Scope** | See Global Scope, Common Scope, Local scope. |
| **Section** | Program, Function and Function block sections are where are located the POUs of a device. POUs located in the Program section are executed by the virtual machine. |
| **Security State** | The indication of the level of access control that is applied to a device, a POU, or a project. |

| | |
|---|---|
| **Selection List** | Also known as a 'combo-box'.<br>When a selection list is provided for a particular cell, clicking on its right part (down arrow), displays the available choices. To make a selection, perform one of the following operations:<br>- click on the item (use the scroll bar first if the required choice is not visible)<br>- move in the list using the cursor keys and press Enter<br>- type the first letter (if more than one item starts with this letter, press the letter again to select the next occurrence). |
| **Separator** | Special character (or group of characters) used to separate the identifiers in a literal language. |
| **Sequential Program** | A program that is executed according to the dynamic behavior of the programming language and where the time variable explicitly synchronizes operations. |
| **Server** | The part of the target that receives Modbus requests to retrieve information about the device run by the virtual machine. |
| **SFB** | Indicates a function block |
| **SFU** | Indicates a function |
| **Shape** | The spatial form or appearance of an object. |
| **Simulation Mode** | Mode in which virtual machines execute the code of the device and the Windows platform performs aspects such as POU execution. |
| **Solution** | A container holding projects and libraries. A solution contains elements that represent the references, data connections, folders, and files needed to make up an application. |
| **Solution Explorer** | A view with a tree-like structure enabling the management of items such as devices, programs, functions, function blocks and dictionaries. |
| **ST** | Structured Text. Programming language. |
| **Standard IEC 61131 Types** | Double integer (DINT), Boolean (BOOL), REAL, TIME, and MESSAGE.<br>See also Data Type. |
| **Statement** | Basic ST complete operation. |

| | |
|---|---|
| **Step** | A basic graphic component representing a steady situation of the process. A step is referenced by a name. The activity of a step is used to control the execution of the corresponding actions.<br>See also Action. |
| **Sub-program** | A program called by a Parent Program. A sub-program is also called a Child program. To call sub-programs written in another language, use a function. A function can be called by any POU. |
| **Symbol Table** | The appli.txt text file corresponding to the variables defined for an application. This file is downloaded onto the target platform. |
| **Target Management** | Operations that control the application of a target including downloading, uploading, starting, stopping, and discovering. |
| **Target Platform** | The hardware platform on which virtual machines run. |
| **TIC** | Target Independent Code produced by the **ISaGRAF** compiler for execution on virtual machines. |
| **Timer (TIME)** | Unit of a timer is the millisecond. Basic type that can be used to define a Variable, a Parameter (POU) or an I/O Device. |
| **Token (SFC)** | Graphical marker used to identify the active steps of an SFC program. |
| **Tool Window** | A Microsoft Visual Studio control that enables application creation and editing. |
| **Toolbox** | The utility containing the elements and shapes available for language and ISaVIEW containers. For language containers, the available elements differ for the individual programming languages. |
| **Top Level Program** | Program put at the top of the hierarchy tree. A top level program is activated by the system.<br>See also Parent Program. |
| **Transition** | A basic graphic component representing the condition between different steps. A transition is referenced by a name. A Boolean condition is attached to each transition. |

| | |
|---|---|
| **Trigger Cycles** | Application property indicating whether a virtual machine cycle executes according to a defined cycle timing. |
| **User Data** | User data is any data of any format (file, list of values) which have to be merged with the generated code of the device in order to download them into the target platform. Such data is not directly operated by the virtual machine and is commonly dedicated to other software installed on the target platform. |
| **User Types** | Types that the user can define using basic types or other user types. User types can be arrays. |
| **User-Defined Function Block** | A custom function block. You create user-defined function blocks in the Function Blocks section for a device. |
| **Validity of a Transition** | Attribute of a transition. A transition is validated (or enabled) when all the preceding steps are active. |
| **Variable** | Unique identifier of elementary data used as information placeholders within POUs. Variables also include function block instances. |
| **Variable Name** | A unique identifier, defined in **ISaGRAF**, for a storage location containing information used in exchanges between devices. |
| **Virtual I/O Device** | I/O Device which is not physically connected to an I/O driver on the target platform. |
| **Virtual Machine** | The compiled **ISaGRAF** software running on the target platform. |
| **VS2010** | Microsoft Visual Studio 2010. |
| **Wiring** | The property of a variable indicating the I/O channel to which the variable is wired.<br>See also I/O Wiring |

# Licensing

**ISaGRAF** enables the creation of virtual machines running on hardware components, called targets.

**ISaGRAF** is available in two types of software licenses:

- Demo version, delivered with the product and available for testing the product. This is a 60 day trial of the fully operational version of **ISaGRAF**.
- Integrated license, included in the installation of the **ISaGRAF** software. The product is licensed upon installation. The Integrated license is available as a Full license or a Limited license. A Full license is a fully operational version of the product while a Limited license can only have one device.
- Engineering license, obtained by manually activating an unlicensed version of the product. The Engineering license is available as a Full license or a Limited license. A Full license is a fully operational version of the product while a Limited license can only have one device.

The Integrated and Engineering licenses are available for the following activation periods:

- Lifetime (does not expire)
- 1 month
- 6 months
- 12 months

**To access Licensing**

1. From the Help menu, click **Licensing CAM 3.**

The Licensing for the **ISaGRAF 3** Concrete Automation Model is displayed.

**To obtain an authorized Engineering license**

1. From the Help menu, click **Licensing CAM 3.**

   The Licensing for the **ISaGRAF 3** Concrete Automation Model is displayed along with three User Codes.

2. Send an e-mail containing the desired activation period and the three User Codes to the support team:
   support@ISaGRAF.com

3. The support team will email you back Registration Keys 1 and 2.

4. Insert the Registration Keys in their appropriate regions and click **Validate**.

   **ISaGRAF** is now licensed.

**To remove an authorized license**

1. From the Help menu, click **Licensing CAM 3.**

   The Licensing for the **ISaGRAF 3** Concrete Automation Model is displayed along with three User Codes.

2. Send an email containing the three User Codes to the support team:
   support@ISaGRAF.com

3. The support team will email you back Registration Keys 1 and 2.

4. Insert the Registration Keys in their appropriate regions and click **Validate**.

   A confirmation code appears.

5. Send an email containing the confirmation code to the support team:
   support@ISaGRAF.com

   **ISaGRAF** is no longer licensed.

# ISaGRAF 5 Concrete Automation Model

The **ISaGRAF 5** Concrete Automation Model enables the creation of **ISaGRAF 5** applications supporting multi-process control. Applications consist of virtual machines running on hardware components, called targets. The development process consists of creating projects made up of devices, representing individual target platforms, on which one or more instances of resources are downloaded. At runtime, instances of resources become individual virtual machines running on these target platforms.

Projects can be developed using different programming languages including some from the IEC 61131-3 standard. When building, resources are compiled to produce very fast "target independent code" (TIC) or "C" code.

Within resources, you can declare variables using standard IEC 61131-3 data types (i.e., Boolean, integer, real, etc.) or user-defined types such as arrays or structures.

You develop projects on a Windows development platform. The **Automation Collaborative Platform** graphically represents and organizes devices, resources, POUs, and networks within a project from many views.

You can choose to simulate the running of a project, after building a project, using high-level debugging tools, before actually downloading the resources making up devices to the target platforms.

# Creating a Project

You can create projects as part of new or existing solutions in the Automation Collaborative Platform. A solution can hold multiple projects and libraries. You can import **ISaGRAF 6** projects that were previously exported in the 7-Zip (.7z) compressed file format.

The following templates are available for **ISaGRAF 6** projects:

- Import ISaGRAF 5 Project
- Import ISaGRAF Zip Project
- ISaFREE_TPL
- Library
- PRJ61499_TPL
- Simulator

For projects, you can specify the following properties:

**CAM**

| | |
|---|---|
| CAM Project | Type of project consisting of the CAM name and version. For example, ISaGRAF 5.23. |
| Comment | Text displayed next to the project name in the Solution Explorer |
| Description | Free-form text describing a project |
| Is Password Protected | Indication that the project is protected by a password controlling its access |
| Name | Name of the project. Project names can have up to 128 characters. |
| Online Behavior | Behavior of the project when switching to online or simulation mode. Possible options are the following:<br>- Design where the project remains in design mode when switching to online or simulation mode<br>- Debug Only where the project remains in design mode until switching to online mode; switching to simulation mode does not affect the project<br>- Simulate Only where the project remains in design mode until switching to simulation mode; switching to online mode does not affect the project<br>- Always where the project switches to either online or simulation mode |

| | |
|---|---|
| Path | Complete path where the Concrete Automation Model (CAM) project file stored on the computer. The path is automatically assigned:<br>%USERPROFILE%\My Documents\ ISaGRAF 6.x\Projects*SolutionName**ProjectName**ProjectName* |
| **Info** | |
| Name | Name of the project. Project names can have up to 128 characters. |
| Path | Complete path where the Automation Collaborative Platform (ACP) project file stored on the computer. The path is automatically assigned:<br>%USERPROFILE%\My Documents\ ISaGRAF 6.x\Projects*SolutionName**ProjectName* |

You can add devices and add resources to existing projects.

Projects are stored in the Projects directory, as MS-Access database (.MDB) files:

%USERPROFILE%\My Documents\ISaGRAF 6.x\Projects

#### To create a project

1. From the File menu, point to **New**, and then click **Project** (or press **Ctrl+Shift+N**).
2. In the Project Types list, expand the *CAM Projects* option, then click **ISaGRAF 5**.
3. From the list of available project templates, click the required template.
4. Specify a name and location for the project, indicate whether to add the project to an existing solution or create a new solution by defining a solution name, then click **OK**. For new solutions, you can choose to create a directory.

#### To import an ISaGRAF project

You can import ISaGRAF 6 projects that were previously exported in the 7-Zip (.7z) compressed file format.

1. From the File menu, point to **New**, and then click **Project**.

2. In the Project Types list, expand the **CAM Projects** option, then click **ISaGRAF 5**.

3. In the *ISaGRAF installed templates* list, click **Import ISaGRAF 5 Project**.

4. Specify a name and location for the project, indicate whether to add the project to an existing solution or create a new solution by defining a solution name, then click **OK**. For new solutions, you can choose to create a directory.

5. In the Choose an *.mdb File dialog box, locate and select the **ISaGRAF 5** project database file (*.mdb) from the previous **ISaGRAF** version, then click **Open**.

**To import a compressed project**

You can import projects created using previous versions of **ISaGRAF 5** and saved in the 7-Zip (.7z) compressed file format.

1. From the File menu, point to **New**, then click **Project**.

2. In the Project Types list, expand the **CAM Projects** option, then click **ISaGRAF 5**.

3. In the *ISaGRAF installed templates* list, click **Import ISaGRAF Zip Project**.

4. Specify a name and location for the project, indicate whether to add the project to an existing solution or create a new solution by defining a solution name, then click **OK**. For new solutions, you can choose to create a directory.

5. In the Choose a File dialog box, locate and select the compressed **ISaGRAF 5** project file from the previous **ISaGRAF** version, then click **Open**.

**To create a library project**

You can create a library project.

1. From the File menu, point to **New**, and then click **Project**.

2. In the Project Types list, expand the **CAM Projects** option, then click **ISaGRAF 5**.

3. In the *ISaGRAF installed templates* list, click **Library**.

4. Specify a name and location for the project, indicate whether to add the project to an existing solution or create a new solution by defining a solution name, then click **OK**. For new solutions, you can choose to create a directory.

### See Also

Creating a Library

# Devices

A device corresponds to a programmable logic controller. Devices contain one or more resources. You can perform the following tasks for devices from the Solution Explorer:

- Adding devices
- Renaming devices
- Deleting devices

For devices, you need to specify the following properties:

**Hardware**

| | |
|---|---|
| Enhanced Target | Indication of whether the device supports the enhanced target including motion control function blocks, safety function blocks, and cycle time in microseconds. |
| Memory Size | Memory allocated by the compiler for hidden temporary variables used while solving complex expressions |
| Support IEC 61850 | Indication of whether the device supports the IEC 61850 standard for the design of electrical substation automation. Using this feature requires importing a structure related to the standard. |
| Target | Target type to which is attached the device. Changing targets for a device affects all resources attached to the device. |

**Info**

| | |
|---|---|
| Comment | Text displayed next to the device name in the Solution Explorer |
| Description | Free-form text describing a device |
| Full Name | Full name of device using the following syntax: *ProjectName.DeviceName* |
| Is Password Protected | Indication that the device is protected by a password controlling its access |
| Name | Name of the device. Device names can have up to 128 characters. |

| | |
|---|---|
| Path | Complete path where the device files are stored on the computer. The path is automatically assigned to a device folder within the project folder:<br>%USERPROFILE%\My Documents\<br>ISaGRAF 6.x\Projects*SolutionName**ProjectName**ProjectName*\\<br>*DeviceName* |
| **Settings** | |
| Online Behavior | Behavior of the device when switching to online or simulation mode. Possible options are the following:<br>- Design where the device remains in design mode when switching to online or simulation mode<br>- Debug Only where the device remains in design mode until switching to online mode; switching to simulation mode does not affect the device<br>- Simulate Only where the device remains in design mode until switching to simulation mode; switching to online mode does not affect the device<br>- Always where the device switches to either online or simulation mode |

**To add a device**

- In the Solution Explorer, right-click the project element, point to **Add**, and then click **New Device**.

**To rename a device**

- In the Solution Explorer, right-click the device, click **Rename**, and then type a name for the device.

**To delete a device**

Deleting a device also removes all resources belonging to the device.

- In the Solution Explorer, right-click the device, and then click **Delete**.

**See Also**

Creating a Project

Creating a Library

# Resources

Resources contain the POUs (programs, functions, and function blocks) and definitions within devices. You can create user-defined functions and user-defined function blocks in the Lib section of the Solution Explorer. You can perform the following tasks for resources from the Solution Explorer:

- Adding resources
- Renaming resources
- Deleting resources

For resources, you need to specify the following properties:

**Code**

| | |
|---|---|
| Code For Simulation | Indication of whether to produce code for simulation for an application |

| | |
|---|---|
| Compiler Options | Check Array Index - Indication of whether to verify array indices<br><br>Dump Configuration Files - Indication of whether to generate of resource level files containing debugging information and place them at the root of the resource folder. The files are named using the resource name as a prefix with .ttc and .tws as extensions.<br><br>Dump Network - Indication of whether to generate network and device level files containing debugging information. The files are placed at the root of the network folder and at the root of the device folder. The files placed in the network folder are named "NetworkConf" and have the extensions .ttc and .tws. The files placed in the device folder are named using the resource name as a prefix and have .ttc and .tws as extensions.<br><br>Dump POU Files - Indication of whether to generate resource level files containing debugging information and place these at root of the resource folder. Some of the files are named using the resource name as a prefix, the POU name as a suffix, and have the extensions .ttc and .tws. Other files are named using the POU name with .lst and .unc as extensions.<br><br>Enable Code Optimization - Indication of whether to optimize common expressions in a linear part of code and set the code generator to optimize the TIC code. Optimization performs many tasks: removes unused temporary variables, replaces each constant expression with its result, replaces repeated expressions and subexpressions with their equivalent values, suppresses unused and surplus target labels and null jumps, and simplifies arithmetic operations.<br><br>Function Internal State Enable - Indication of whether to produce internal state information for functions. Functions containing no internal state information denote that the invocation of a function with the same arguments always yields the same values. When set to *True*, local variables having the *var* direction are initialized using their initial values only at run-time startup. When set to *False*, function calls only initialize local variables, having the *var* direction, at every call.<br><br>Generate Map File - Indication of whether to generate resource level files containing debugging information. The files are placed at the root of the resource folder and are named using the resource name as a prefix with .ttc, .tws, and .map as extensions. |

| | |
|---|---|
| Embed Symbol Table | Indication of whether to embed, on the target, the symbol table specified as the type to embed |
| Embedded Table Type | The type of symbol table to download to the virtual machine with the resource code: None, Reduced, and Complete. The symbol table groups the variable names of the resource. The reduced symbol table contains only names of variables having a defined address. |
| Embedded Zip Source | Indication of whether to embed an exchange file (compressed 7-Zip format) holding all data from a project, device, or resource on the target. This exchange file is the same as the file created when exporting an element. |
| Structured C Source Code | Indication of whether structured C source code is produced by the compiler. Structured C source code can then be compiled and linked with libraries to produce embedded executable code. |
| TIC Code | Indication of whether Target Independent Code is produced by the compiler. TIC code can be executed on virtual machines. |

**Hardware**

| | |
|---|---|
| Target | The hardware platform on which Virtual Machines run resources of a project |

**Info**

| | |
|---|---|
| Comment | Text displayed next to the resource name in the Solution Explorer |
| Description | Free-form text describing a resource |
| Extended Parameters | OEM-defined parameters for resources enabling the customization instances of individual ISaVM tasks. These parameters are sent to virtual machines with the resource code. |
| Full Name | Full name of resource using the following syntax: *ProjectName.DeviceName.ResourceName* |
| Is Password Protected | Indication that the resource is protected by a password controlling its access |
| Memory Usage (Code) | Indication of the amount of memory used by the code for the programs of the resource (in bytes) |
| Memory Usage (Data) | Indication of the amount of memory used by the variables of the resource (in bytes) |

| | |
|---|---|
| Memory Usage (Retain Space) | Indication of the amount of memory used by the retain variables of the resource (in bytes) |
| Name | Name of the resource. Resource names can have up to 128 characters. |
| Number | Unique number identifying a resource within the project. This number is automatically assigned. When changing this number, you need to assign a number that is unique within the project. The resource number identifies the virtual machine that will run the resource code. |
| Path | Complete path where the resource files are stored on the computer. %USERPROFILE%\My Documents\ ISaGRAF 6.x\Projects*SolutionName**ProjectName**ProjectName*\ *DeviceName**ResourceName* |

**Memory Size for Online Changes**

| | |
|---|---|
| Code Size | For online changes, the amount of memory reserved for code sequence changes |
| Maximum Extra POUs | The maximum number of POUs that can be added during online changes |
| SFC States Mem Size | The memory space allocated for step and transition structures. A step requires 40 bytes and a transition requires 20 bytes. |
| User Variable Size | For online changes, the amount of memory reserved for adding variables data. When generating symbol monitoring information for a POU, the same amount of memory is also reserved for the POU. |

**Settings**

| | |
|---|---|
| Cycle Time | The amount of time given to each cycle. If a cycle is completed within the cycle timing period, the system waits until this period has elapsed before starting a new cycle. The cycle consists of scanning the physical inputs of the process to drive, executing the POUs of the resource, then updating physical outputs. The virtual machine executes the resource code according to the execution rules. |

| | |
|---|---|
| Cycle Time Units | Unit of measure for the cycle time. Possible values are ms (milliseconds) or µs (microseconds). To use µs, the target must support this unit of measure. |
| Detect Errors | Indication of whether to store errors. You need to define **Nb Stored Errors**. |
| Execution Mode | Indication of whether a resource executes in real time or cycle-to-cycle. RealTime mode is the run time normal execution mode where target cycles are triggered by the cycle timing. In cycle-to-cycle mode, the virtual machine loads the resource code but does not execute it until you execute one cycle or activate real-time mode. |
| Memory For Retain | Location where retained values are stored (the required syntax depends on the implementation) |
| Nb Stored Errors | Number of entries, i.e., the size of the queue (FIFO) in which detected errors are stored |
| Online Behavior | Behavior of the resource when switching to online or simulation mode. Possible options are the following:<br>- Design where the resource remains in design mode when switching to online or simulation mode<br>- Debug Only where the resource remains in design mode until switching to online mode; switching to simulation mode does not affect the resource<br>- Simulate Only where the resource remains in design mode until switching to simulation mode; switching to online mode does not affect the resource<br>- Always where the resource switches to either online or simulation mode |
| Trigger Cycles | Indication of whether a resource cycle executes according to the defined Cycle Time |

**SFC Dynamic Behavior Limits**

| | |
|---|---|
| Gain Factor | For SFC, specifies factor of dynamic behavior limits determining the amount of memory, allocated by a target at initialization time, designated to manage token moving. The amount of allocated memory is calculated as a linear relation with the number of SFC POUs:<br><br>Alloc Mem (bytes) = N * NbElmt * sizeof(typVa)<br><br>NbElmt = GainFactor * NbOfSFC + OffsetFactor<br><br>Where:<br><br>N = 5 (constant linked to SFC engine design)<br>NbElmt = The maximum number of transitions that can be valid for each executed cycle, i.e., transitions with at least one of their previous steps being active.<br>typVa = 16 bits in the medium memory model (32 bits in the large memory model)<br>GainFactor and OffsetFactor = the linear parameters of the linear relation<br>NbOfSFC = the number of SFC POUs in the project |
| Offset Factor | Same as Gain Factor |

For bindings, resources use the HSD network.

**To add a resource**

- In the Solution Explorer, right-click the device element, point to **Add**, and then click **New Resource**.

A resource is added to the device.

**To rename a resource**

- In the Solution Explorer, right-click the resource, click **Rename**, and then type a name for the resource.

**To delete a resource**

Deleting a resource also removes all programs, functions, function blocks, and variables defined for the resource.

- In the Solution Explorer, right-click the resource, and then click **Delete**.

## See Also

Debugging

## Programs

You define programs in the Programs section of a resource in the Solution Explorer. Within a Programs section, sequential programs must be adjacent. Programs belonging to a same section must have different names.

For programs, you need to specify the following properties:

**Code Generation**

| | |
|---|---|
| Generate Debug Info | Indication of whether to generate information required for debugging using step-by-step execution |
| Generate Monitoring Symbols | For graphical POUs, indication of whether to generate information required for graphically displaying the output values of elements when debugging or simulating |

**Info**

| | |
|---|---|
| Comment | Text displayed next to the program name in the Solution Explorer |
| Description | Free-form text describing a program |
| Full Name | Full name of program using the following syntax: *ProjectName*.*DeviceName.ResourceName.ProgramName* |
| Is Password Protected | Indication that the program is protected by a password controlling its access |
| Language | Programming language of the POU |
| Name | Name of the program. Program names must begin with a letter and can have up to 128 characters. |
| Order | Position of the program within the execution order |
| Path | Complete path where the program files are stored on the computer. %USERPROFILE%\My Documents\ ISaGRAF 6.x\Projects*SolutionName**ProjectName**ProjectName*\\ *DeviceName**ResourceName**ProgramName* |

**Settings**

| | |
|---|---|
| Interrupt Enabled | For targets supporting interrupts, you can configure interrupts to control the moment of execution of cyclic programs (ST, LD, FBD, and SAMA). Such programs are executed independently of the execution order applied to other programs. Interrupts can be called from code to execute a program. When enabled for a program, the program moves to the Interrupts section of the Solution Explorer. When enabling an interrupt for a program, you need to define interrupt parameters:<br>- Interrupt Data Type, the data type of the interrupt<br>- Interrupt Initial Value, the initial value of the interrupt<br>- Interrupt Selection, enables selecting from available interrupt definitions. |

**To add a program**

You define programs for a resource.

- In the Solution Explorer, right-click the program element for a resource, point to **Add**, and then click the required programming language.

**To rename a program**

- In the Solution Explorer, right-click the program, click **Rename**, and then type a name for the program.

**To configure an interrupt for a program**

1. In the Solution Explorer, select the program.
2. In the Properties window, set the Interrupt Enabled property to True.

   The program moves to the Interrupts section.
3. In the Interrupts section, select the program, then expand the Interrupt Parameters properties and configure the interrupt settings.

**To delete a program**

- In the Solution Explorer, right-click the program, then click **Delete**.

# Functions

You define functions in the Functions section of a resource in the Solution Explorer.

For functions, you can specify the following properties:

**Code Generation**

| | |
|---|---|
| Generate Debug Info | Indication of whether to generate information required for debugging using step-by-step execution. |

**Info**

| | |
|---|---|
| Comment | Text displayed next to the function name in the Solution Explorer |
| Description | Free-form text describing a function |
| Full Name | Full name of function using the following syntax: *ProjectName.DeviceName.ResourceName.FunctionName* |
| Is Password Protected | Indication that the function is protected by a password controlling its access |
| Language | Programming language of the POU |
| Name | Name of the function. Function names are limited to 128 characters beginning with a letter followed by letters, digits, and single underscore characters. These names cannot have two consecutive underscore characters. |
| Order | Position of the function within the execution order |
| Path | Complete path where the function files are stored on the computer: %USERPROFILES%\My Documents\ ISaGRAF 6.x\Projects*SolutionName**ProjectName**ProjectName*\\ *DeviceName**ResourceName**FunctionName* |

When adding functions, you also need to define parameters. Functions can have a maximum of 128 parameters (inputs and outputs). When defining parameters, consider the following limitations:

- Parameter names are limited to 128 characters and can begin with a letter followed by letters, digits, and single underscores

- Possible data types for parameters are BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, Array types, Structure types, Function blocks
- For String type variables, string capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string
- For user defined addresses, the format is hexadecimal and the value ranges from 1 to FFFF
- For dimensions, example: [1..10] for a one dimensional array, [1..4,1..7], for a two dimensional array

**To add a function**

1. In the Solution Explorer, right-click the Functions element, point to **Add**, and then click the required programming language for the function.
2. To define the parameters for the function, right-click the function and then click **Parameters**.

   The Block Selector displays the Parameters section where you define the parameters for the function.

**To rename a function**

- In the Solution Explorer, right-click the function, click **Rename**, and then type a name for the function.

**To delete a function**

- In the Solution Explorer, right-click the function, and then click **Delete**.

# Function Blocks

You define function blocks in the Function Blocks section of a resource in the Solution Explorer.

For function blocks, you can specify the following properties:

**Code Generation**

| | |
|---|---|
| Generate Debug Info | Indication of whether to generate information required for debugging using step-by-step execution. |
| Generate Monitoring Symbol | For graphical POUs, indication of whether to generate information required for graphically displaying the output values of elements when debugging or simulating |
| Instance Symbols Extra Bytes | Size of memory reserved for each function block instance for adding symbols monitoring information during online changes. Note that a string-type output takes up to 260 bytes. |

**Info**

| | |
|---|---|
| Comment | Text displayed next to the function block name in the Solution Explorer |
| Description | Free-form text describing a function block |
| Full Name | Full name of function block using the following syntax: *ProjectName.DeviceName.ResourceName.FunctionBlockName* |
| Is Password Protected | Indication that the function block is protected by a password controlling its access |
| Language | Programming language of the POU |
| Name | Name of the function block. Function block names are limited to 128 characters beginning with a letter followed by letters, digits, and single underscore characters. These names cannot have two consecutive underscore characters. |
| Order | Position of the function block within the execution order |
| Path | Complete path where the function block files are stored on the computer: %USERPROFILES%\My Documents\ISaGRAF 6.x\Projects*SolutionName**ProjectName**ProjectName**DeviceName**ResourceName**FunctionBlockName* |

**Settings**

| | |
|---|---|
| Tokens Limit | For SFC and basic IEC function blocks, the maximum number of tokens for a POU is equal to the number of parallel steps below a transition plus one. For example, when there are four parallel steps below a transition, the tokens limit must be set to a minimum of five. |

When adding function blocks, you also need to define parameters. Function blocks can have a maximum of 128 parameters (inputs and outputs). When defining parameters, consider the following limitations:

- Parameter names are limited to 128 characters and can begin with a letter followed by letters, digits, and single underscores
- Possible data types for parameters are BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, Array types, Structure types, Function blocks
- For String type variables, string capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string
- For user defined addresses, the format is hexadecimal and the value ranges from 1 to FFFF
- For dimensions, example: [1..10] for a one dimensional array, [1..4,1..7], for a two dimensional array

For instances of function blocks, you can reset the initial values defined for individual instances.

**To add a function block**

1. In the Solution Explorer, right-click the Function Blocks element, point to **Add**, and then click the required programming language for the function.

2. To define the parameters for the function block, right-click the function block, and then click **Parameters**.

   The Block Selector displays the Parameters section where you define the parameters for the function block.

**To reset the initial values of function block instances**

1. From the Solution Explorer, right-click the function block, point to **Refactor**, and then click **Reset Initial Values of Instances**.

2. In the Refactoring dialog box, select the required instances of the function block for which to reset the initial values, and then click **OK**.

**To rename a function block**

- In the Solution Explorer, right-click the function block, click **Rename**, and then type a name for the function block.

**To delete a function block**

- In the Solution Explorer, right-click the function block, and then click **Delete**.

# Variables

Variables are defined for their scope. For instance, global variables are available for use throughout the programs, functions, and functions blocks of a resource. Whereas, variables defined for a program, a function, or a function block are local to that element. You define variables in the Variables grid. You can create groups to which you add existing variables. Variables can belong to multiple groups. For individual variable scopes, you can import and export variables data having the Microsoft Excel (*.xls) format.

When defining variables data using a spreadsheet you enter each piece of information in a separate cell, leave cells empty if items are to be omitted, and save the file in XLS format. These requirements are automatically followed by the export utility; you must respect these when building a file to be imported.

When defining complex variables such as arrays and structures, the syntax for the variable name is as follows:

- For arrays: arrayname[index]

  ```
  Name,Alias,Data Type,StringSize,InitValue,Direction,Wiring,Attribute ...
  array1,,BOOL,0,,, ...
  "array1[1,1]",,BOOL,0,,, ...
  "array1[1,2]",,BOOL,0,,, ...
  "array1[1,3]",,BOOL,0,,, ...
  "array1[1,4]",,BOOL,0,,, ...
  "array1[1,5]",,BOOL,0,,, ...
  ```

- For structures: structurename.membername

  ```
  Name,Data Type,Dimension,Alias,Comment,InitValue,Direction ...
  structure1,,T9K_DI_FULL,0,, ...
  structure1.DI,,BOOL,0,, ...
  structure1.LF,,BOOL,0,, ...
  structure1.DIS,,BOOL,0,, ...
  structure1.CF,,BOOL,0,, ...
  structure1.V,,UINT,0,, ...
  structure1.STA,,USINT,0,, ...
  ```

When managing variables data, you can perform the following tasks:

- Import and export variables data
- Creating groups for variables data

**To create a variable group**

When adding variables to a group, you can add these to the group from the variables grid or you can drag these between the variables grid and the group grid.

**1.** In the Solution Explorer, right-click Variable Groups item, and then click **Add New Variable Group**.

   The group is added.

**2.** To add variables to the group using the contextual menu options:

   - Open the variables grid, select one or more variables, then right-click the selection, point to **Add to Group**, and then click the required group name.

**3.** To add variables to the group by dragging:

   **a)** Open the variables grid and the group grid by double-clicking and place both grids side-by-side.

   **b)** In the variables grid, select the consecutive variables, then drag the selected variables from the cell having the arrow in the left most column to within the variables group grid.

# Choosing Project Templates for Targets

When creating projects, you select a project template depending on the operating system, target type, and features required to develop your application. Each template has different features and is designed for use with a corresponding target type. The following describes the templates, their compatible targets, and available features.

| Template Name, Target Name | Description |
|---|---|
| Import ISaGRAF 5 Project | Enables importing a multi-resource **ISaGRAF 4** or **ISaGRAF 5** project into the **ISaGRAF 6** workbench. |
| Import ISaGRAF Zip Project | Enables importing a compressed **ISaGRAF 6** multi-resource project into the **ISaGRAF 6** workbench. |
| ISaFREE_TPL,<br>ISAFREE-TGT | Enables creating a single-resource project for use with the **ISaGRAF** Free Windows target. Projects can have a maximum size of 3200 bytes.<br><br>Features: C functions and function blocks, enhanced target features (microsecond cycle timing, motion control and safety function blocks), password protection, TIC code optimization, online changes, bindings, retain values, interrupts, flexible arrays and function block parameters by reference, multiple network instances of the same type, set priority for SFC transitions, wiring for complex variable members. |
| Library | Enables creating a library project starting with one resource in one device. |

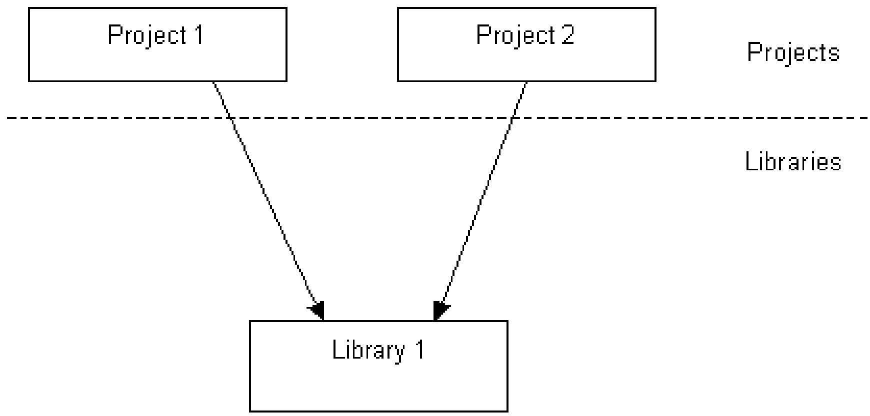
| | |
|---|---|
| PRJ61499_TPL, SIMULATOR | Enables creating a project using the IEC 61499 distributed method.<br><br>Features: Enhanced target features (microsecond cycle timing, motion control and safety function blocks), password protection, TIC code optimization, bindings, online changes, retain values, microsecond cycle timing, flexible arrays and function blocks passed by reference, POUs of 64 KB and greater, multiple network instances of the same type, setting of SFC transition priority, and wiring on complex variable members. |
| Simulator, SIMULATOR | Enables creating a project starting with one resource in one device for use with the Simulator target.<br><br>Features: Enhanced target features (microsecond cycle timing, motion control and safety function blocks), password protection, TIC code optimization, bindings, online changes, retain values, microsecond cycle timing, flexible arrays and function block parameters by reference, POUs of 64 KB and greater, multiple network instances of the same type, setting of SFC transition priority, and wiring on complex variable members. |

**See Also**

Creating a Project

# Creating a Library

Libraries are special projects made up of devices and resources in which you define functions, function blocks, global variables, arrays and structures for reuse throughout **ISaGRAF** projects. Libraries also enable you to modularize projects and to isolate functions and function blocks so that these can be validated separately.




A project can depend on more than one library and different projects can call the same library. When creating a library, it can contain functions, function blocks, defined words, arrays and structures. These library elements can be called from a project once the library is added as a dependency. Functions and function blocks can be written using the IEC 61131-3 languages (FBD, LD, SAMA, SFC, or ST).

You create libraries as part of a solution in the Automation Collaborative Framework. A solution can hold multiple projects and libraries.

You base a library on a Library template then develop its elements, i.e., devices, resources, programs, functions, and function blocks. Libraries are stored in the same location as projects and are also MS-Access database (.MDB) files.

The target type of a library resource affects the usability of functions and function blocks throughout projects using the library. A library can only have one device target type. Functions and function blocks can only be used in resources referring to the same target type, except when they use the SIMULATOR target type. When library resources use the SIMULATOR target type, all of their functions and function blocks can be used in any project resource regardless of its target type.

Library functions and function blocks must have unique names. When they have the same names as those defined in a project in which they are used, only those from the project are recognized. Furthermore, you do not need to compile functions and function blocks in the library before using them in projects. These are compiled in the calling project space, in order to take care of the compiling options defined for the project.

**To create a library**

1. From the File menu, point to **New**, and then click **Project**.
2. In the Project Types list, expand the *CAM Projects* option, then click **ISaGRAF 5**.
3. From the list of available project templates, click the Library template.
4. Specify a name and location for the library, indicate whether to add the library to an existing solution or create a new solution by defining a solution name, then click **OK**. For new solutions, you can choose to create a directory.

### See Also

Using a Library in a Project
Creating a Project

# Using a Library in a Project

Projects can use functions and function blocks from one or more libraries. You need to create libraries before using them. Furthermore, you need to define a project's dependencies, i.e., the set of libraries the project will use, before using a library's defined elements. A project can depend on more than one library.

Library functions and function blocks can refer to some global defined words or data types defined in the library. In such a case, these defined words and data types from the library can also be used in the project.

A library cannot use functions and function blocks from another library. In other words, you cannot define external dependencies for a library. However, a function or function block from a library can call other functions or function blocks from the same library. Furthermore, functions or function blocks from libraries can call 'C' written functions and function blocks defined for the corresponding target.

All functions and function blocks within a project, including those coming from libraries, must have unique names. When more than one uses the same name, the following conditions apply:

- If the functions or function blocks come from different libraries, warnings are generated at compilation and only the first definition is recognized.
- If one function or function block is defined in the project and the other from a library, only the one defined in the project is recognized. The other is ignored.

Furthermore, when the same name is used for several types or several defined words having different definitions in a project and attached libraries, an error is generated at compilation time. However, when a data type or defined word is defined several times with the same contents or definition, a warning is reported but the project can be compiled.

You add dependencies onto libraries from the Dependencies dialog box. In this dialog box, the Libraries list displays the libraries on which a project has dependencies while the Solution list displays all libraries contained in the solution.

**Note:** When redefining the location of a library dependency you can modify the path in the library properties; removing the library will result in a loss of all project references.

**To use a library in a project**

1. Right-click the project for which to add a dependency, point to Add, and then click **Add Dependency**.

2. In the Dependencies dialog box, click Browse to locate the library on which to create the dependency.

The library is displayed in the Libraries list.

### See Also

Creating a Library

# Setting Project Access Control

For project security, you can set access control using a password for projects, resources, devices, POUs, and library functions and function blocks. Password definitions are limited to eight characters and can consist of letters, digits, and symbols. When projects are password-protected they cannot be opened for editing. Project sub-elements, can have their own level of access control. For example, a POU having its own password remains locked and cannot be modified without entering its password.

**Note:** Since POUs are encrypted, you need to retain password definitions.

In the Solution Explorer, the following indicate the security state for elements:

 Indicates that a lock is applied to the element

When opening a project having password-protected elements, you are only prompted to enter the password once for each element. Password-protected elements have the following modification restrictions:

| Password-Protected Element | Modification Restrictions |
|---|---|
| Project | Opening the project |
| Device | Adding, editing, and deleting a resource, program, library function, or library function block |
| Resource | Adding, editing, and deleting a program, library function, or library function block |
| Program | Viewing the program |
| Library Function | Viewing the function |
| Library Function Block | Viewing the function block |

You can edit existing passwords for projects and project sub-elements. You can also remove existing passwords. When copying, pasting, importing, and exporting elements having access control, password definitions are retained.

**To set a password**

1. In the Solution Explorer, right-click the required element, and then click **Password**.

**2.** In the *Set Password* dialog box, enter the required information, then click OK.

**a)** In the *Password* field, type the required password.

**b)** In the *Confirm Password* field, re-type the required password.

**To edit a password**

**1.** In the Solution Explorer, right-click the required element, and then click **Password**.

**2.** In the *Set Password* dialog box, enter the required information, then click **OK**.

**a)** In the *Old Password* field, type the current password.

**b)** In the *Password* field, type the required password.

**c)** In the *Confirm Password* field, re-type the required password.

**To remove a password**

**1.** In the Solution Explorer, right-click the required element, and then click **Password**.

**2.** In the *Set Password* dialog box, enter the required information, then click **OK**.

- In the *Old Password* field, type the current password.
- The *Password* and *Confirm Password* fields must remain blank.

### See Also

Setting Target Access Control

# Setting Target Access Control

For device target security, you can set access control by defining a password for the device target. Password definitions are limited to eight characters and can consist of letters, digits, and symbols. Target access control prevents the connection of all IXL clients not having the password for the target. Users having the password can attach the target to devices in different projects.

**Note:** The password definitions for device targets are saved on target platforms.

You can edit existing passwords for device targets. You can also remove existing passwords for device targets. When setting, editing, and deleting the password for a device target, the attached target must be running.

**To set a target password**

1. In the Solution Explorer, right-click the device element, and then click **Target Password**.
2. In the *Set Password* dialog box, enter the required information, then click **OK**.
    - **a)** In the *Password* field, type the required password.
    - **b)** In the *Confirm Password* field, re-type the required password.

**To edit a target password**

1. In the Solution Explorer, right-click the required device element, and then click **Target Password**.
2. In the *Set Password* dialog box, enter the required information, then click **OK**.
    - **a)** In the *Old Password* field, type the current password.
    - **b)** In the *Password* field, type the required password.
    - **c)** In the *Confirm Password* field, re-type the required password.

**To remove a target password**

1. In the Solution Explorer, right-click the required device element, and then click **Target Password**.

2. In the *Set Password* dialog box, enter the required information, then click **OK**.
   - In the *Old Password* field, type the current password.
   - The *Password* and *Confirm Password* fields must remain blank.

### See Also

Setting Project Access Control

# Importing Target Definitions

You can import target definitions into a project. These target definitions are *.tdb files.

**To import a target definition file into a project**

1. From the Solution Explorer, right-click the project and point to **Import**, and then click **Import Target Definitions**.

2. In the Open window, browse to locate the target definitions (*.tdb) file to import into the project, then click **Open**.

When the importation process is completed, the features from the target definition are available for use in the project.

# Importing and Exporting Elements

You can import elements, i.e., projects, devices, resources, and POUs, having been previously exported. Exporting an element creates a copy in XML format of the element definitions, including sub-elements, and stores this information in a compressed 7-Zip (.7z) exchange file. You can import elements into devices, resources, and programs in the same project or in other projects.

When importing elements, you can select individual sub-elements to import or choose to import all sub-elements. **ISaGRAF** places imported elements at the proper location within a project. For example, when importing a resource element into a POU, the resource is added to the device containing the POU. The Output window details the progress of import operations.

When exporting definitions for elements, the resulting exchange file contains all sub-element definitions as well as global and local variables. You can also choose to export only the variables for certain elements. For devices and resources, you can export global variables. For POUs, you can export local variables.

You specify the location in which to save exchange files. You can also choose to set a password for an exported exchange file. When importing and exporting elements having access control, password definitions are retained.

**To import elements**

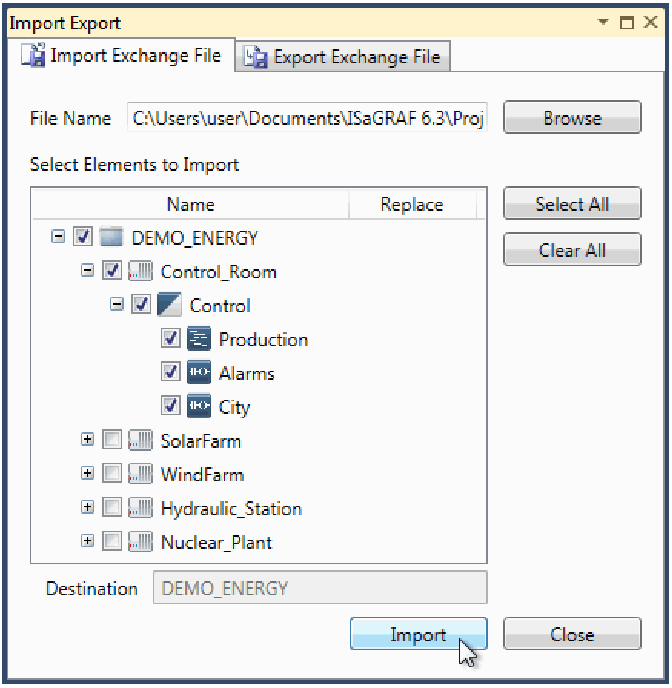
You can only import elements having been previously exported and stored as compressed exchange files.

1. In the Solution Explorer, right-click the destination element for the exchange file, point to **Import**, and then click **Import Exchange File**.

2. In the Import Export dialog box, on the *Import Exchange File* tab, browse to select the exchange file to import.

   - In the **Select Import Exchange File** dialog box, select the exchange file to import, and then click **Open**.

3. From the *Select Elements to Import* display, select the elements to import, and then click **Import**.




Using the *Select All* option, you can select all the elements displayed. The *Clear All* option enable you to deselect all elements, then reselect only those required.

**4.** When the imported element name exists at the destination, you need to choose one of the following actions to resolve the conflict.

- Skip imported element and use existing one instead.
- Create a new copy of the element from the imported one.
- Replace existing element with the imported one.

**5.** When the import process is complete, in the Import Export dialog box, click .

The imported elements are available for use.

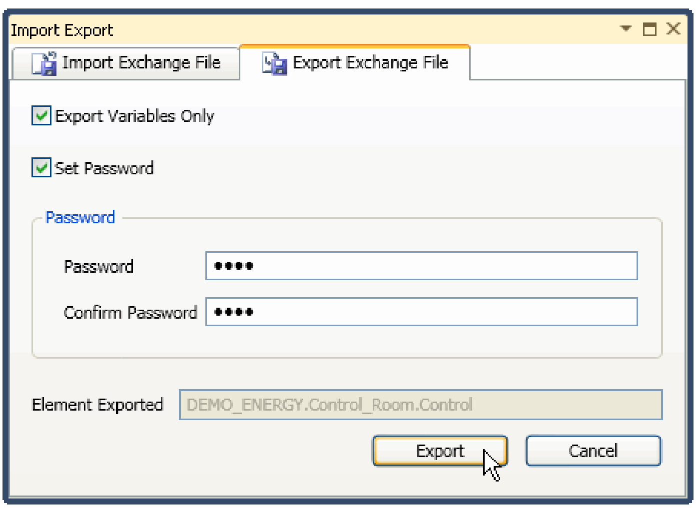
**To export elements**

When exporting elements, these are saved as exchange files having a 7-Zip (.7z) compressed format.

1. In the Solution Explorer, right-click the element to export, point to **Export**, and then click **Export *Item***.

2. In the Import Export dialog box, specify the options for the exchange file (optional), then click **Export**.

   - To export only the variables associated with the element, select the *Export Variables Only* option.
   - To set a password for the exported exchange file, select the *Set Password* option, then define and confirm the password by typing in the fields provided.




3. On the Save As dialog box, specify a name and location in which to save the exported file, then click **Save**.

4. When the export process is complete, in the Import Export dialog box, click .

The exchange file containing the exported element is placed at the specified location.

# Importing and Exporting Variables Data

You can import variables that were previously exported and saved as Microsoft Excel spreadsheets (.xls). Exporting variables enables management of variables data in Excel, including adding, removing, and modifying variables. You can import previously exported Excel files into other resources and programs in the same project or in other projects.

When importing variables, you import the fields selected during the export process. For previously exported Excel files containing modified content, any additional columns of data using proper syntax will be imported. The Output window details the progress of import operations, including the names and location of the variables added.

When exporting variables, you can select the fields of the variables to export. You also specify the location in which to save the exported files.

You can also import files containing manually defined variables for use in resources and programs. When importing files created manually, you must include a header row containing the same syntax used in files exported from **ISaGRAF**. The Excel file syntax uses the internal names for the columns of data instead of those displayed in the Variable Export/Import dialog box. Any rows of data using improper syntax will not be imported.

The following table displays the syntax used in Excel files and the associated dictionary properties:

| File Column | Dictionary Property | Description |
|---|---|---|
| Name | Name | Name of the variable |
| Data Type | Data Type | Data type of the variable |
| Dimension | Dimension | The number of elements defined for an array |
| String Size | String Size | The maximum character length for string-type variables |
| Initial Value | Initial Value | The value held by a variable when the virtual machine begins executing the resource. The format is comma separated values (CSV). |

| File Column | Dictionary Property | Description |
|---|---|---|
| Direction | Direction | For I/O wiring, indicates whether a variable is an input, output, or internal. |
| Attribute | Attribute | Indicates the read and write access rights |
| Retained | Retained | Indicates whether the value of the variable is saved by the virtual machine at each cycle |
| Comment | Comment | User-defined free-format text for variables |
| Alias | Alias | Any name |
| Wiring | Wiring | Indicates the I/O channel wired to the variable |
| Address | Address | User-defined address of the variable |
| Retained Flags | Retained | Enables retaining specific elements of a variable and indicates whether to use the initial value of a variable or the value previously retained on the target. The format is comma separated values (CSV). |
| Groups | Groups | Variable group containing the variables listed in alphabetical order |
| Comment Fields | Comment | User-defined free-format text for array elements. Each array element of the same type can have a different comment. The format is comma separated values (CSV). |

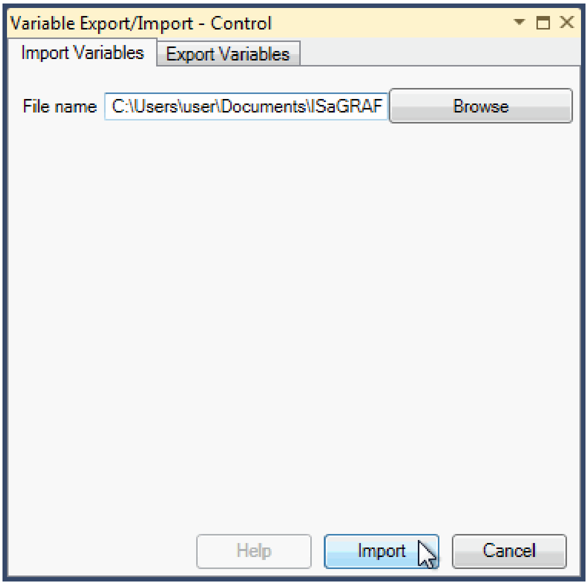
**To import variables**

You can only import variables having been previously exported and stored as Excel (.xls) files.

1. In the Solution Explorer, right-click the destination element for the Excel file, point to Import, then click **Variables from Excel...**..

2. In the Variable Export/Import dialog box, on the *Import Variables* tab, click browse to select the Excel file to import.

- In the **Import/Export File** dialog box, select the Excel file to import, then click **Open**.




3. In the Variable Export/Import dialog box, click **Import.**

4. When the import process is complete, click  .

The imported variables are available for use.

**To export variables**

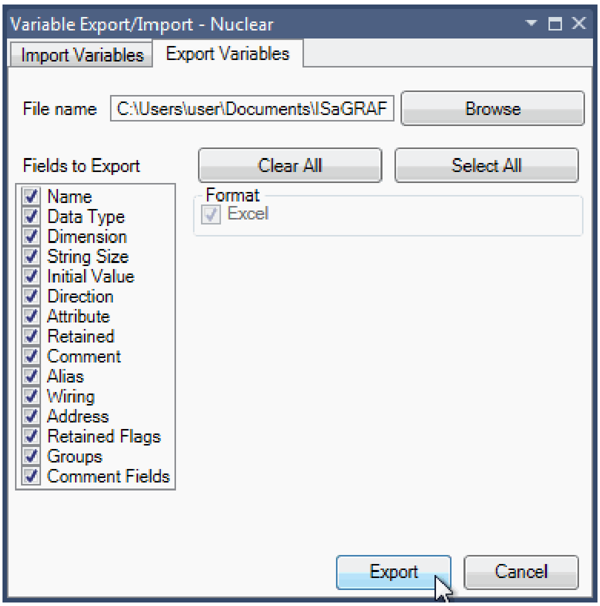
You can export selected fields of variables data in Excel (.xls) format.

1. In the Solution Explorer, right-click the resource or POU containing the variables to export, point to Export, then click **Variables to Excel...**.

2. In the Variable Export/Import dialog box, on the *Export Variables* tab, browse to select the destination for the exported variables.

3. In the **Import/Export File** dialog box, specify the name of the Excel file, and then click **Save**.

4. From the *Fields to Export* check box list, select the variables data to export, then click **Export**.




Using the *Select All* option, you can select all the fields displayed. The *Clear All* option enable you to deselect all fields, then reselect only those required.

5. When the export process is complete, click .

The variables are exported to the specified file.

# Generating Code

Before downloading code onto your target systems, you need to build the code for the whole solution. This operation builds the code for all projects within the solution, and builds information used to recognize your systems on networks. When a solution contains more than one project, you can build the code for individual projects within the solution. Once a solution or project has been built, subsequent build operations only regenerate the parts of the solution or project needing regeneration. You can also choose to build project elements, including devices, resources, and POUs. When building POUs, **ISaGRAF** only verifies the programming syntax without producing code.

When managing code, you can perform the following tasks:

- Building Solutions and Project Elements
- Rebuilding Solutions
- Cleaning Solutions and Project Elements

# Building Solutions and Project Elements

You can choose to compile project files that were modified since the last build. You can build modified project files belonging to entire solutions. Once a project has been built, subsequent builds only recompile the parts of the project needing recompiling.

When a solution contains more than one project, you can build the modified project files for individual projects. You can also choose to build individual project elements including devices, resources, and POUs.

You can rebuild solutions to ensure that the compiled version is up-to-date. When rebuilding solutions, intermediate and output files are deleted, then a build operation is performed. Deleting the intermediate and output files ensures that the entire solution is compiled during a rebuild operation. After rebuilding solutions, online changes become unavailable.

The compiler generates different code for simulation than for targets. Therefore, you need to specify the applicable target in the properties of devices before building.

When building solutions and project elements, you can view the progress of the build in the Output window. When the build is complete, you can view generated errors in the Error List.

**To build a solution or project element**

This operation builds the code for all resources of the projects and builds information used to recognize your systems on networks. You cannot build projects open in read-only mode. Before building a project, make sure the applicable target type is specified for the devices.

- In the Solution Explorer, right-click the required solution or project element, then click **Build** (or press **Ctrl+Shift+B**).

The build process is initiated for the required project element or solution.

**To view the build progress and generated errors**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Projects**, click **General**, then select the following options, and then click **OK**.

- Always show Error List if build finishes with errors
- Show Output window when build starts

**3.** Build the required solution or project element.

The Output and Error List windows are displayed.

**See Also**
Downloading Code to Targets
Rebuilding Solutions
Cleaning Solutions and Project Elements

# Rebuilding Solutions

You can choose to clean solutions, deleting the intermediate and output files, then rebuild all project files and components. After rebuilding solutions, online changes become unavailable.

You can view the progress of rebuild operations in the Output window. When the rebuild is complete, you can view generated errors in the Error List.

**To rebuild a solution**

1. In the Solution Explorer, click the solution element.
2. From the Build menu, click **Rebuild Solution**.

The rebuild process is initiated for the solution.

**To view the rebuild progress and generated errors**

1. From the Tools menu, click **Options**.
2. In the Options dialog box, expand **Projects**, click **General**, then select the following options, and then click **OK**.
   - Always show Error List if build finishes with errors
   - Show Output window when build starts
3. Build the required solution.

The Output and Error List windows are displayed.

### See Also

Downloading Code to Targets
Cleaning Solutions and Project Elements

# Cleaning Solutions and Project Elements

You can clean solutions, projects, devices, and resources. Cleaning these deletes the intermediate and output files generated during the last build operation. Performing cleaning operations removes the capacity to perform online changes for the selected element. For example, after cleaning a device, online changes become unavailable for that equipment.

**To clean a solution**

- In the Solution Explorer, right-click the solution, then click **Clean Solution**.

The intermediate and output files are deleted for the solution.

**To clean a project, device, or resource**

- In the Solution Explorer, right-click the required project, device, or resource, then click **Clean Selection**.

The intermediate and output files are deleted for the project element.

### See Also

Building Solutions and Project Elements
Rebuilding Solutions

# Running an Application Online

Running online signifies that an application is connected to a target allowing for the normal execution where target cycles are triggered by the cycle timing. While running online, you can perform target management, debugging, and monitoring operations. However, you cannot perform target management and debugging operations at the same time. You can also simulate the running of an application for debugging purposes.

Before running an application on a target, you need to build the project code and download the application code onto the target.

**To run an application online**

1. Specify the applicable target type and IP addresses for the devices in the project.

**Note:** The compiler generates different code for simulation than for targets.

2. Build the project code.
3. To run an application online, download the application code onto the target.
4. In the Debug toolbar, from the drop-down combo-box, select **Online**.
5. In Debug menu, click **Start Debugging**.

### See Also

Simulating
Debugging
Target Management
Monitoring

## Target Management

Target management operations affecting target behavior include downloading resource code to targets, uploading code from targets, stopping and starting resources as well as performing online changes.

**See Also**

Debugging

## Downloading Code to Targets

You perform download operations for projects having resources with code to send to targets. When simulating a project, you do not need to perform a download operation. When performing download operations, you can also choose to send custom files to the target except when using failover mechanisms. Such files are placed at the root of the target folder on the target platform.

Each time you perform a download operation, the **Automation Collaborative Platform** verifies the coherency between the current resource definitions and the resources' code to download. The Workbench also verifies the coherency between all versions of the resource code.

The code (corresponding to the run-time engine capabilities) must first be generated by building the project. The code type is determined by the target definition.

The Configuration manager must be running on the target platform.

The computer where the **Automation Collaborative Platform** is installed must be connected to the hardware device through a network supported by the Debugger. The standard networks used by the **Automation Collaborative Platform** are Ethernet (ETCP) and Serial COM port (ISaRSI).

You can choose to store resource, device, and project code on targets. When setting up a resource's properties, you select the required element to download to the target from the EmbeddedZipSource options.

**To download project code to a target**

1. To store code on the target, from the Solution Explorer, select the required resource, then from the Properties window, perform one of the following:

- To store code for the resource, from the *EmbeddedZipSource* drop-down combo-box, select **Resource**
- To store code for the device, from the *EmbeddedZipSource* drop-down combo-box, select **Device**.
- To store code for the project, from the *EmbeddedZipSource* drop-down combo-box, select **Project**.

**2.** To send custom files to the target, place the required files in the *To Download* folder located in the device directory.

**Note:** Custom files are placed at the root of the target folder on the target platform. When using a failover mechanism, you cannot send custom files to a target.

**3.** Build the project code.

**4.** In the Solution Explorer, right-click the project element, and then click **Download**.

## Uploading Code from Targets

You can upload code for projects, devices, and resources when the code has been stored on the target (if non-volatile storage exists for the platform). When uploading, a copy of the element is added to the Solution Explorer.

Before uploading an element's source file, you need to download its source code onto the target. When setting up the resource's properties, you select the required element to download to the target in the EmbeddedZipSource option.

**To upload an element from sources on a target**

- From the Solution Explorer, right-click the resource, device, or project for which to upload source code, and then click **Upload**.

A copy of the element is added to the Solution Explorer.

### See Also

Downloading Code to Targets

## Stopping and Starting Resources

You can stop and start a virtual machine. Stopping a resource terminates the virtual machine, changing the resource state to CODE. The resource state appears next to the resource icon in the Solution Explorer. Starting a resource launches the kernel process, producing the same result as downloading a resource. Once the resource is running online, you can apply the necessary execution mode, affecting the resource state.

**Note:** The STOP resource state indicates the cycle-to-cycle execution mode.

**To stop a resource running on a target**

1. In the Solution Explorer, select the resource to stop.
2. From the Target Execution toolbar, click .

**To start a resource on a target**

1. In the Solution Explorer, select the resource to start.
2. From the Target Execution toolbar, click .

**See Also**

Downloading Code to Targets
Debugging

## Performing Online Changes

You can modify a resource while it runs. This is sometimes necessary for chemical processes where any interruption may jeopardize production or safety. When performing online changes, you can choose to update a running resource at the time of download or at a later time. Note that sending custom files located in the *To Download* folder to a target is only available when performing a download operation; sending custom files is not possible when performing online changes.

**Warning:** Online changes should be used with care. **ISaGRAF** may not detect all possible conflicts generated by user-defined operations as a result of these online changes.

The initial values of variables are applied upon starting resources. Online changes do not start resources.

For all **ISaGRAF** versions, the following limitations exist for online changes:

Declared, i.e., user-defined, arrays and structures cannot be modified. Declared arrays and structures are defined as data types.

I/O device instances cannot be added or deleted; these instances can be modified.

Device and resource properties cannot be modified.

Undeclared arrays cannot be added or deleted. Undeclared arrays are defined as variables in a dictionary instance.

The following tasks are available for various **ISaGRAF** versions when performing online changes:

| | | |
|---|---|---|
| **ISaGRAF 4.X** targets or later | Internal Variables | Adding, deleting, and relocating internal variables.<br>Modifying the body of POUs. |

| | | |
|---|---|---|
| **ISaGRAF 5.10** targets or later | Bindings | To enable online changes for legacy bindings, set the *LegacyBindingDefault* property to 0 in the diamond.ini file. This file is installed at the following location: %ALLUSERSPROFILE%\ISaGRAF\6.x\CAM ISaGRAF 5\5.3\Bin<br><br>Adding, deleting, and editing.<br><br>Creating and deleting bindings between variables.<br><br>Changing the consumer error variable and consumption behavior of a binding. Changing the producing variable, consuming variable, or network for a binding creates a new one.<br><br>Adjusting the update timeout period in the resource network parameters. The update timeout period is the maximum time during which the consumer can remain in the update state. |
| | I/O Variables | Wiring, unwiring, and swapping I/O variables whose data type (scalar type for arrays), length (string variables), dimension (arrays), and address remains unchanged. For these I/O variables, you can modify the direction (input or output only), scope, attribute (read, write, or free), retain flag, alias, and comment. When modifying the direction, I/O variables cannot change to or from the internal type. Note that modifying the I/O wiring causes the values of new and removed output I/O variables to be reinitialized. |
| | I/O Channels | Changing the wired variable as well as the reverse/direct, gain, offset, and conversion settings. |

| | | |
|---|---|---|
| **ISaGRAF 5.23** targets or later | Internal Variables | When renaming or changing the data type of internal variables, the Workbench creates new variables. Therefore, variables are initialized.<br><br>Changing the alias, initial value, group, scope, direction, retain setting, address, and comment of variables. When changing the initial value of a read-only internal variable, the Workbench reinitializes the variable. When changing the scope of a variable, the Workbench reinitializes the variable.<br><br>Modifying the length of string variables. When decreasing the length, the contents of the string is truncated to the new length.<br><br>Switching a variable attribute between the input and output attribute. You cannot switch variables between the internal and input/output attribute.<br><br>Adding and removing elements in arrays for internal variables. For multi-dimensional arrays, you can only add elements to the first dimension. The Workbench initializes these new elements. Adding elements to other dimensions causes the Workbench to initialize a new array. |
| | Programs | Adding, deleting, renaming, and reordering (for execution within the programs section) programs. When renaming programs, the Workbench detects a CRC mismatch and updates the code on the target for the program and reinitializes all local variables. When renaming SFC programs, instance data and local variables are not preserved, i.e, elements are reset to their initial state.<br><br>When planning to add programs (other than SFC) using online changes, you need to allocate a sufficient number of maximum extra POUs.<br><br>When planning to add SFC programs using online changes, you need to allocate sufficient memory space for SFC programs.<br><br>Adding, deleting, renaming steps and transitions as well as modifying the initial step or the flow between elements. When modifying SFC programs, instance data and local variables is preserved, i.e., elements are not reset to their initial state. |

| | | |
|---|---|---|
| **ISaGRAF 5.50** targets or later | Functions and Function Blocks | Adding and deleting "C" function block instances having initialization or exit functionality implemented. |

**To perform an online change**

You can perform online changes after building a project. Online changes are unavailable after cleaning projects, cleaning solutions, and rebuilding solutions.

- From the Solution Explorer, right-click the project for which to perform the online change, then click **Online Change**.

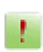
# Debugging

When developing an application, you can choose to debug, i.e., detect and remove errors, from a project while running the application online, i.e., on a target, or simulating. Before running an application online, you need to download the application code onto the target.

While in real-time mode, each resource is executed by a virtual machine on the real platform. A download operation is required to download the code of each resource onto the corresponding platform. You can also switch a resource to cycle-to-cycle mode.

A resource where real-time mode is activated is in the RUN state.

When debugging, the state of a resource is displayed in its icon in the Solution Explorer. The possible states of a resource are the following:

| | |
|---|---|
|  | The resource is running on the device. The resource is in the RUN, STOP, ERROR, STEPPING, or STEPPING_ERROR state. |
|  | The application running on the virtual machine does not match the project. |
|  | The virtual machine is unable to establish communication with the target run-time or unable to locate the code. The resource is in the CODE or NOCODE state. |

To enable debugging a project, you must first build the project, then download the project code to the target.

When switching an application to debugging, the **Automation Collaborative Platform** verifies the coherency between the current resource definitions and the resources' compiled code. The **Automation Collaborative Platform** also verifies the coherency between all versions of the resource code.

You can execute a resource in one of two execution modes:

- Real-time, the run time normal execution mode where target cycles are triggered by the programmed cycle timing. While in real-time mode, you can switch the resource to cycle-to-cycle mode. When debug information is generated for POUs in a resource, the resource automatically switches to step-by-step mode when the application encounters a breakpoint.

- Cycle-to-cycle, a cyclical execution mode where the virtual machine loads the resource code but does not execute it until you execute one cycle or activate real-time mode. When debug information is generated for POUs in a resource, the resource automatically switches to step-by-step mode when the application encounters a breakpoint. You can also switch to step-by-step mode by stepping.

The state of the resource appears next to the resource icon in the Solution Explorer.

| Resource State | Description |
|---|---|
| RUN | The resource is running in real-time mode. You can switch the resource to cycle-to-cycle mode. |
| STOP | The resource is in cycle-to-cycle mode.<br>Possible operations are:<br>- switch the resource to real-time mode<br>- execute one cycle<br>- step into or step over the next line of code (when step-by-step mode is instantiated) |
| ERROR | The resource is in error.<br>Possible operations are:<br>- switch the resource to real-time mode<br>- execute one cycle<br>- step into or step over the next line of code (when step-by-step mode is instantiated) |
| STEPPING | The resource is in step-by-step mode.<br>Possible operations are:<br>- switch the resource to real-time mode<br>- switch the resource to cycle-to-cycle mode returning the resource to the start of its cycle without executing the remaining code<br>- execute one cycle<br>- step into or step over the next line of code (when step-by-step mode is instantiated) |
| STEPPING_ERROR | The resource is in stepping error mode. This state is caused when an invalid operation occurs such as a division by 0 or a bound check error. You can switch the resource to cycle-to-cycle mode returning the resource to the start of its cycle without executing the remaining code. |

| Resource State | Description |
| --- | --- |
| CODE | The virtual machine is unable to execute the resource. Verify that the virtual machine matches the target definition in the Workbench. |
| NOCODE | The virtual machine is unable to locate the application code. |

When running online, a resource is activated in the RUN state. A resource where cycle-to-cycle mode is activated can be in one of three states: STOP, BREAK, and ERROR. When viewing the values of variables in dictionary instances, the logical and physical values display the following temporary messages before loading the actual values:

- OFFLINE, indication that the variable is not present in the running application code
- WAIT, indication that the variable is either:
  - In online mode and attempting to connect to the target
  - In simulation mode and attempting to connect to the simulator

While debugging, you can lock and unlock I/O channels of an I/O device.

**To debug an application**

Before debugging an application, you need to build the application code and download the code to the target.

1. Build the project code.
2. Download the code to the target.
3. In the Debug toolbar, from the drop-down combo-box, select **Online.**
4. From the Debug menu, click **Start Debugging** (or press **F5**).

## See Also

Resources
Forcing the Values of Variables

## Accessing Diagnostic Information (System Variables)

You can access diagnostic information for individual resources while running an application in simulation mode.

System variables hold the values of resource variables relating to cycle count, timing, kernel bindings, and resource information. You can view system variables from the dictionary instances for resources. You can read from and write to system variables. The available system variables are the following:

| Variable Name | Type | Read/Write | Description |
|---|---|---|---|
| __SYSVA_CYCLECNT | DINT | Read | Cycle counter |
| __SYSVA_CYCLEDATE | TIME | Read | Timestamp of the beginning of the cycle in milliseconds |
| __SYSVA_KVBPERR | BOOL | Read/Write | Kernel variable binding producing error (production error)<br>This system variable is not available for use. |
| __SYSVA_KVBCERR | BOOL | Read/Write | Kernel variable binding consuming error (consumption error) |
| __SYSVA_MICROCYCLEDATE | UDINT | Read | Timestamp of the beginning of the current cycle in microseconds (µs) |
| __SYSVA_MICROTCYMAXIMUM | UDINT | Read | Maximum cycle time since last start in microseconds (µs) |
| __SYSVA_MICROTCYCURRENT | UDINT | Read | Current cycle time in microseconds (µs) |
| __SYSVA_MICROTCYCYCTIME | UDINT | Read/Write | Programmed cycle time in microseconds (µs) |
| __SYSVA_RESNAME | STRING | Read | Resource name (max length=255) |

| Variable Name | Type | Read/Write | Description |
|---|---|---|---|
| __SYSVA_SCANCNT | DINT | Read | Input scan counter |
| __SYSVA_TCYCYCTIME | TIME | Read/Write | Programmed cycle time |
| __SYSVA_TCYCYCTIMEBASE | UDINT | Read/Write | Current cycle base time in milliseconds (ms) or microseconds (µs) |
| __SYSVA_TCYCURRENT | TIME | Read | Current cycle time |
| __SYSVA_TCYMAXIMUM | TIME | Read | Maximum cycle time since last start |
| __SYSVA_TCYOVERFLOW | DINT | Read | Number of cycle overflows |

| Variable Name | Type | Read/Write | Description |
|---|---|---|---|
| __SYSVA_RESMODE | SINT | Read | Resource execution mode. Possible modes are:<br>-4: Stopped in stepping mode after bound check exception<br>-3: Stopped in stepping mode after division by zero exception<br>-2: Stopped in stepping mode after exception<br>-1: Fatal error<br>0: No resource available<br>1: Stored resource available NOT USED (CMG)<br>2: Ready to run<br>3: Running in real time<br>4: Running in cycle by cycle<br>5: Stopped from encountering an SFC breakpoint<br>7: Stopped while in stepping mode |
| __SYSVA_CCEXEC | BOOL | Write | Execute one cycle when application is in cycle to cycle mode |

**Warning:** For the _SYSVA_CCEXEC system variable, its use in an ST program is not significant since resources run in cycle-to-cycle mode. Therefore, programs are not executed.

**To view system variables**

- From the Solution Explorer, double-click the **Global Variables** instance for the required resource.

The system variables are displayed in the grid.

# Logging Target Execution Events

You can log target execution events received from **ISaGRAF** targets. Logged events are stored in a log file, in Unicode format, located in the Events Logger folder of the current project's directory. A new log file is automatically created each day at 00:00:00 hours.

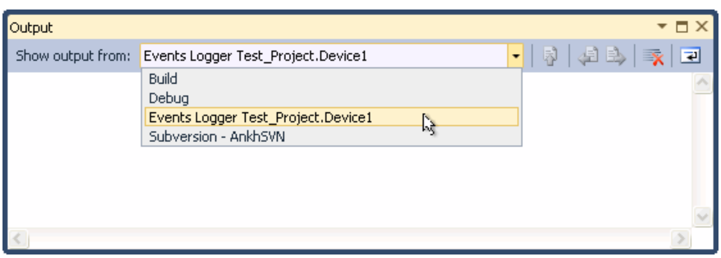
You can view log files in text format using a text editor.

When logging events from the workbench, the workbench automatically points towards the application's project and the logger is started. You can also choose to start logging events from a command line.

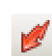
### To log target execution events

When logging events, the application must be online.

1. From the *Target Execution* toolbar, click  .
2. In the Output window, select the **Events Logger** option from the drop-down combo-box.




   The Output window displays the events in real-time and a copy of the events is saved in the log file.

3. To stop logging events, in the *Target Execution* toolbar, click .

### To view a log file

The default location of the log file is in the Events Logger folder of the project's directory. The name of the log file is Events_*YYYYMMDD*.txt where *YYYY* is the year, *MM* is the month, and *DD* is the day on which the file is created.

- In the Events Logger folder, double-click the .txt file.

## Forcing the Values of Variables

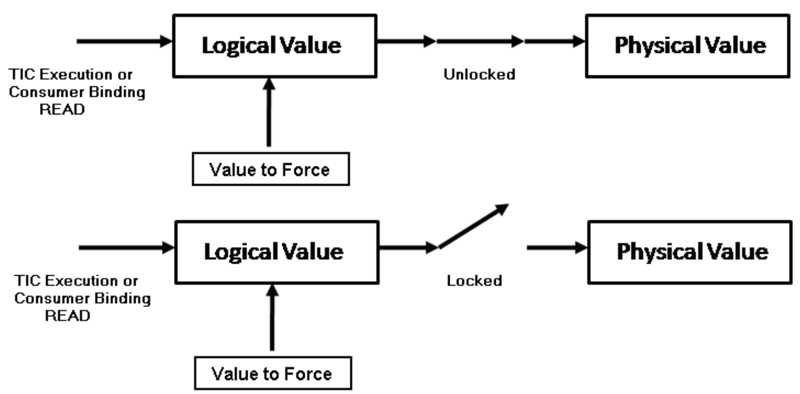
While debugging, you can force, i.e., override, the values of variables. These variables can be user-defined or directly represented. The behavior of a variable is defined by its logical value, physical value, lock state, and direction. When forcing the values of variables, the value to overwrite depends on the direction of the variable. You lock, unlock, and force the values of variables from the Dictionary.

Locking and unlocking operates differently for simple variables, array and structure elements, and function block parameters. For simple variables, individual variables are locked and unlocked directly. For simple-type members of a complex variable such as a structure or array, locking or unlocking any member affects the entire complex variable. For array and structure elements, locking and unlocking an element affects all members. For function block parameters, locking a parameter affects only that parameter. For function blocks, you need to instantiate these before locking their parameters.

For locked variables, the values displayed in the Logical Value and Physical Value columns differ depending on their direction:

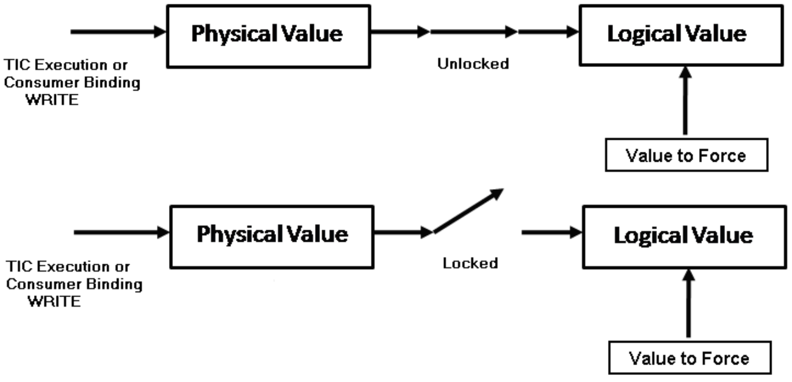
**Internal Variable (Read) Behavior**

Example: To force a counter for a function block.



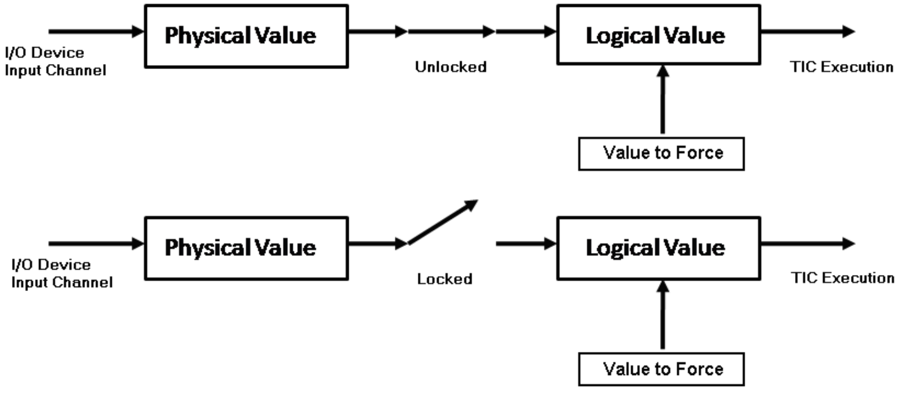
**Internal Variable (Write) Behavior**

Example: To force the result of an internal calculation.



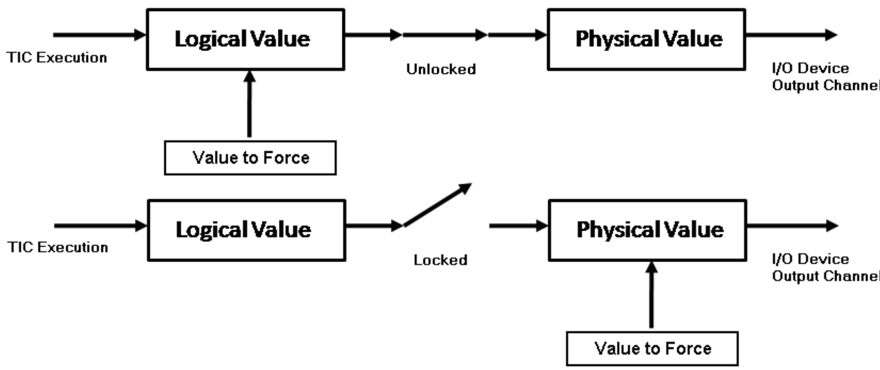

**Input Variable (Read) Behavior**

Example: To force the temperature reading from a sensor.



**Output Variable (Write) Behavior**

Example: To force the closing of an actuator valve.




When forcing the values of unlocked variables, these values may be overwritten by the next cycle execution.

**To force the value of a variable**

While debugging, you can force the values of locked user-defined or directly-represented variables.

1. From the Dictionary instance, double-click the required variable.

   The *Write* dialog box is displayed.

2. To modify the lock on the variable, in the *Lock* field, click the slider, then click **Write**.

3. To write the required value for the variable, modify the *DataType value* field, then click **Write**.

   When modifying a date in the *DataType value* field, a calender box is displayed. To select a date, click within the calender box. You can move between months using the arrow buttons.

**See Also**
Debugging

## Simulating

Simulating the running of an application signifies that virtual machines execute the code of individual resources and the Windows platform performs aspects such as POU execution. Virtual machines ignore inputs and outputs.

The compiler generates different code for simulation than for online.

Before simulating an application on a target, you need to build the project code.

**To simulate the running of an application**

1. From the Deployment view, specify the applicable target type and IP addresses for the devices in the project.

   a) Click the target, then from the Properties window, expand the *Hardware* node and in the *Target* property, select the required target from the drop-down combo-box.

   b) Click the connection between the target and the network, then from the Properties window, expand the *Info* node and type the required IP address in the field provided.

2. Build the project code.

3. In the Debug toolbar, from the drop-down combo-box select **Simulation**.

4. From the Debug menu, click **Start Debugging**.

# Monitoring

While running an application online, debugging, or simulating, you can monitor variables, updated by the running online (TIC) code or simulation code, in Dictionary instances as well as graphical programs and function block instances. For individual graphical POUs, you enable monitoring by generating symbols monitoring for each. Generating monitoring information increases the size of the TIC code created.

For dictionary instances, the logical values, physical values, and lock status of variables are displayed in their respective columns. For graphical programs and function block instances, values are displayed differently depending on their type:

- Boolean type variables are displayed using color. The variable color continues to the next input. The default colors are red when True and blue when False.
- SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, and STRING type variables are displayed as a numeric or textual value. When the variable is a structure type, the displayed value is the selected member.

When variables are unavailable, in Dictionary instances, the logical and physical values display the following messages:

- OFFLINE, indication that the variable is not present in the running application code
- WAIT, indication that the variable is either:
  - In online mode and attempting to connect to the target
  - In simulation mode and attempting to connect to the simulator

**To generate symbols monitoring information for a graphical POU**

When debugging graphical POUs, you can monitor the output values of functions and operators by enabling the *Generate Symbols Monitoring* property.

1. In the Solution Explorer, select the graphical POU for which to generate symbols monitoring.
2. In the Properties window, set the *Generate Symbols Monitoring* property to True.

**See Also**

Running an Application Online
Debugging
Simulating

# Getting Started

The **ISaGRAF 5** Concrete Automation Model enables the creation of applications supporting multi-process control. Applications consist of virtual machines running on hardware components, called target platforms. The development process consists of creating projects made up of devices representing individual target platforms, on which one or more instances of resources are downloaded. At runtime, instances of resources become individual virtual machines running on these target platforms.

Solutions containing one or more projects are developed in the following languages of the IEC 61131-3 standard: SFC: Sequential Function Chart, FBD: Function Block Diagram, LD: Ladder Diagram, SAMA (Scientific Apparatus Makers Association) diagrams, and ST: Structured Text. Using the IEC 61499 language, i.e., distribution method, enables the distribution of function blocks across multiple resources. When building, resources are compiled to produce very fast "target independent code" (TIC) or "C" code.

Within resources, you can declare variables using standard IEC 61131-3 data types (i.e., Boolean, integer, real, etc.) or user-defined types such as arrays or structures. Resources can share variables using external bindings. For IEC 61499 programs, bindings between function blocks declared in different resources are automatically created.

You develop projects on a Windows® development platform. The Automation Collaborative Platform graphically represents and organizes devices, resources, POUs, variables, and networks within a solution from many views:

- Add-in Manager
- Bindings
- Block Library
- Block Selector
- Controller Status
- Customize...
- Data Types
- Deployment View
- Description Window
- Device View
- Dictionary
- Document Overview
- Error List
- External Tools
- Find and Replace
- I/O Wiring
- ISaVIEW
- Language Editors
- Locked Variables Viewer
- Navigation Window

- Options...
- Output Window
- Parameters View
- Properties Window
- Solution Explorer
- Spy Lists
- Toolbox
- Variable Dependencies
- Variable Selector

Libraries made up of devices and resources enable defining functions, function blocks, global variables, arrays, and structures for reuse throughout projects.

Individual resources, from the devices making up a project, are downloaded, using the ETCP (TCP-IP), HSD, or ISARSI (Windows COM port) network, onto target nodes running real-time operating systems. Communication between devices can be implemented using the default TCP-IP network or proprietary network protocol.

You can choose to simulate the running of a project, after building a project, using high-level debugging tools, before actually downloading the resources making up devices to the target nodes.

When getting started, the following information guides you through the different facets:

- System Requirements for Development Platforms
- Differences with Previous Versions
- Naming Conventions and Limitations
- Introducing the Automation Collaborative Platform (ACP)
- Walking Through an Existing Application
- Starting with a Basic Application
- Importing an Existing Application

# System Requirements for Development Platforms

## Suggested Requirements

To use **ISaGRAF**, you need the following hardware and software.

### Hardware

- A computer with a 2.2 GHz or faster processor.
- RAM
  - 1 GB of RAM for x86 operating systems
  - 2 GB of RAM for x64 operating systems
  - When running ISaGRAF on a Virtual Machine, an additional 512 MB of RAM is necessary
- 4 GB of available hard disk space
- A hard disk running at 5400 RPM
- A CD-ROM drive on the Windows network (for installation from disk)
- A TCP/IP network
- An SVGA monitor having at least 1024 X 768 pixels screen resolution
- A DirectX 9-capable video card that runs at a display resolution of 1024 x 768 or higher

### Software

**ISaGRAF** supports the following operating systems:

- Windows® 7 (x86 and x64)
- Windows® 8 (x86 and x64)

**Note:** If Visual Studio 2010 was previously installed, when running the ISaGRAF installation the Visual Studio 2010 Service Pack 1 will be installed. This may affect Visual Studio functionality.

## Differences with Previous Versions

For users of previous versions of **ISaGRAF**, the following list compares different aspects of the workbench:

| ISaGRAF 6.x Workbench | ISaGRAF 5.x Workbench |
|---|---|
| Provides a tab-oriented environment enabling navigation between multiple POUs and Dictionary instances | Provides a window/editor-based environment requiring closing the Dictionary before using newly-defined variables in a POU. |
| Supports the FBD, LD, SFC, and ST IEC 61131-3 programming languages as well as SAMA diagrams using the FBD programming language. Also supports the IEC 61499 programming language. | Supports the FBD, LD, SFC, LD, FC, and IL IEC 61131-3 programming languages. |
| The Solution Explorer provides an organized view of projects and their elements. The Solution Explorer can display multiple projects. You can also perform many tasks from this view. | The Project Tree view displays the project structure and enables accessing most aspects of a currently opened project. |
| The dictionary displays variables in contextual instances for individual resources and POUs.<br>In accordance with the IEC 61131-3 standard, the available options for dictionary variables are the following:<br>1) For Attribute: Read/Write, Write, and Read.<br>2) For Direction: Var (replacing Internal), VarOutput (replacing Output), VarInput (replacing Input), VarDirectlyRepresented, and VarGlobal. | The dictionary displays all variables for individual resources.<br>The available options for dictionary variables are the following:<br>1) For Attribute: Free, Read, and Write<br>2.) For Direction: Internal, Output, and Input |
| Opening projects in Read-Only mode is not available. | Can open an existing project in Read-Only mode. |

| ISaGRAF 6.x Workbench | ISaGRAF 5.x Workbench |
|---|---|
| Supports external bindings. Internal bindings are automatically converted to external bindings. | Supports external and internal bindings |
| Devices represent target definitions; devices replace configurations | Configurations represent target definitions |
| Elements are imported/exported in compressed 7-zip (.7z) exchange files | Elements are imported/exported in compressed .PXF exchange files |
| POUs are displayed and edited using language containers in the language editor. POUs are also displayed in the Solution Explorer. | POUs are displayed in the link architecture view. POUs are editable using various language editors. |
| In the language editor, a document overview and zooming enables focusing on areas of the workspace | In the language editor, zooming enables focusing on POUs displayed in the workspace |
| You define the properties of projects, devices, resources, and POUs by selecting the element in the Solution Explorer and entering the required information in the Properties window. | You define the properties of project elements from various dialog boxes accessed from the menus. |
| After rebuilding the solution, online changes are not permitted | You can perform online changes after rebuilding projects |
| You can download code for resources, devices, or projects and store it on the target | You can download code for resources and store it on a disk or use another storage method |
| You can set a password for devices (sets the access control for the target), projects, POUs, and library functions and function blocks | You can set a password for resources, projects, POUs, and targets |
| The Toolbox displays the language-specific elements for a selected POU. | The Toolbar displays the language-specific elements for a selected POU. |
| I/O wiring is performed from the I/O Device View, which is accessed from the contextual menu for the resource within the Solution Explorer. | I/O wiring is performed from the I/O wiring tool |

| ISaGRAF 6.x Workbench | ISaGRAF 5.x Workbench |
|---|---|
| The Deployment View graphically displays the devices, networks, and connections of projects and solutions in a separate window. | The hardware architecture view graphically displays the configurations, networks, and connections of a project in the workspace. |
| You can generate documentation for projects, devices, resources, POUs, and variables. You can specify the following options:<br>• Choose the Sections template, modifying the items listed in the Sections pane.<br>• Set the page orientation for the generated documentation<br>• Set the page size for the generated documentation<br>• Set the margins for the generated documentation<br>• Select the Microsoft Word® template, includes whether footers are included<br>• Set the POU diagram scaling<br>• Set the link type<br>• Set the comment style<br>• Define the output file name and location | You can build and print documentation for open projects. You can specify the following options:<br>• Set the page orientation for printing. You can choose to print FBD and LD diagrams differently from the rest of the document. Diagrams are automatically scaled to fit the paper<br>• Enable displaying headers/footers on each page and on cover pages<br>• Choose the image displayed and select the format for headers/footers<br>• Specify the page numbering method<br>• Choose to include printing history on cover pages<br>• Enable displaying margins and define margin width<br>• Define the font used to print text |
| Description window enables displaying and editing text descriptions for projects, devices, resources, and POUs. Clicking items within the Solution Explorer refreshes the contents of the Description window, enabling the exploration of projects. | Descriptions are accessed via contextual menus for programs and resources displayed in the link architecture and hardware architecture views, respectively. Descriptions contain editable text only. |

| ISaGRAF 6.x Workbench | ISaGRAF 5.x Workbench |
|---|---|
| Error List window displays warnings, errors, and messages generated when building projects and solutions, projects, devices, resources, or POUs. Errors in the code are accessed directly from the Error List view. | Error List window displays warning, errors, and messages generated by building solutions, projects, configurations, resources, or POUs. Errors in the code are accessed from the Output window. |
| Cross Reference Browser displays information including names, properties, locations, and comments associated with the variables, programs, functions, function blocks, and defined words used within projects. | The Browser displays information including names and locations associated with the variables, programs, functions, function blocks, and defined words used in projects. |
| Add-in Manager enables specifying the loading method of registered add-ins displayed in the dialog box. You can define whether add-ins loads at start-up time, using command line prompts, or both. You can also enable and disable add-ins displayed in the Add-in Manager dialog box. | ProHook dynamic link library enables the usage of user-programmed functions (i.e. hook functions) with the ISaGRAF workbench. At start-up time, the library is loaded and the hook functions are enabled. |
| External Tools option enables launching other applications from inside **ISaGRAF**. | Tools contained in external applications are launched from outside **ISaGRAF**. |
| Renumbering addresses is not available | Renumbering addresses automatically generates contiguous addresses in the variables grid |
| Default fonts and colors are modified in the Options dialog box. | Fonts and colors are customized using the options available in the customization editor. |

# Naming Conventions and Limitations

**Projects**

| | |
|---|---|
| Project names | Project names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |
| Device quantity | Projects can contain multiple devices |

**Resources**

| | |
|---|---|
| Resource names | Resource names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |

**Devices**

| | |
|---|---|
| Device names | Device names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |

**Networks**

| | |
|---|---|
| Network instances | Projects can have an unlimited quantity of network instances |

**POUs (Programs, Functions, and Function Blocks)**

| | |
|---|---|
| POU names | POU names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |
| POUs per project | Projects can contain up to 65 536 POUs |
| Function parameters | Functions can have a maximum of 128 parameters (127 inputs and one output) |
| Function parameter names | Function parameter names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |
| Function block parameters | Function blocks can have a maximum of 128 parameters (inputs and outputs) |
| Function block parameter names | Function block parameter names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |

**Variables**

| | |
|---|---|
| Variable name | Variable names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |
| BOOL variables | Boolean variables can have the boolean value TRUE (1) or FALSE (0). |
| SINT variables | SINT variable integer values range from -128 to +127. Short integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| USINT variables | USINT variable integer values range from 0 to 255. Unsigned short integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| BYTE variables | BYTE variable integer values range from 0 to 255. BYTE constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| INT variables | INT variable integer values range from -32768 to +32767. Integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| UINT variables | UINT variable integer values range from 0 to 65535. Unsigned integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |

| | |
|---|---|
| WORD variables | WORD variable integer values range from 0 to 65535. WORD constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| DINT variables | DINT variable integer values range from -2147483648 to +2147483647. Double integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| UDINT variables | UDINT variable integer values range from 0 to 4294967295. Unsigned double integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| DWORD variables | DWORD variable integer values range from 0 to 4294967295. Double word constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| LINT variables | LINT variable integer values range from -9223372036854775808 to +9223372036854775807. Long integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| ULINT variables | ULINT variable integer values range from 0 to 18446744073709551615. Unsigned long integer constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |
| LWORD variables | LWORD variable integer values range from 0 to 18446744073709551615. Long word constants must begin with a prefix identifying the base. There is no prefix for DECIMAL values. For HEXADECIMAL values the prefix is "16#", for OCTAL values the prefix is "8#", and for BINARY values the prefix is "2#". |

| | |
|---|---|
| REAL variables | Real variables have six significant digits. For larger values, the maximum possible value is ±3.402823466E+38 while for smaller values, the minimum possible value is ±1.175494351E-38. Therefore, values greater than ±3.402823466E+38 and greater than 0.0 but less than ±1.175494351E-38 are not supported. Real literal values can be written with either decimal or scientific representation. The exponent part of a real scientific expression must be a signed integer value ranging from -37 to +37. The scientific representation uses the 'E' letter to separate the mantissa part and the exponent. |
| LREAL variables | Long real variables have 15 significant digits. For larger values, the maximum possible value is ±1.7976931348623158e+308 while for smaller values, the minimum possible value is ±2.22507385850721E-308. Therefore, values greater than ±1.7976931348623158e+308 and greater than 0.0 but less than ±2.22507385850721E-308 are not supported. Long real literal values can be written with either decimal or scientific representation. The range of a real scientific expression must be a signed integer value from 1.7E -308 to 1.7E +308. |
| TIME variables | Time variables can have positive values ranging from 0 to 49d17h2m47s294ms. The time literal value must begin with the "T#" or "TIME#" prefix. |
| DATE variables | Date variable values range from 1970-01-01 to 2038-01-18. The date literal expression must begin with the "D#" or "DATE#" prefix. |
| STRING variables | STRING variable string capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string. String variables can contain any character of the standard ASCII table. Characters must be preceded and followed by single quote (') characters. When placing single quote (‘) characters within a string literal, these characters must be preceded by the dollar ($) character. |
| Alias names | Alias names can have up to 128 characters consisting of letters, digits, and the following special characters: !, #, $, %, &, \, *, +, -, /, <, :, =, >, ?, @, ^, _, `, \|, and ~. |
| Address | The user-defined address of a variable consists of four digits in hexadecimal format ranging from 0001 to FFFF. |

| | |
|---|---|
| Array names | Array names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |
| Structure names | Structure names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |
| **Defined Words** | |
| Defined Word names | Defined word names can have up to 128 characters and must begin with a letter or single underscore followed by letters, digits, and single underscores. |
| Defined word equivalents | Defined word equivalents can have up to 128 characters. |
| **I/O Wiring** | |
| I/O device order | The I/O device order ranges from 0 to 65535 |
| **SFC Programs** | |
| Priority of transitions | The priority of transitions value ranges from 1 to 255. |
| **Access Control** | |
| Password definitions | Password definitions are limited to eight characters consisting of letters, digits, and symbols. |

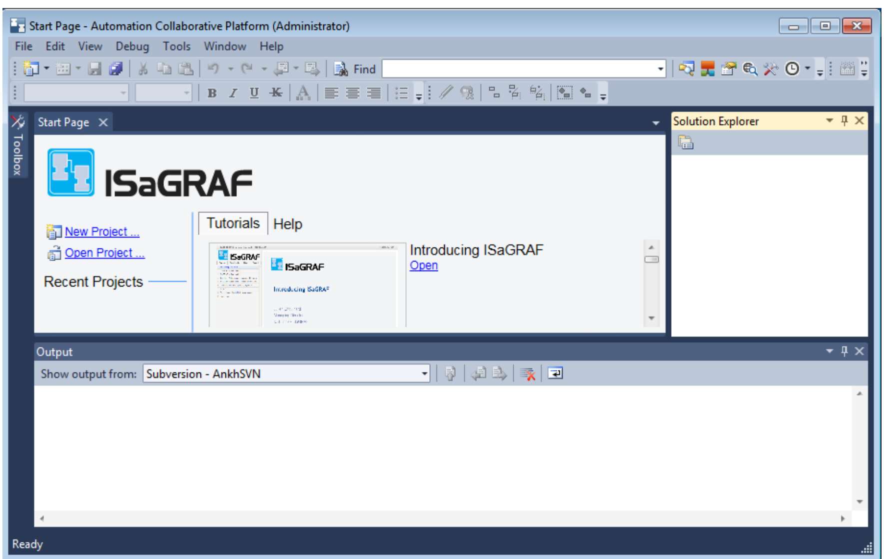
# **Introducing the** Automation Collaborative Platform (ACP)

The **Automation Collaborative Platform** (**ACP**) provides a robust integrated development environment (IDE) enabling the development of process control applications. The **ACP** workbench offers a complete suite of tools for building applications.

**To get to know the different aspects of the ACP**

1. From the Start menu, click **All Programs**, then **ISaGRAF 6.4**, and then click **Automation Collaborative Platform**.

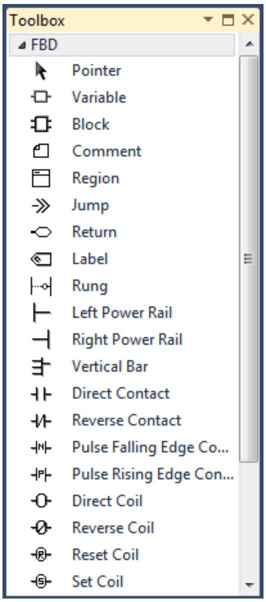
   The **ACP** is launched displaying the Start Page, Solution Explorer, and Output window. The Toolbox is displayed in auto hide mode.



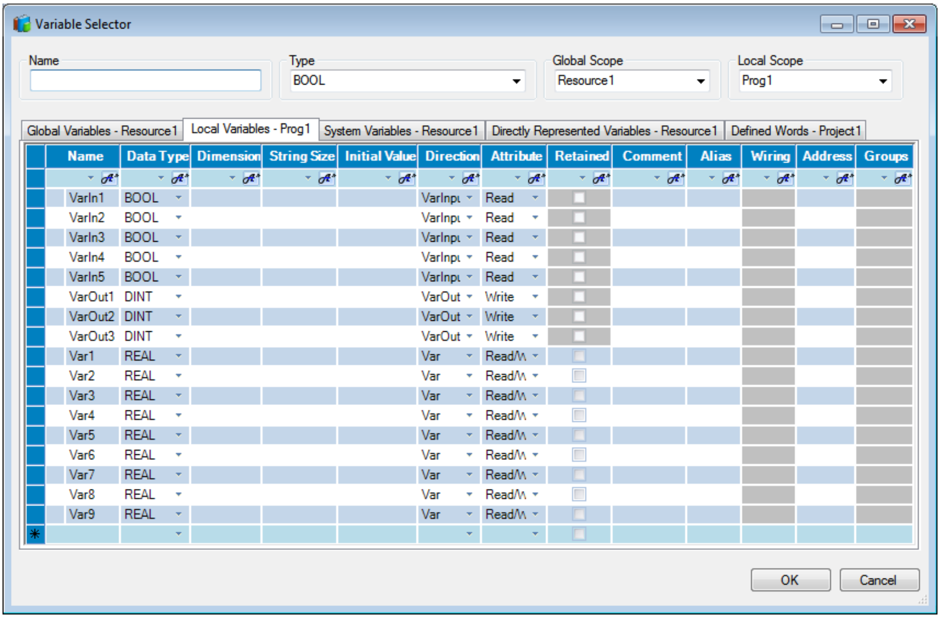

   The **Start Page** enables opening new or recent projects, viewing tutorials, as well as accessing the Getting Started help pages. The **Solution Explorer** displays open solutions consisting of projects and their elements. The **Output window** displays the compilation progress and errors. The **Toolbox** displays the available elements for insertion in programs.

2. When adding elements in the language container, you can use the following **ACP** features:

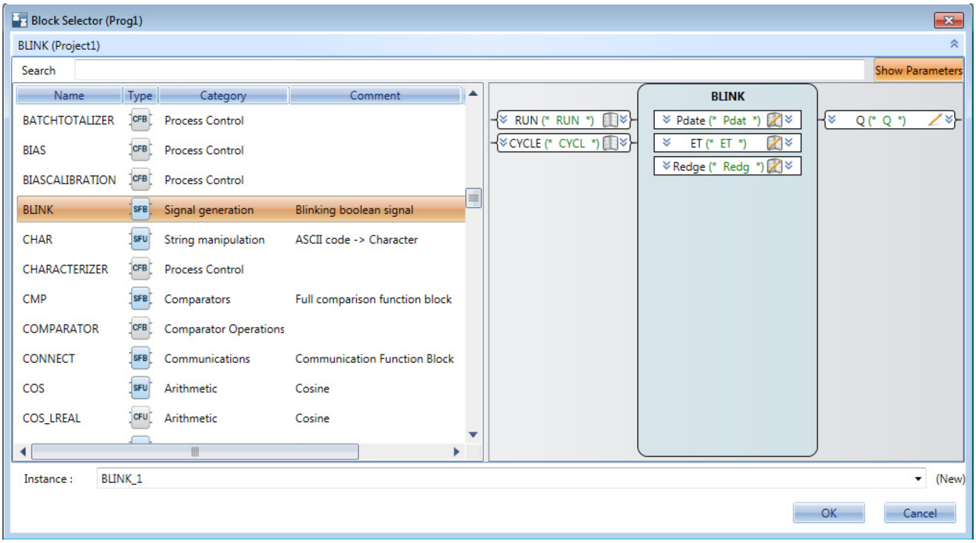
   - To display program-specific elements for insertion in the language container, from the View menu, click **Toolbox**.



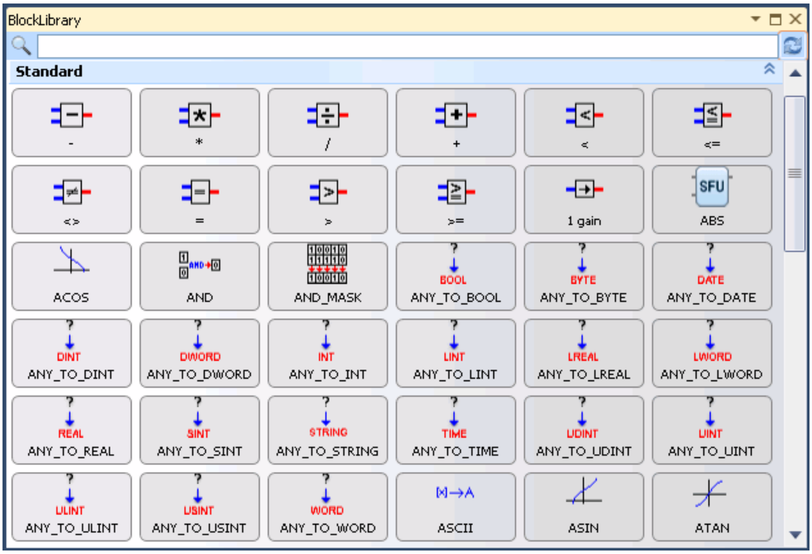
- To display variables defined for a program, from the Toolbox, drag the Variable icon into the language container. The **Variable Selector** is displayed.



- To display the list of the blocks available for a program, from the Toolbox, drag the Block icon into the language container. The **Block Selector** is displayed. You can also access the **Parameters** display from the Block Selector.

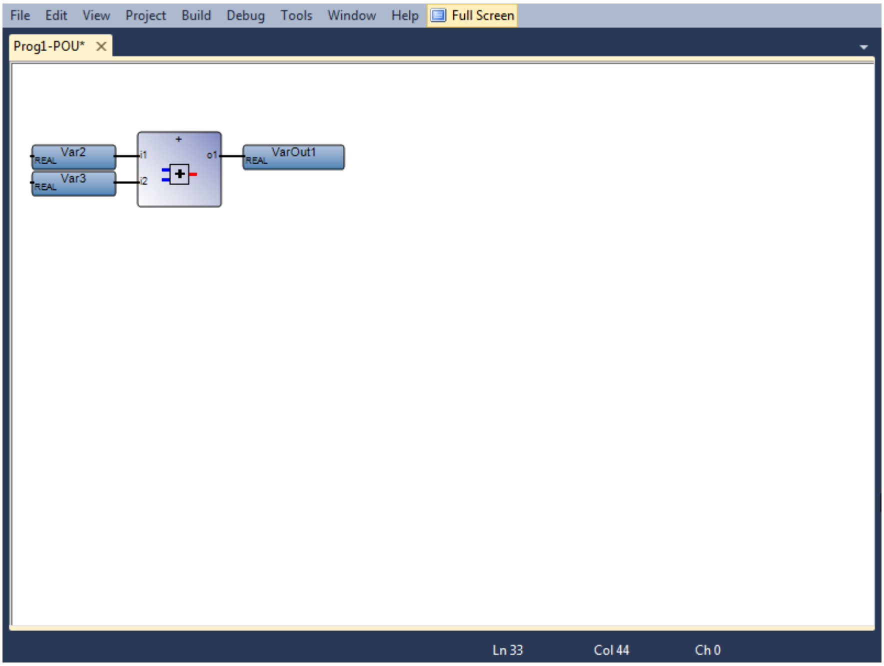


- To display a graphical view of standard operators, as well as standard and user-defined functions and function blocks available for the POUs of a project, from the View menu, click **Block Library**.

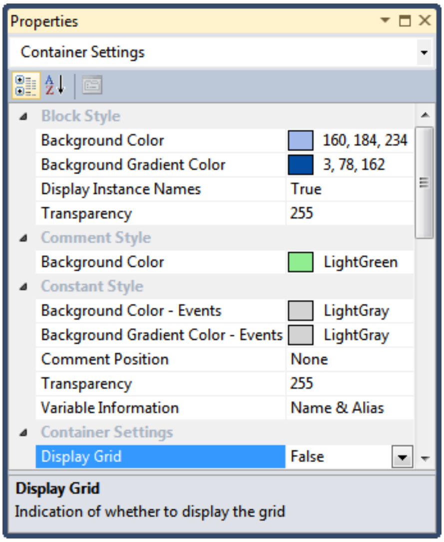



- To view, add, or edit the rich text descriptions for ISaGRAF project elements, select the required element in the Solution Explorer, then from the View menu, click **Description Window**.

3. To work in full screen mode, from the View menu, click **Full Screen**. Full screen mode enlarges the workspace to fill the screen, hiding other tabbed windows.

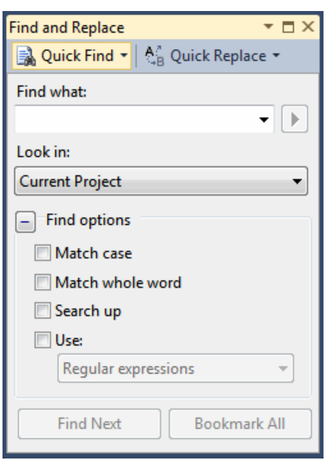



4. To display the Properties window, from the View menu, click **Properties Window**. The properties window enables viewing and editing the properties of items selected within language containers, ISaVIEW instances, the Solution Explorer, and the Deployment View. You can view properties alphabetically or categorically.

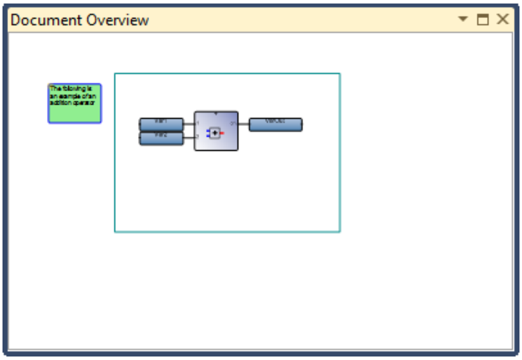



5. You can navigate through program content, including application code, using the following **ACP** features:

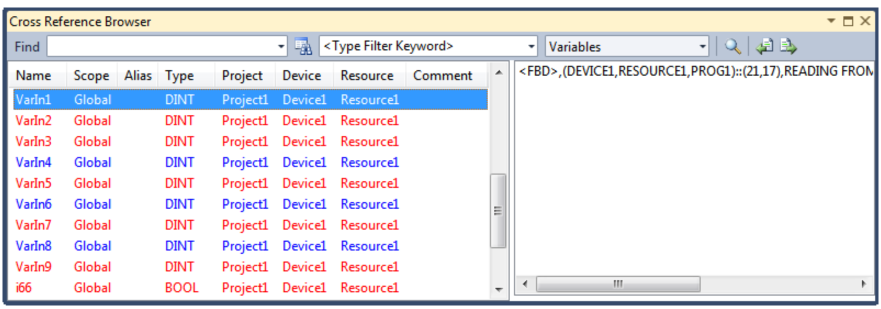
    - To find and replace strings and expressions in files, from the Edit menu, point to **Find and Replace**, and then click the required option. For example, click Quick Find to display Quick Find options.



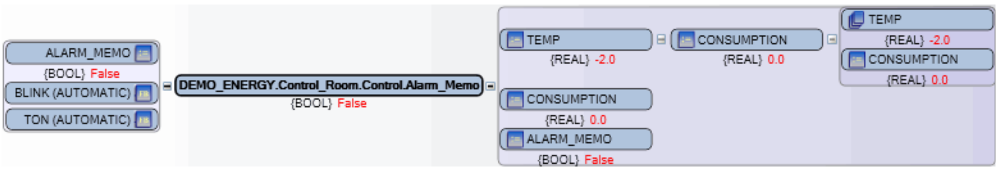
- To focus on an area displayed within a program opened for editing, from the View menu, click **Document Overview**.




- To view and jump to instances of ISaGRAF elements within a project, from the View menu, click **Cross Reference Browser**, refresh the list of cross references, then locate the required element by entering the required search information.



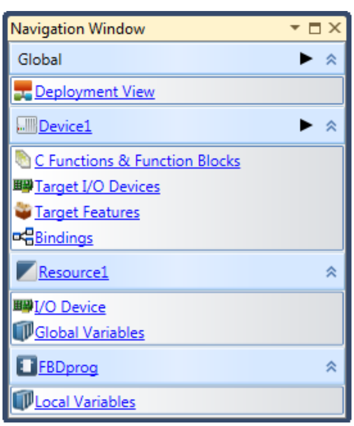
- To view the ascending and descending dependencies of variables, from a graphic program or dictionary instance, right-click the required variable, and then click **Dependencies**. Before viewing variable dependencies, refresh the cross references for the project.




Dependencies are also available while editing, debugging, or running online.

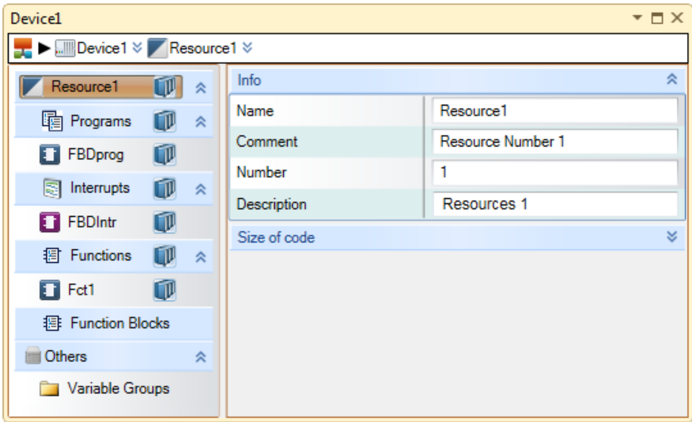
6. You can navigate through the different elements and aspects of projects using the following **ACP** features:

   - To navigate through project aspects and elements, from the View menu, click **Navigation Window**. The environment provides the global view (listing the devices contained in one or more projects within a solution), the deployment view, and the device view.

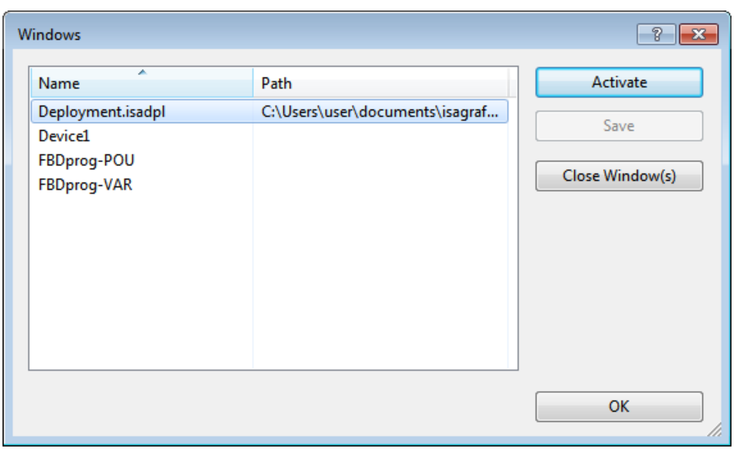



The initial aspects and elements displayed vary depending on the item selected in the Solution Explorer.

- To navigate through project elements, from the Solution Explorer, right-click the required device and then click **Open**. The **Device View** is displayed and enables accessing device and resource information such as available POUs, C function and function block parameters, interrupts, target I/O devices, target features, and resource properties.

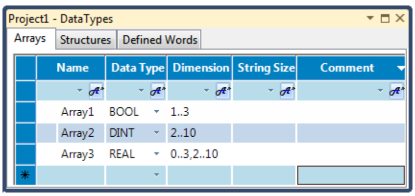



- To navigate through Active Files open in the current project, from the Window menu, click **Windows**. Active files consist of language containers, the Deployment view, and other windows docked in the workspace.

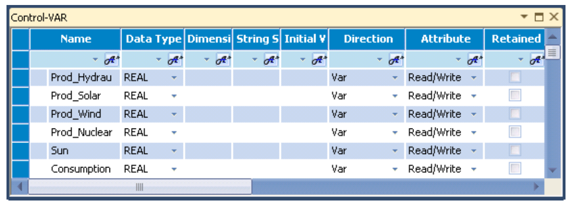



7. When managing elements, you can use the following **ACP** features:

- To manage local variables, global variables, arrays, structures, and defined words, in the Solution Explorer, double click the required Local Variables, Global Variables or Data Type instance. The **Dictionary** is displayed.

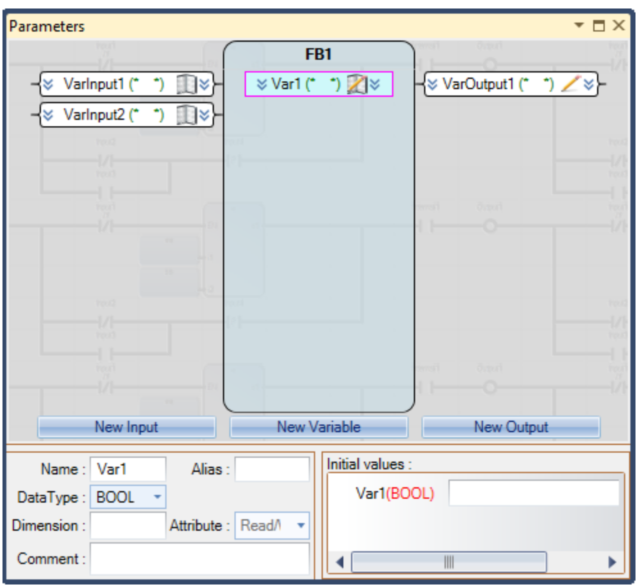



| Name | Data Type | Dimension | String Size | Comment |
|---|---|---|---|---|
| Array1 | BOOL | 1..3 | | |
| Array2 | DINT | 2..10 | | |
| Array3 | REAL | 0..3,2..10 | | |

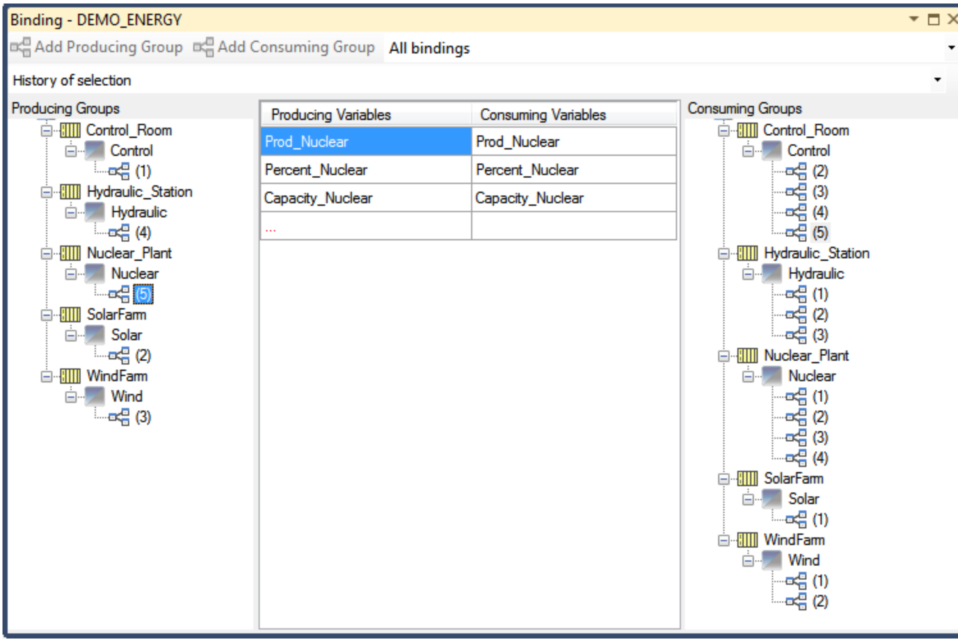



| Name | Data Type | Dimensi | String S | Initial V | Direction | Attribute | Retained |
|---|---|---|---|---|---|---|---|
| Prod_Hydrau | REAL | | | | Var | Read/Write | |
| Prod_Solar | REAL | | | | Var | Read/Write | |
| Prod_Wind | REAL | | | | Var | Read/Write | |
| Prod_Nuclear | REAL | | | | Var | Read/Write | |
| Sun | REAL | | | | Var | Read/Write | |
| Consumption | REAL | | | | Var | Read/Write | |

- To manage parameters and local variables for user-defined POUs, right-click the POU, and then click **Parameters**.

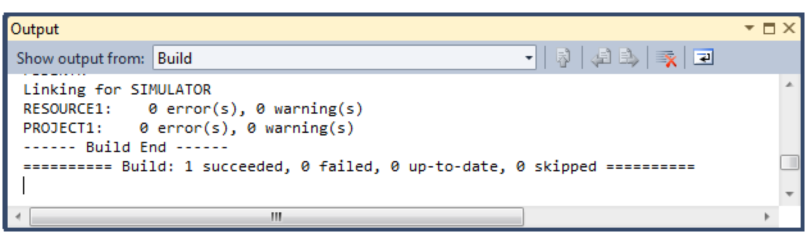



**8.** To create external bindings, i.e., access paths, between variables located in different resources, in the Solution Explorer, right-click the project, device, or resource and then click **Binding**.

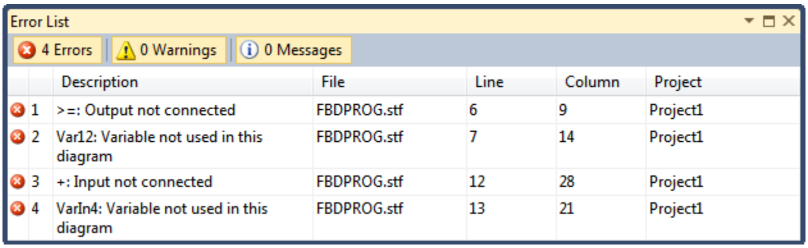



9. When debugging applications, you can oversee application performance using the following **ACP** features:

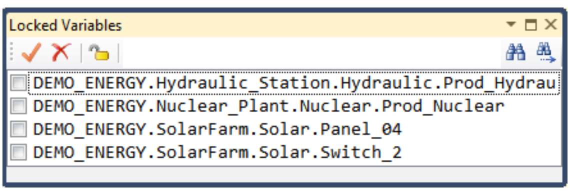
- To view the build information, from the View menu, click **Output**.



- To view the errors, warnings, and messages produced when editing and building programs, from the View menu, click **Error List**.

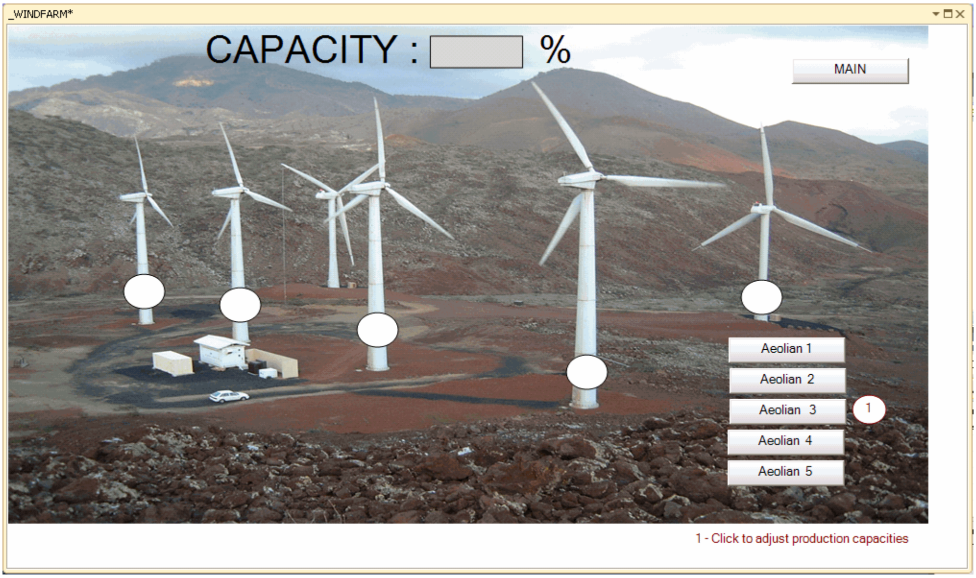



- To view or unlock locked variables while debugging, running online, and simulating, from the Debug menu, click **Locked Variables**.



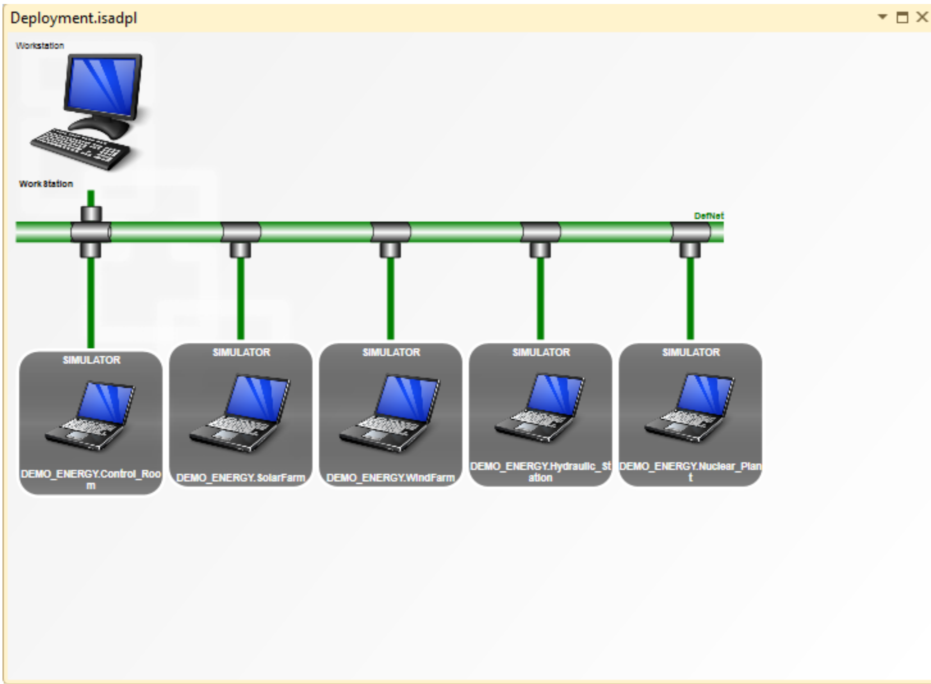

**10.** To add an ISaVIEW screen, right-click the device, resource, or program in the Solution Explorer, point to **Add**, and then click **New ISaVIEW.**

You can monitor or run control processes, locally or remotely, by creating ISaVIEW screens. You can define animation effects for the objects inserted in the ISaVIEW screens. Design mode enables editing the screen objects and animation mode executes the animation effects.

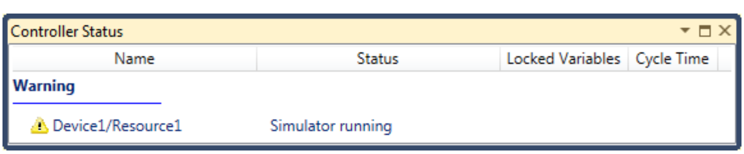



**11.** You can view information about devices using the following **ACP** features:

- To graphically display the devices, networks, and connections of a project, from the View menu, click **Deployment View**.




- To access real-time status information for all devices in a project, from the View menu, click **Controller Status**.

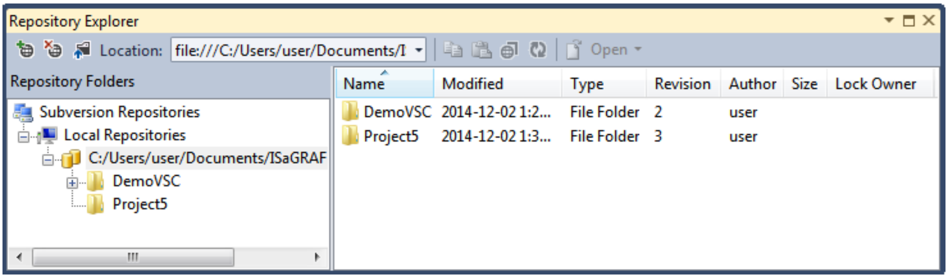



**12.** To implement a failover mechanism where a secondary device takes over if the primary device fails, in the Solution Explorer, right-click the device, and then click **Failover Configuration**.

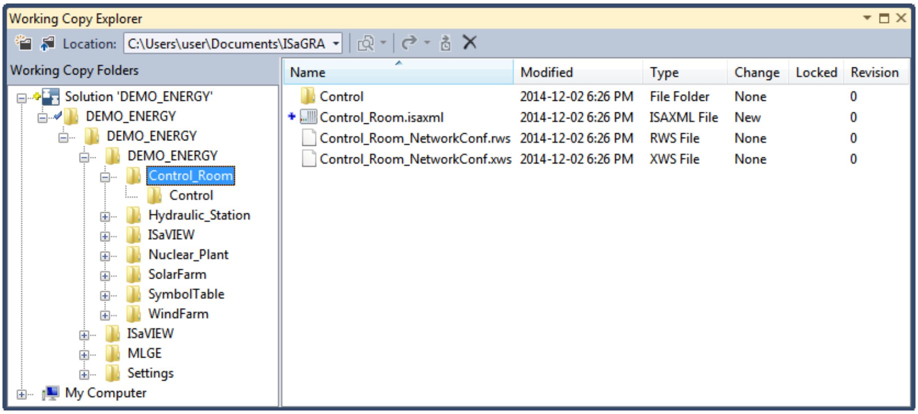
The failover mechanism is available with the failover project template.

13. For version source control, the following options are available for managing changes to ISaGRAF elements:

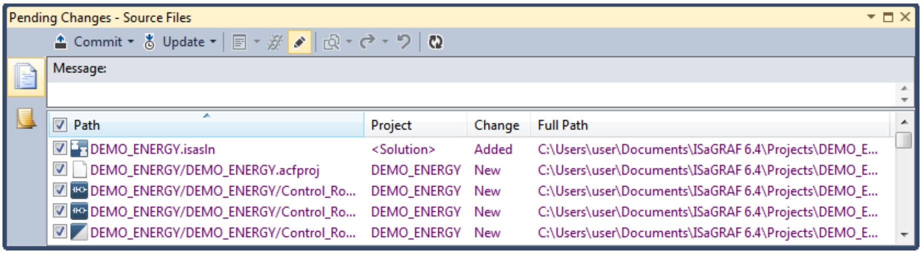
- To view the files for the elements of repositories, from the View menu, click **Repository Explorer**.



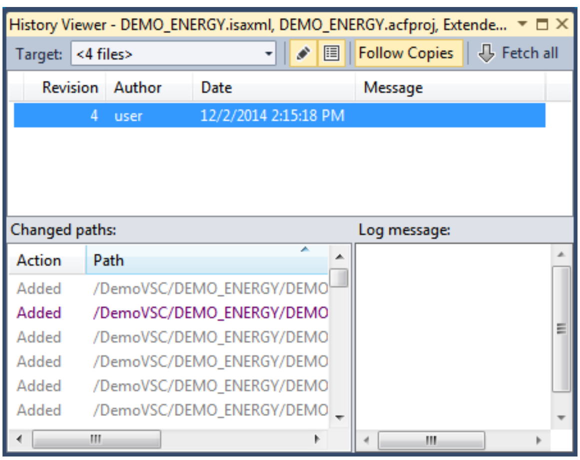

- To view the directories and files of local working copies from repositories, from the View menu, click **Working Copy Explorer**.



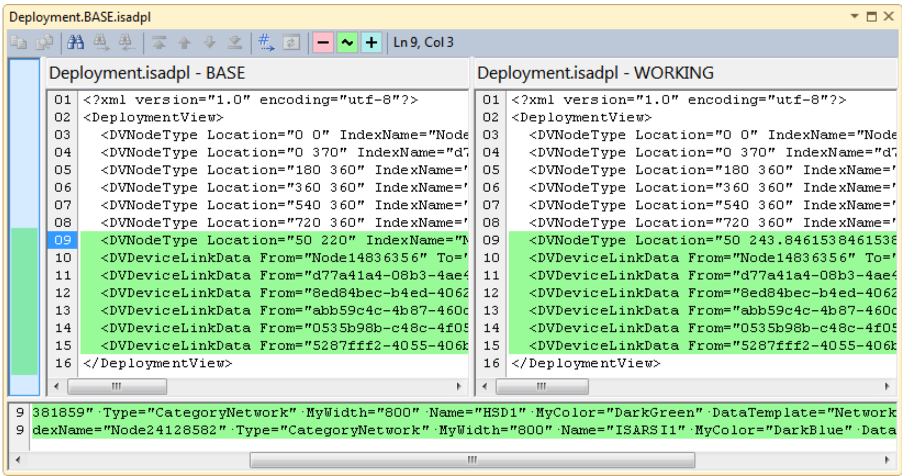
- To view a list of project changes that have not yet been committed to version source control, from the View menu, click **Pending Changes.**



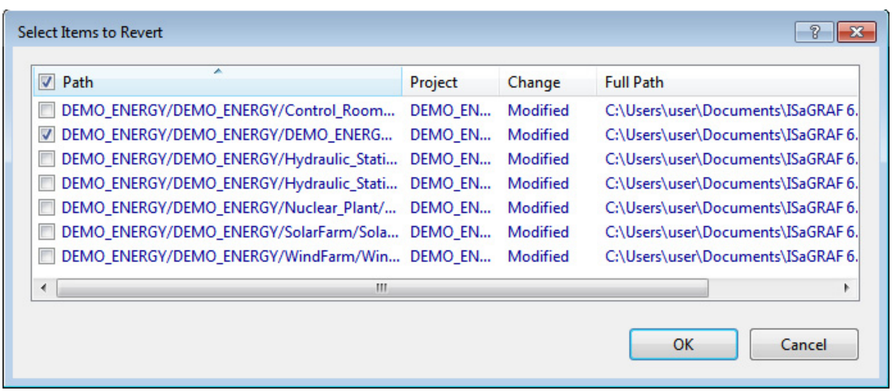

- To view the history of elements committed to version source control, from the File menu, point to Subversion, then click **View History**.



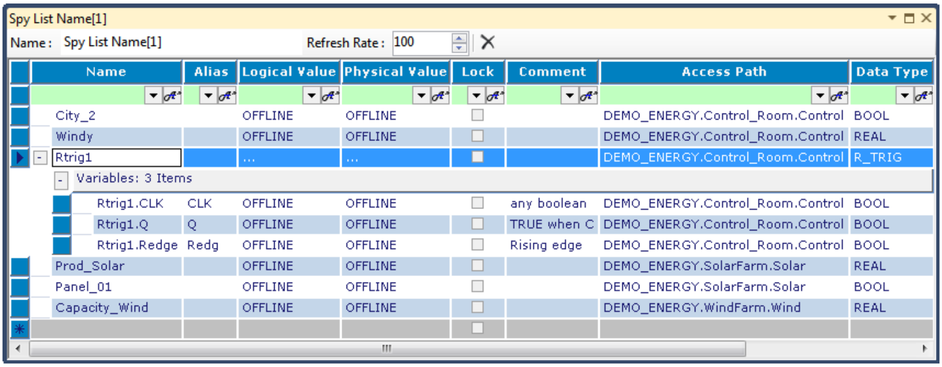
- To compare different version of elements committed to version source control, from the File menu, point to Subversion, and then click **Compare.**



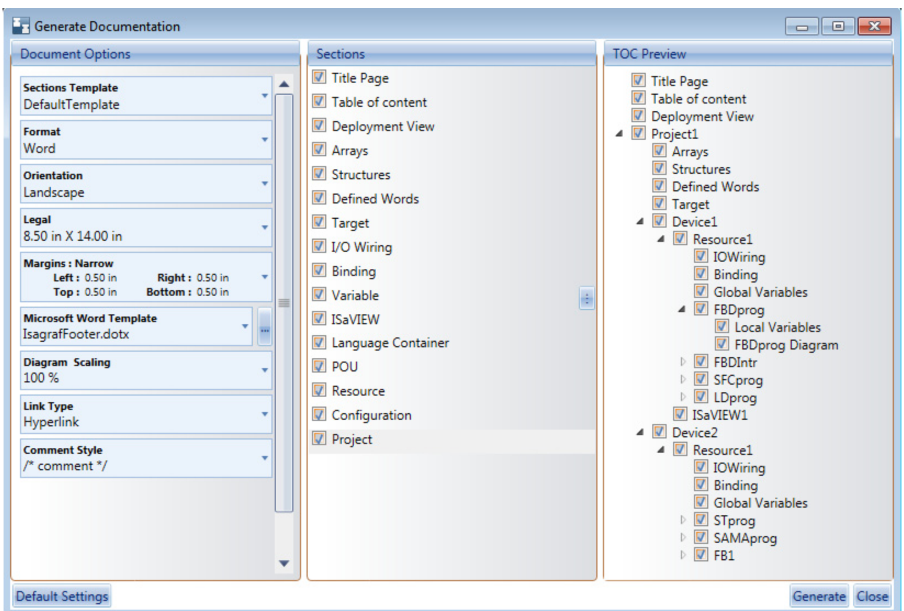

- To revert elements to a prior version, from the File menu, point to Subversion, and then click **Revert**.




**14.** To view changes in the values of variables and function block instances, from the Debug menu, point to **Spy Lists,** then click the required spy list instance.

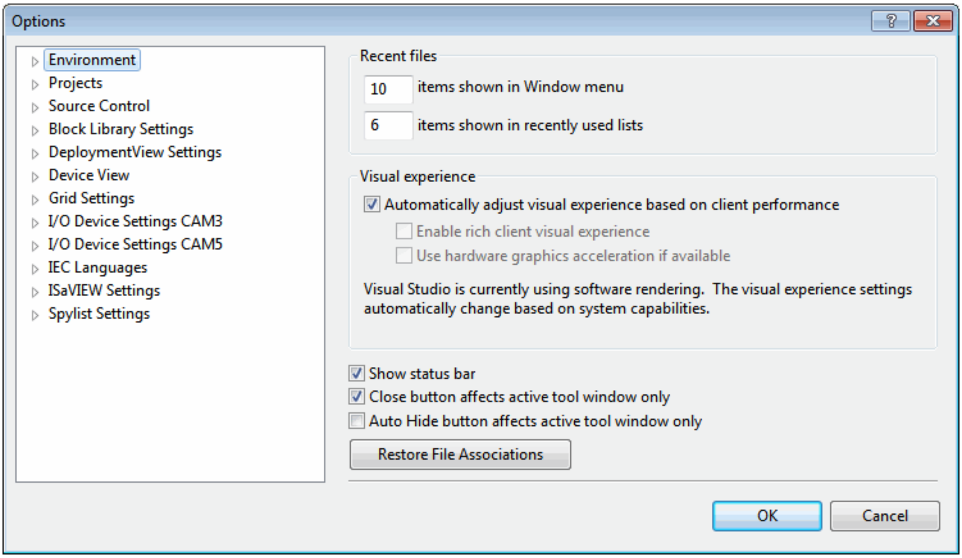


15. To generate documentation for projects, devices, resources, programs, and variables, from the File menu, click **Generate Documentation**.

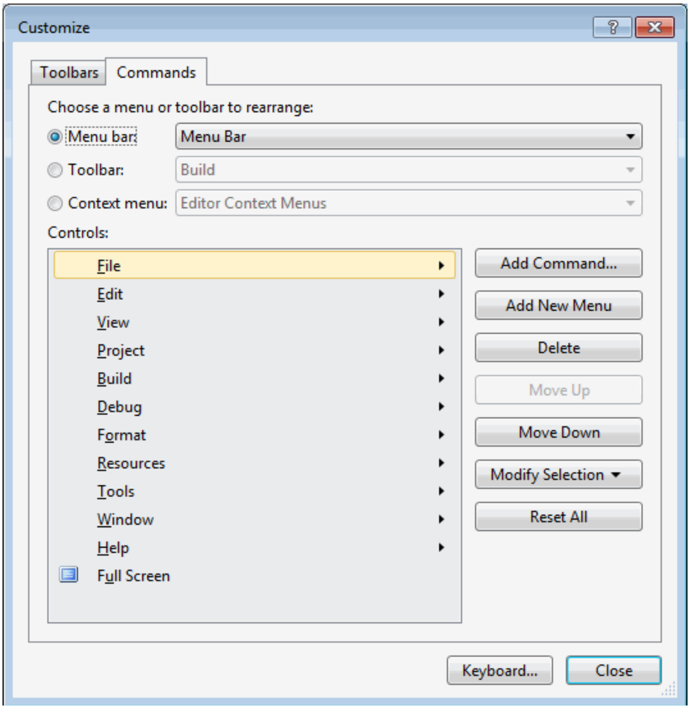


16. You can customize the Workbench using the following **ACP** features:

- To customize the environment, project, Source Control, Block Library, Deployment view, Device view, various grids, I/O device, IEC languages, ISaVIEW, and Spy List options, from the Tools menu, click **Options...**

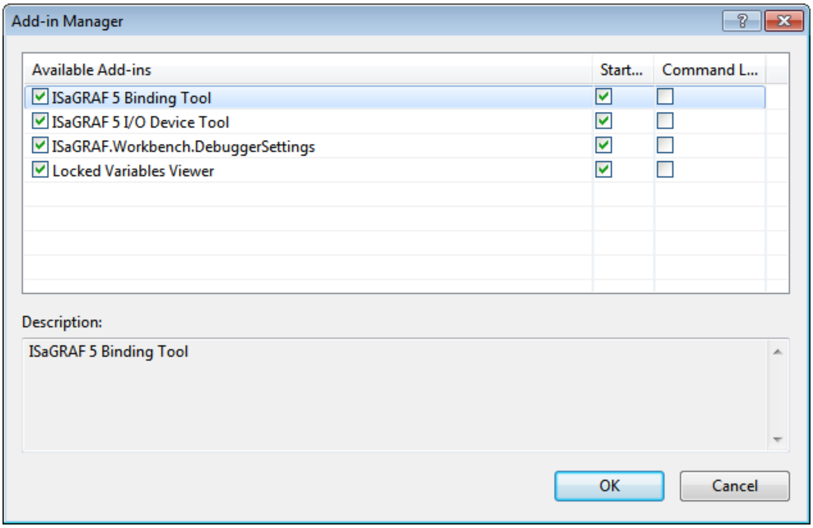


- To create or customize Toolbars, Menu bars, and Context menus, from the Tools menu, click **Customize...**

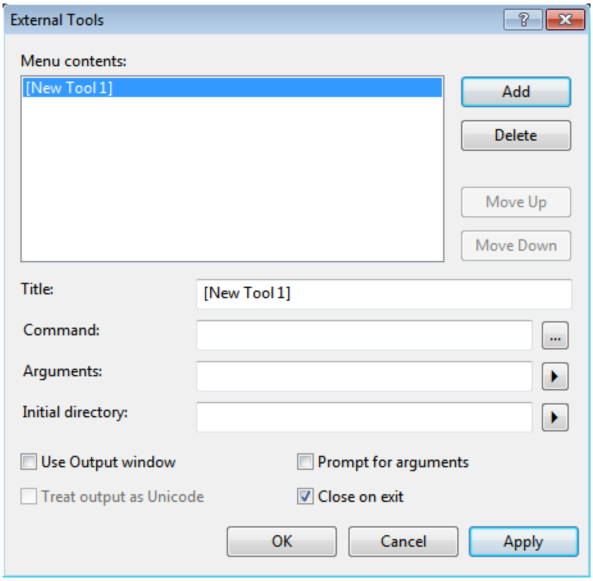



**17.** You can manage add-ins and external tools using the following **ACP** features:

- To manage registered add-ins, from the Tools menu, click **Add-in Manager...**




- To add external tools, from the Tools menu, click **External Tools...**



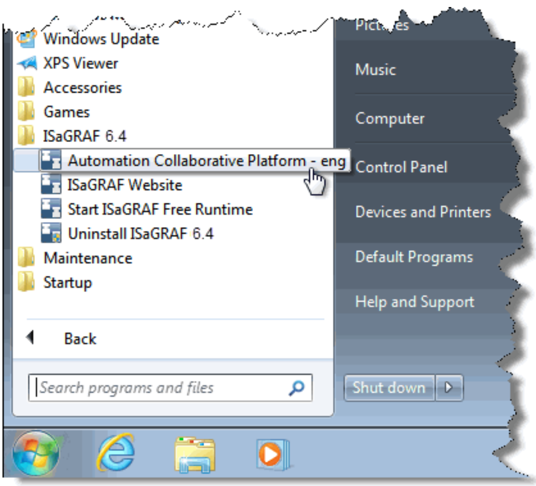
# Walking Through an Existing Application

This section describes a demo project included with the default installation.

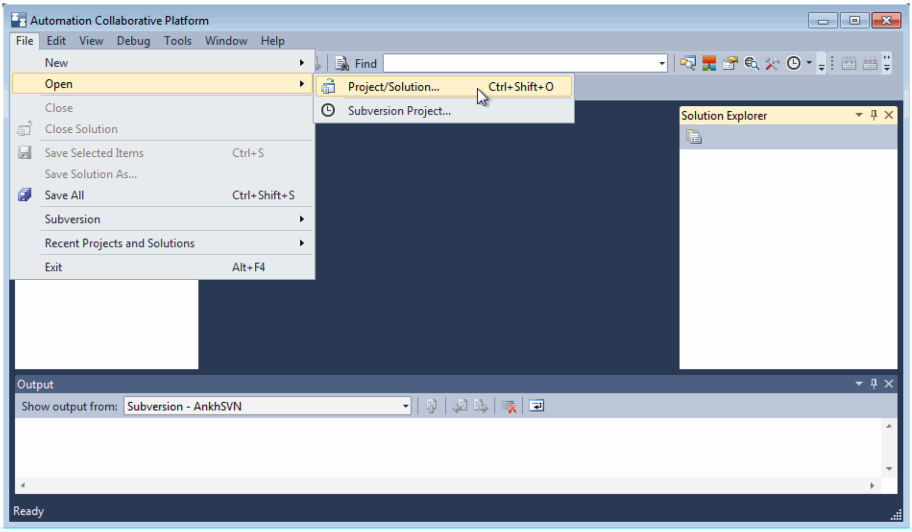
**To walk through an existing application**

1. Launch the **ACP** and open an existing application.

   **a)** From the Start menu, point to **All Programs**, click **ISaGRAF 6.4**, and then click **Automation Collaborative Platform**.

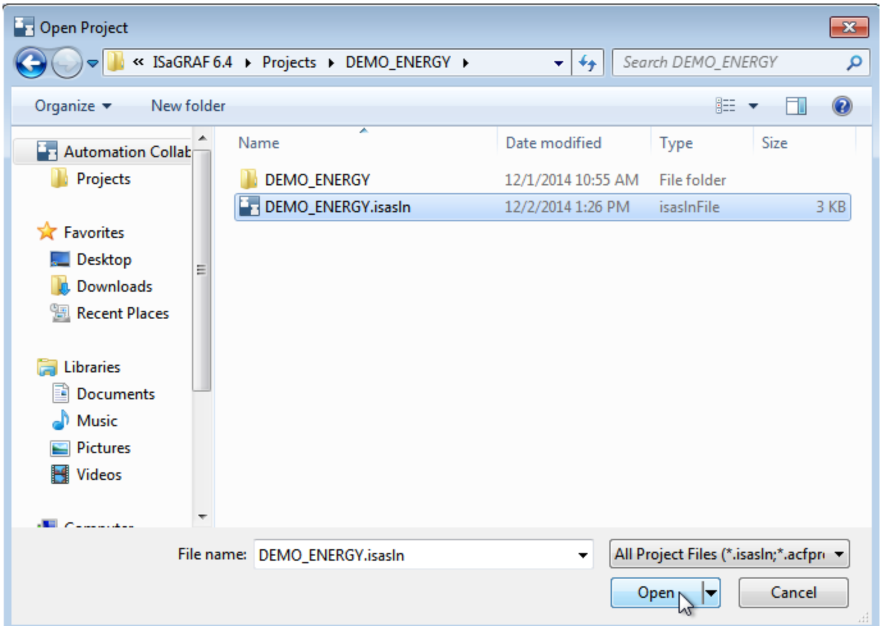



   **b)** From the File menu, point to **Open**, then click **Project/Solution...**.




**c)** In the Open Project dialog box, select and open the **DEMO_ENERGY.isasln** project, located in the following directory:

%USERPROFILE%\My Documents\ISaGRAF 6.*x*\Projects\DEMO_ENERGY\DEMO_ENERGY.isasln

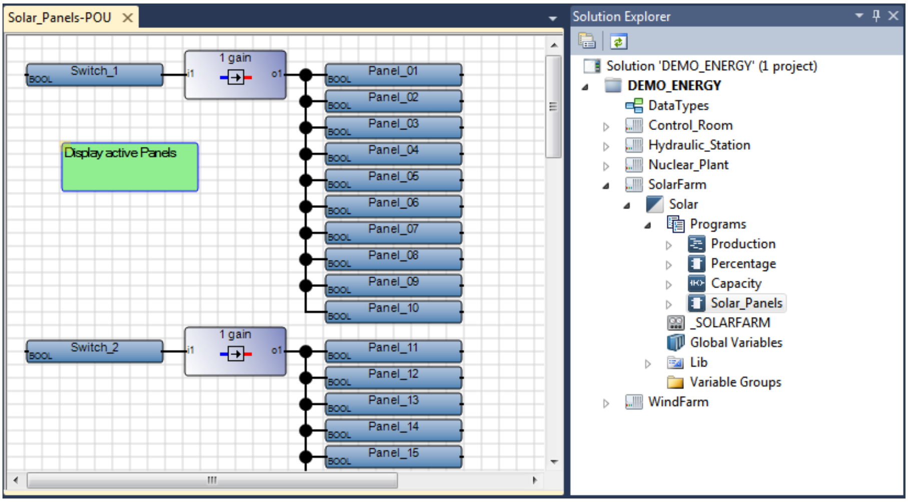



The DEMO_ENERGY project is displayed.

**2.** Review the application components.

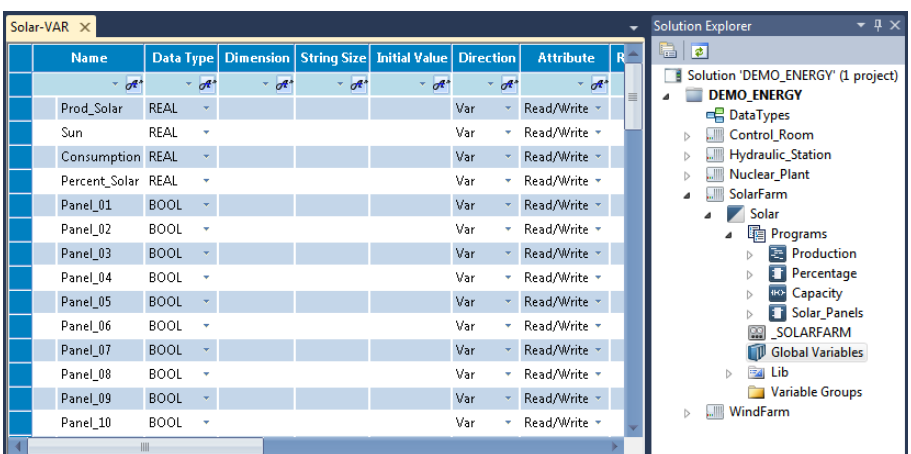
**a)** From the View menu, click **Solution Explorer**.

**b)** To view available programs, expand the project, device, resource, and program elements, then view the programs by double-clicking the required program instance.




Opened programs are displayed in the language container.

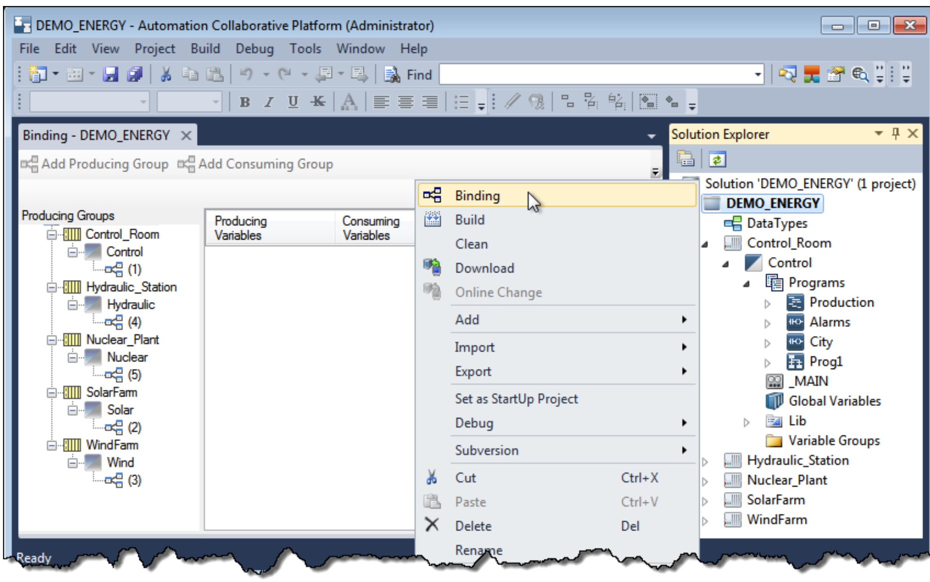
**c)** To view the dictionary variables, in the Solution Explorer double-click **Global Variables**.




The dictionary is displayed in the workspace. You can add, edit, and remove variables. You can sort and filter the variables displayed, as well as arrange the columns to display.

3. Configure bindings for variables located in different resources.

   a) To configure directional links between variables located in different resources, from the Solution Explorer, right-click the project, resource, or device, and then click **Binding**.

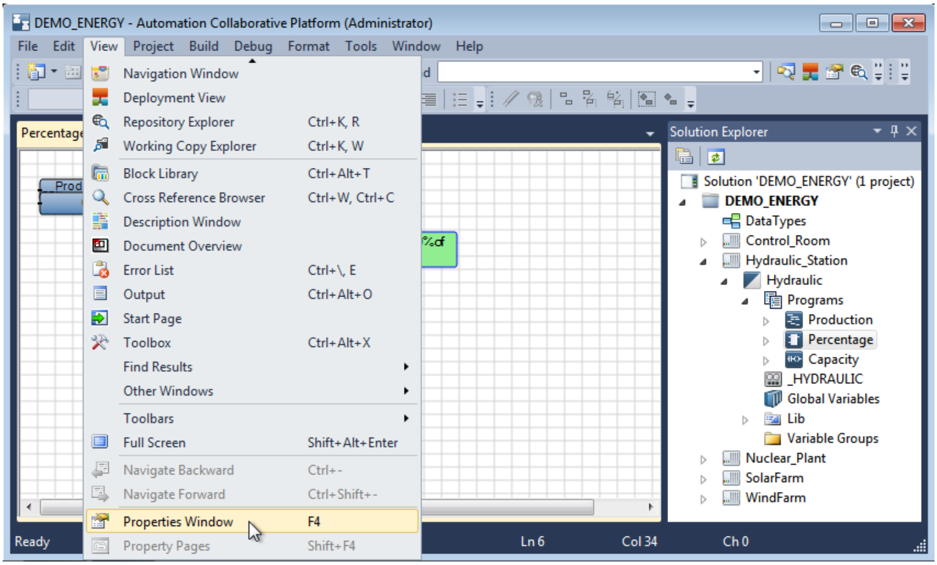



   b) From the **Bindings View**, in the *Producing Groups* column, expand the resource node, then view the binding variables by clicking .

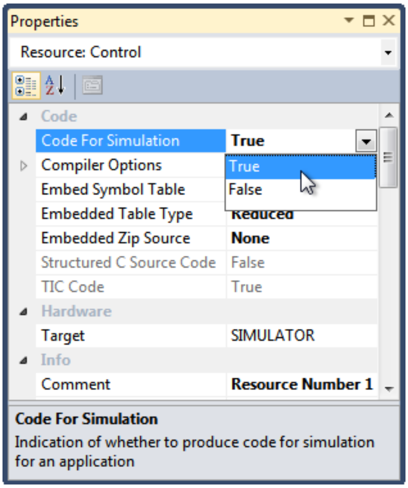
   The variables displayed in the *Producing Variables* column are bound to the variables displayed in the *Consuming Variables* column. You can edit, add, and delete the groups of producing and consuming variables.

4. For each resource, program, and target, set the properties for debugging.

   a) From the View menu, click **Properties Window**.




   b) In the Solution Explorer, select the individual resources, then from the Properties window, set *Code For Simulation* to **True**.




**c)** In the Solution Explorer, select individual programs, then in the Properties window, make the following modifications for each programming language:

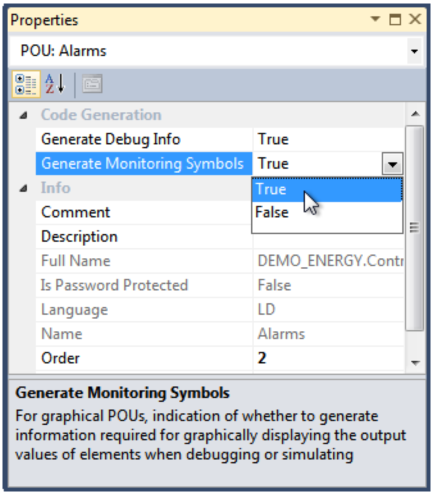
- For ST programs, set *Generate Debug Info* to **True**.

- For LD programs, set *Generate Debug Info* and *Generate Monitoring Symbols* to **True**.

- For FBD programs, set *Generate Debug Info* and *Generate Monitoring Symbols* to **True**.

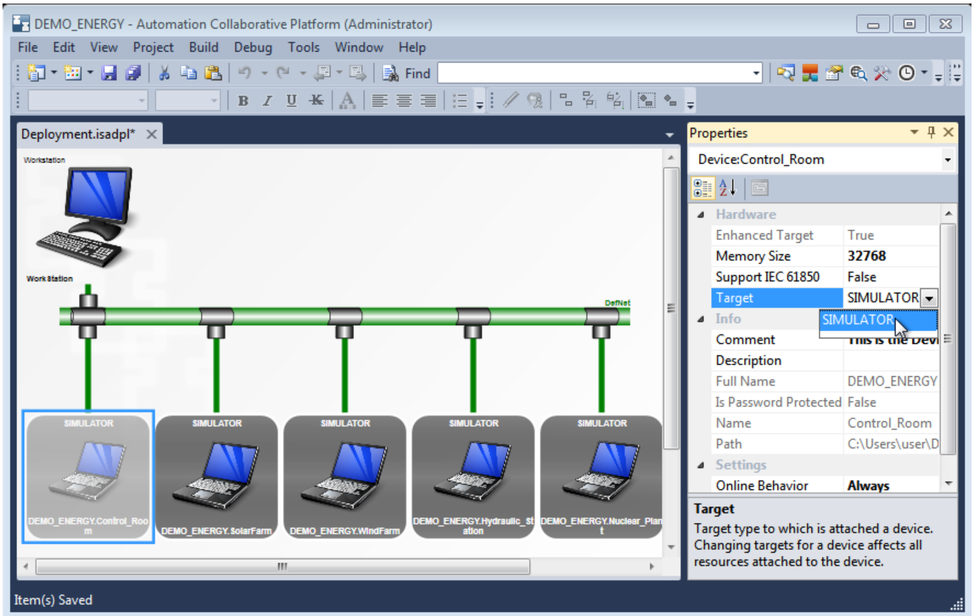
- For SAMA programs, set *Generate Monitoring Symbols* to **True**.

- For IEC 61499 programs, set *Generate Debug Info* and *Generate Monitoring Symbols* to **True**.

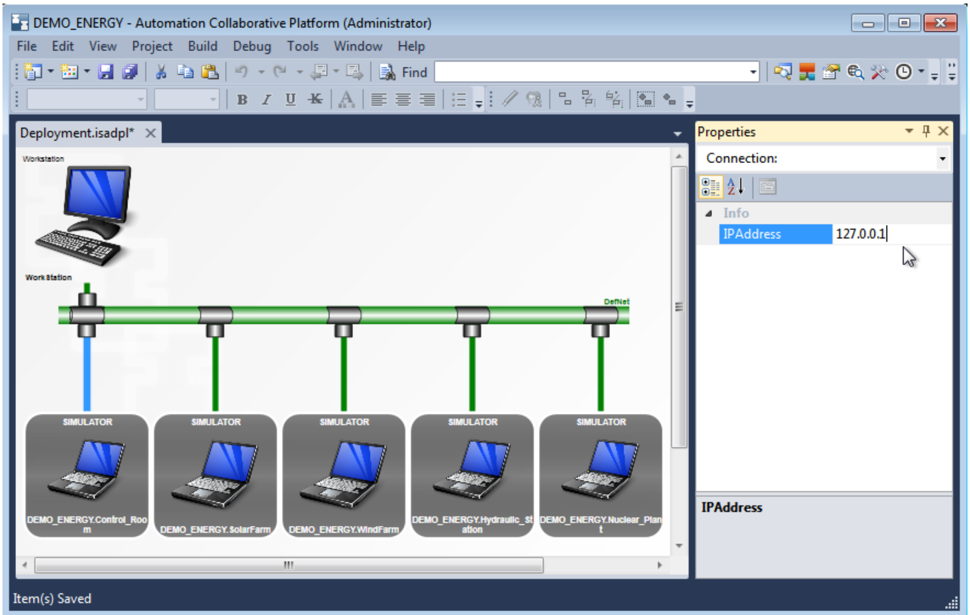



**d)** From the View menu, click **Deployment View**, and then make the following modifications to the properties:

- In the Deployment view, select the target, then in the Properties window, for the *Target* property, select the required target type from the drop-down combo box.

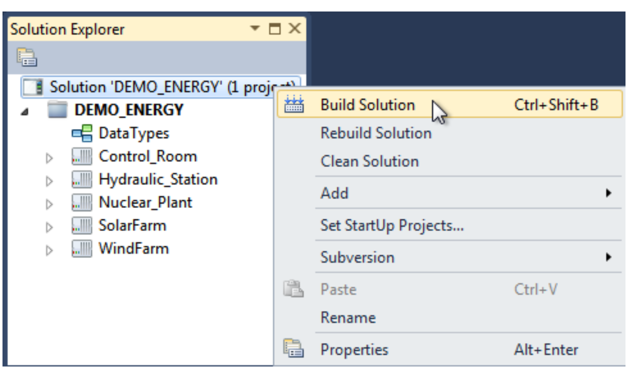



- In the Deployment view, select the connection between the target and the network, then in the Properties window, in the *IPAddress* property, type the required IP address.

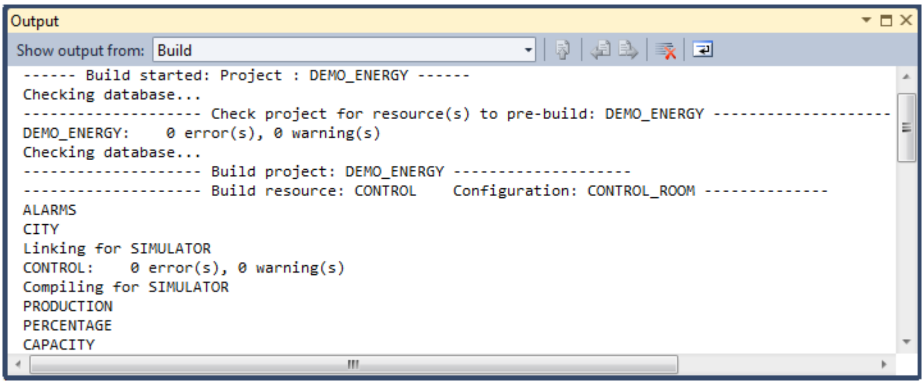



**5.** Build the solution, then view any generated errors, warnings, and messages.

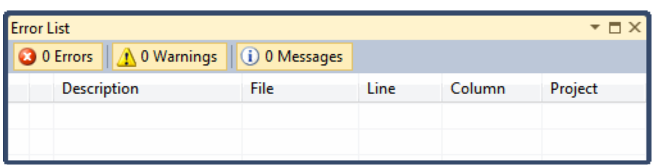
**a)** In the Solution Explorer, right-click the solution element, then click **Build Solution**.




**b)** To view the build information, from the View menu, click **Output**.




**c)** To view the errors, warnings, and messages generated during the build, from the View menu, click **Error List**.

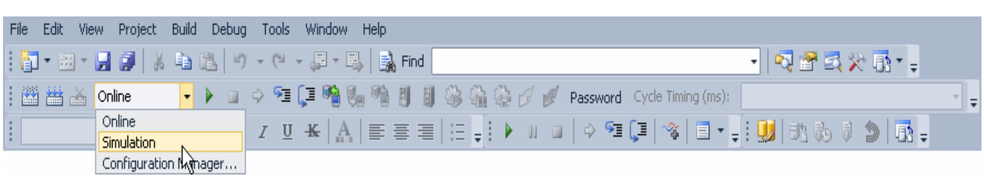



You can choose to display errors, warnings, or messages in the Error List. You can also sort the list of errors, warnings, and messages displayed.

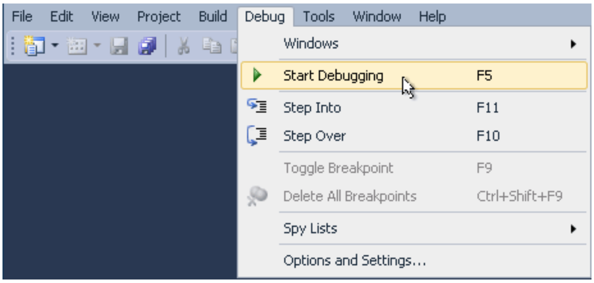
**6.** Debug the project.

You can simulate the running of an application without downloading code onto your target platform. However, when running an application online, you must download the project code onto the target before debugging.

**a)** In the Target Execution toolbar, from the Solution Configurations drop-down combo-box, select **Simulation**.



**b)** To begin the debugging process, from the Debug menu, click **Start Debugging**.

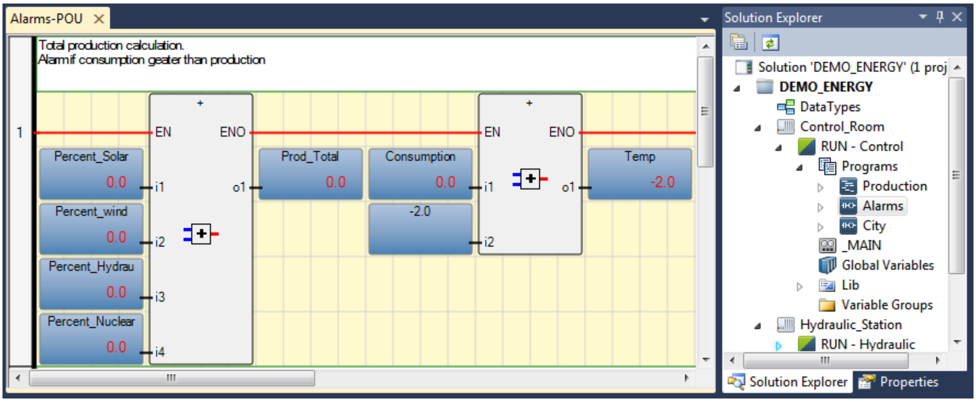



You can monitor the progress of the simulation using the Output window.

**7.** While in debug mode, view the programs and dictionary variables.

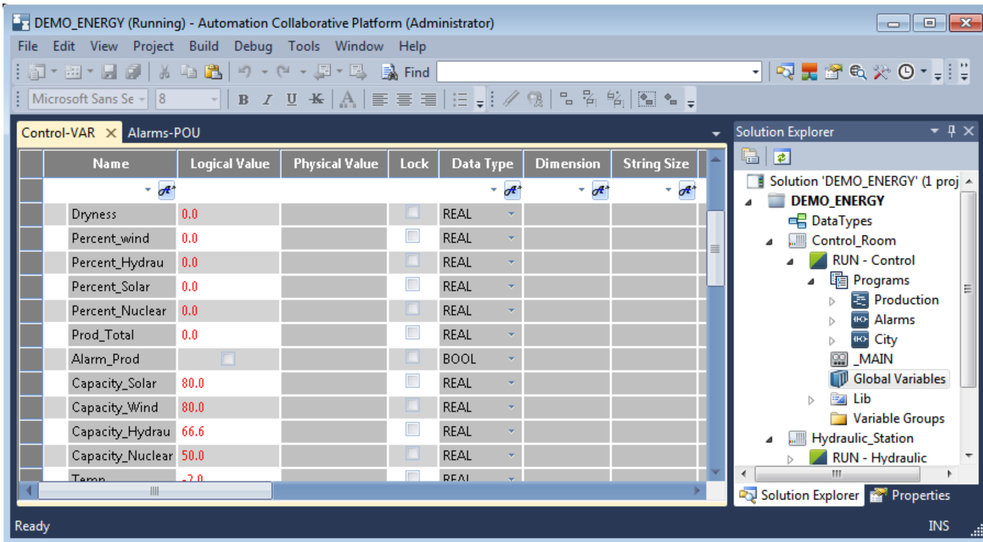
**a)** From the Solution Explorer, view the individual programs by double-clicking the required program instance.

The program is displayed in the language container. Boolean variables are displayed using color: red when True and blue when False. Numerical and textual values are displayed in red.




**b)** From the Solution Explorer, view the dictionary variables by double-clicking **Global Variables** or **Local Variables**.

The dictionary is displayed in the workspace. Note that the logical and physical values are displayed in red.




**8.** To stop the debugging process, from the Debug menu, click **Stop Debugging**.

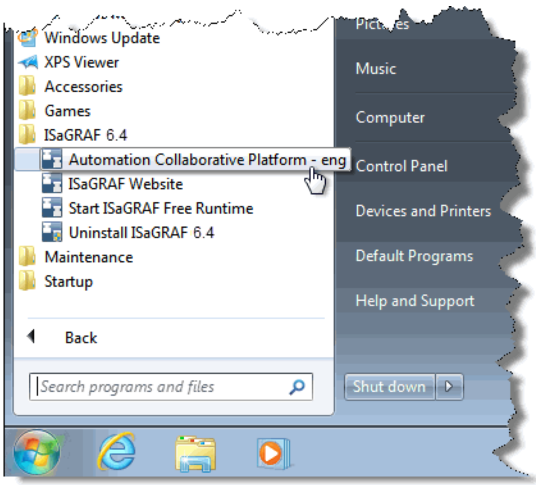
# Starting with a Basic Application

This section is a guideline to creating a basic solution and project by following the required steps. The project detailed in this section uses the ISaFREE_TPL template consisting of one resource in one device.

**To start a new project having one resource in one device**

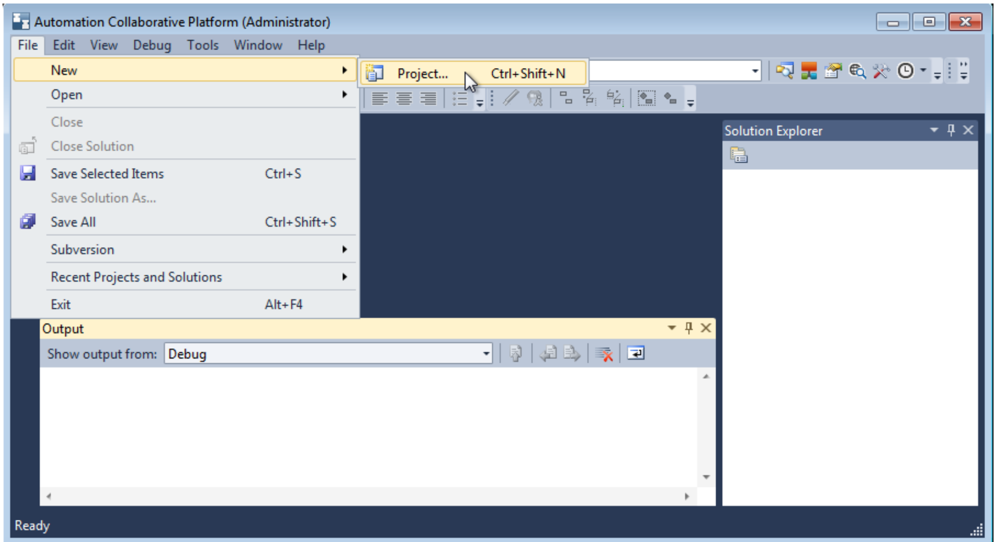
1. To launch the **ACP** and create a new solution, perform the following:

    a) From the Start menu, point to **All Programs**, click **ISaGRAF 6.4**, and then click **Automation Collaborative Platform**.

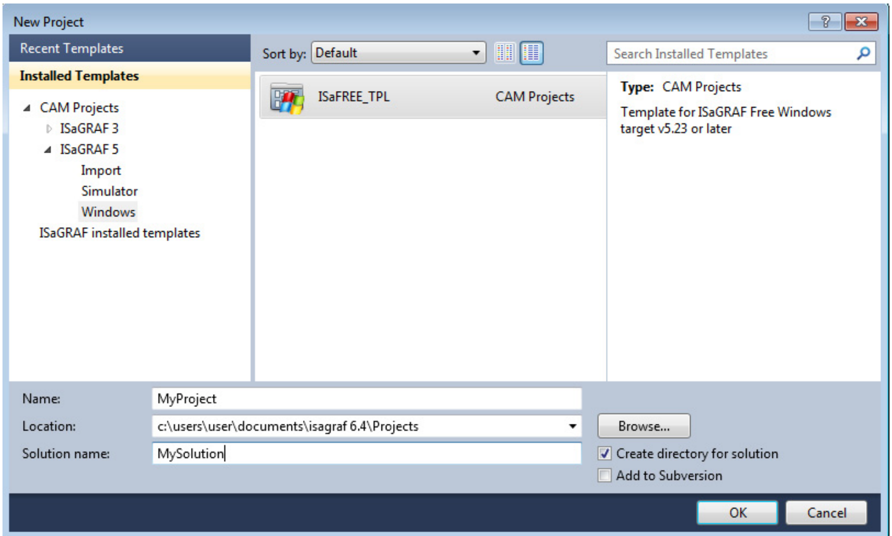



    The Workbench is displayed.

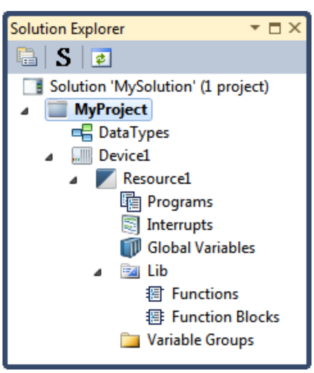
    b) From the File menu, point to **New**, and then click **Project...**




**c)** In the New Project dialog box, expand the **ISaGRAF 5** projects node, select the Windows template section, then click the **ISaFREE_TPL** template. You then select *Create directory for solution* and specify a solution name. You must also specify a name and save location for the project, then click OK.

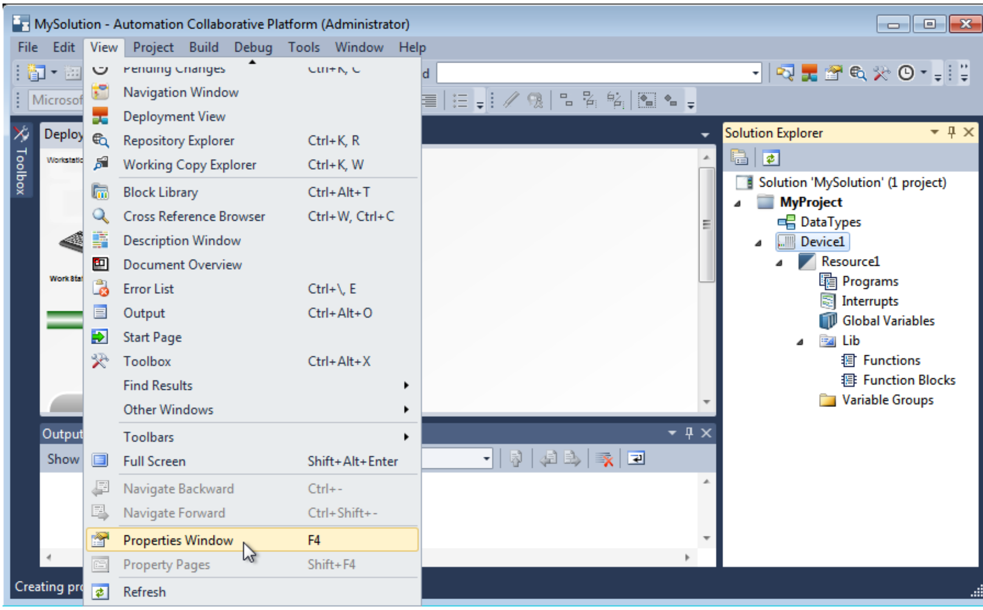



2. In the Solution Explorer, expand the project elements and note the device and resource created from the **ISaFREE_TPL** template.

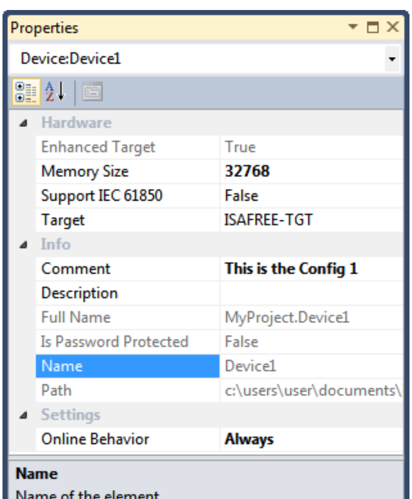



3. Specify the properties for the device and the resource.

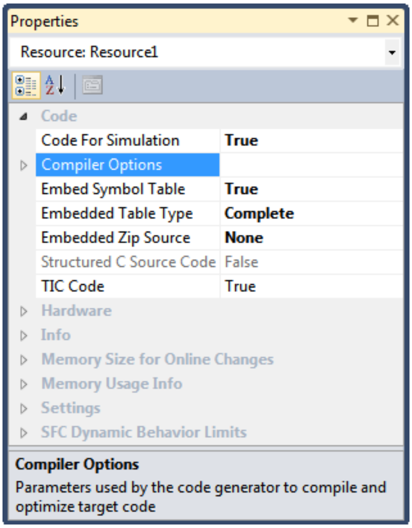
   a) In the Solution Explorer, select the device, then from the View menu, click **Properties Window**.




**b)** In the Properties window, note the definitions for the memory size, target type, comment, description, password protection, and name.

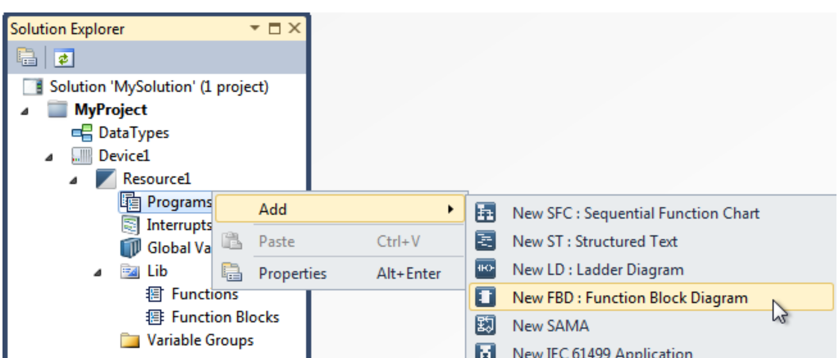


**c)** In the Solution Explorer, select the resource, then in the Properties window, note the many properties for the code, hardware, info (resource), memory size for online changes, memory usage info, settings, and SFC dynamic behavior limits.

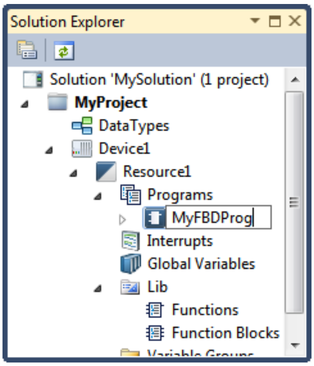



**4.** In the Solution Explorer, add a program and define the program name.

**a)** Right-click the programs element, point to **Add**, and then click the required programming language.

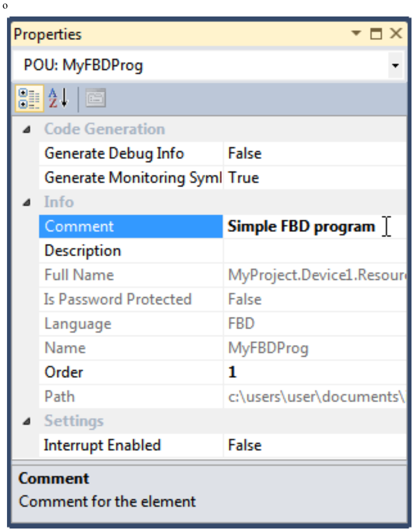


**b)** Right-click the added program, click **Rename**, and then type the desired name in the space provided.

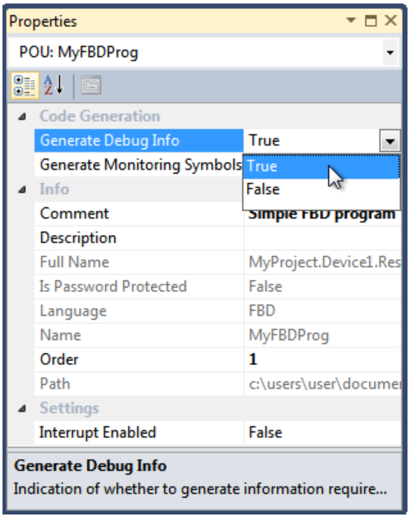



**5.** In the Properties window, define the properties for the program.

**a)** For *Comment*, type a comment in the space provided.



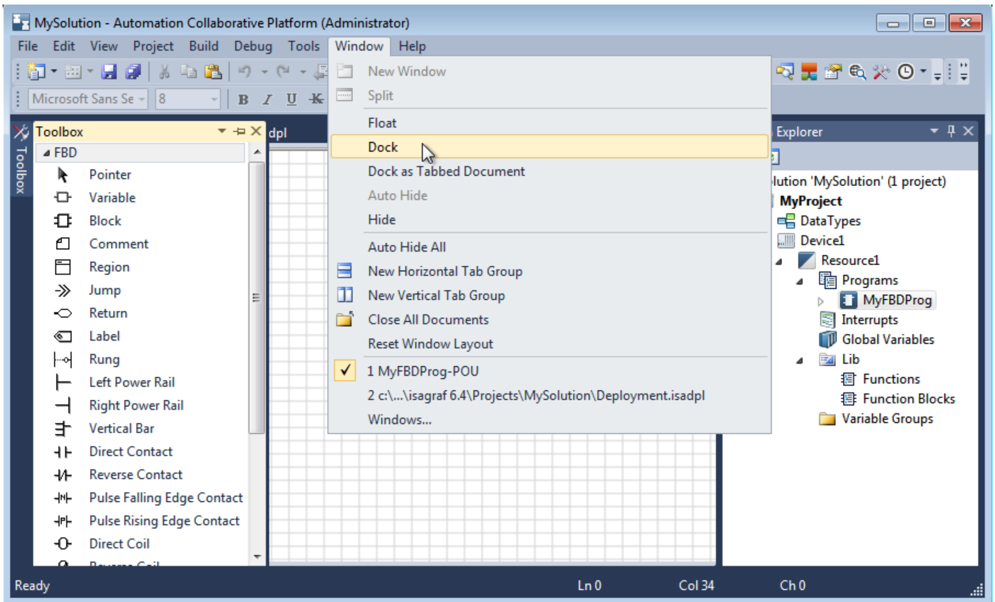

**b)** Set *Generate Monitoring Symbols* and *Generate Debug Info* to **True**.




**6.** In the language container, add elements to the program.

**a)** From the Solution Explorer, double-click the program instance. The program is displayed in the language container. By default, the Toolbox is auto-hidden as a tab on the left edge of the Integrated Development Environment (IDE).

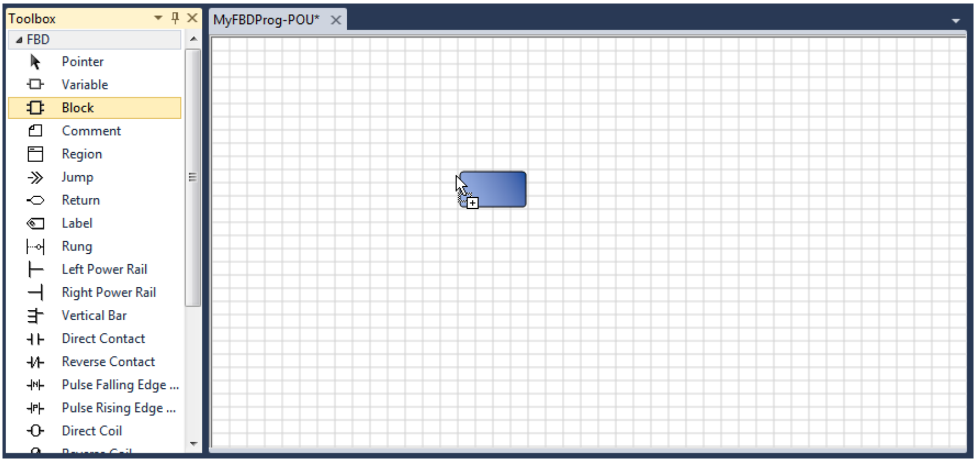
**b)** To display the Toolbox, click the tab so the Toolbox slides into view. From the Window menu, click **Dock**.




The Toolbox window is docked in the IDE.

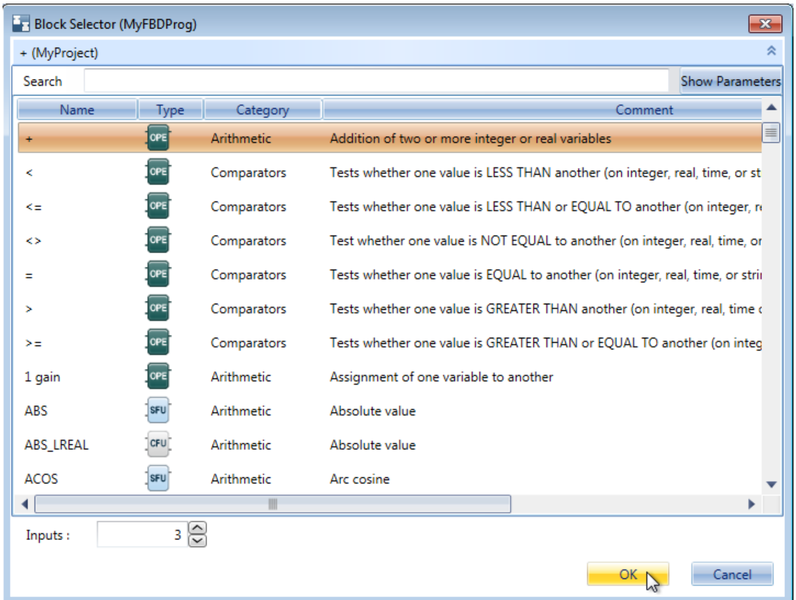
**c)** Add a block in the language container.

i) From the Toolbox, drag the **Block** element into the language container.




The Block Selector is displayed.

ii) In the *Block Selector* list, select the required POU, specify the instance and the number of inputs (when applicable), then click **OK**.



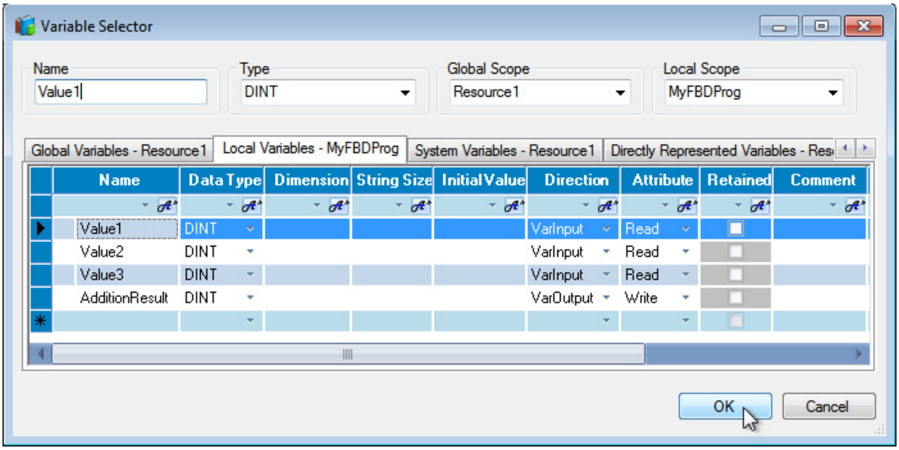

The block is displayed in the language container.

**d)** Add a variable in the language container.

i) From the Toolbox, drag the **Variable** element into the language container.

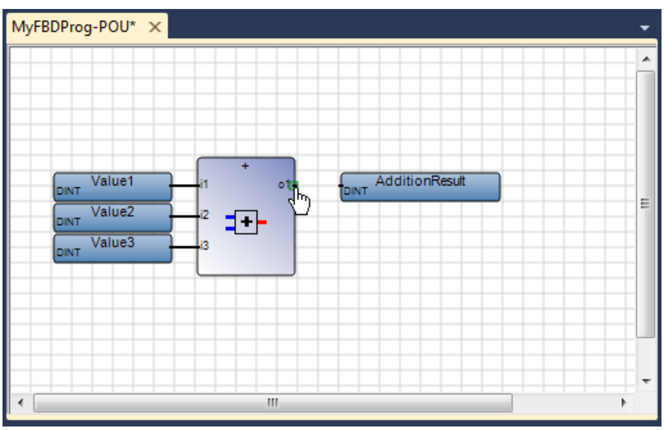
The Variable Selector is displayed, with tabs containing lists for *Global variables*, *Local variables*, *System variables, Directly Represented Variables*, and *Defined Words*.

ii) In the *Local Variables* list, enter the variable name, data type, and other required information into the cells provided, then click **OK**.



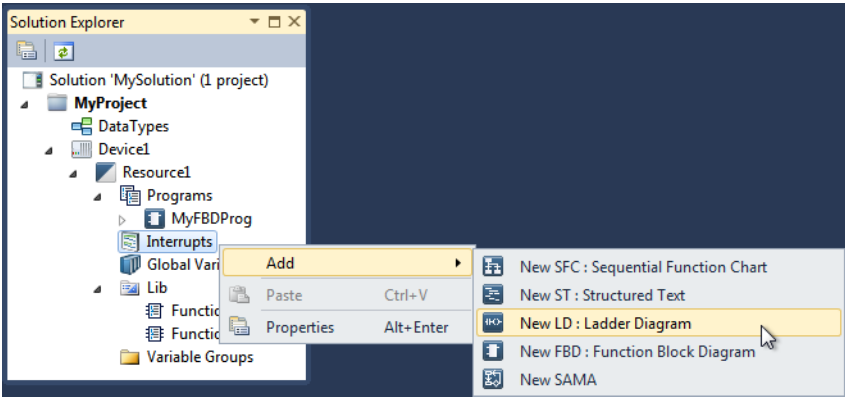

The variable is displayed in the language container.

e) Draw links from the variable to another variable, a block input, or a block output.

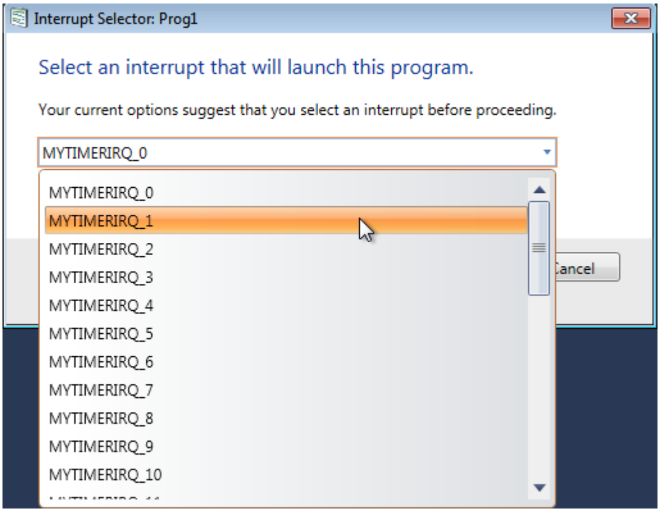



7. In the Solution Explorer, configure an interrupt to control the moment of execution for cyclic programs (SFC, ST, LD, FBD, or SAMA) and modify the properties.

a) Right-click the interrupts element, point to **Add**, and then click the desired programming language.

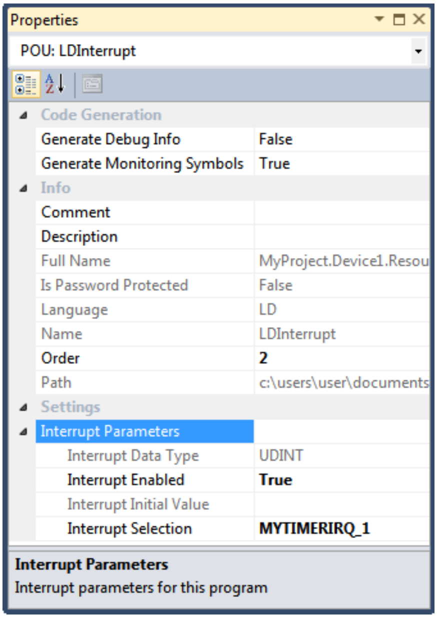



**b)** From the Interrupt Selector dialog box, select the required interrupt, and then click **OK**.




**c)** Right-click the added interrupt program, click **Rename**, and then type the desired name in the space provided.

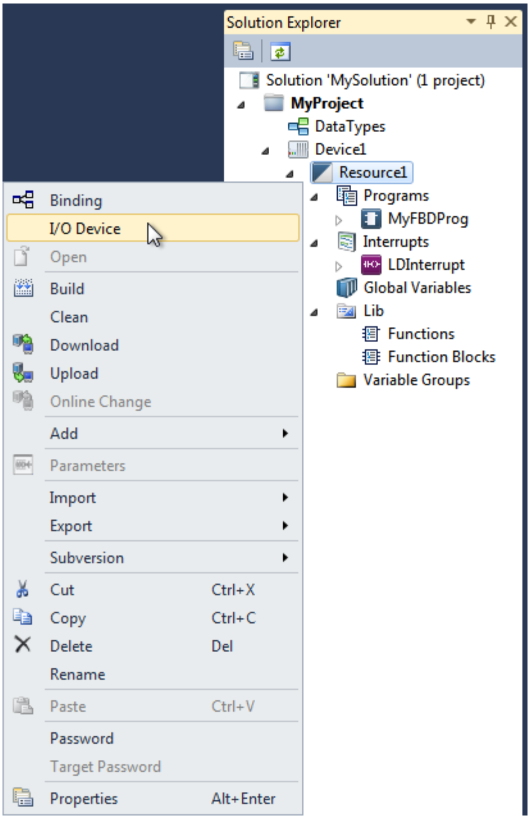
**d)** In the Properties window, note the definitions for the *Interrupt Parameters*.




**e)** From the Solution Explorer, double-click the interrupt instance. In the language container, add elements to the interrupt program.

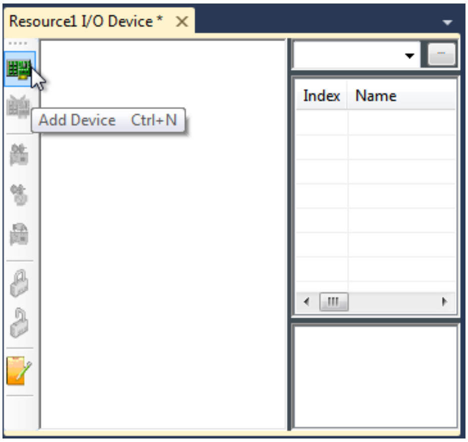
**8.** To link variables to the channels of I/O devices existing on a target system, perform the following:

**a)** From the Solution Explorer, right-click the resource, and then click **I/O Device**.



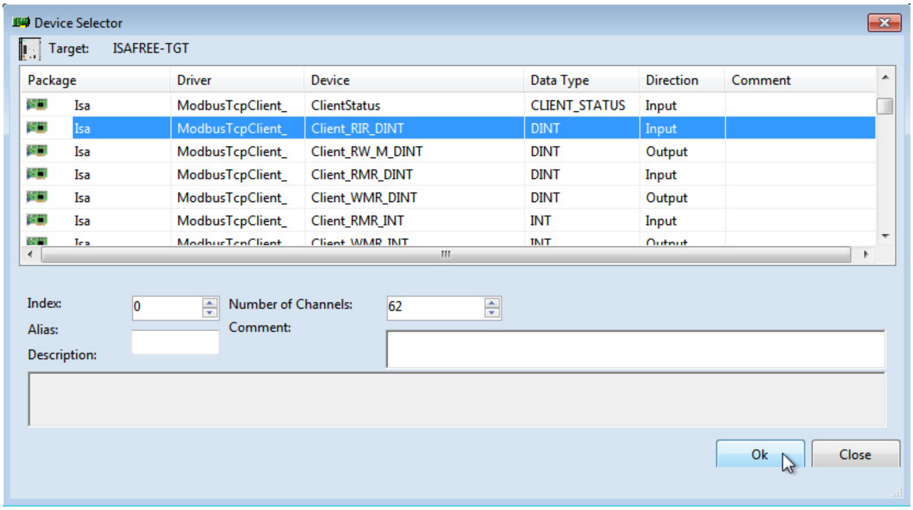

The I/O Wiring view is displayed in the workspace.

**b)** From the I/O Wiring toolbar, add an I/O Device.

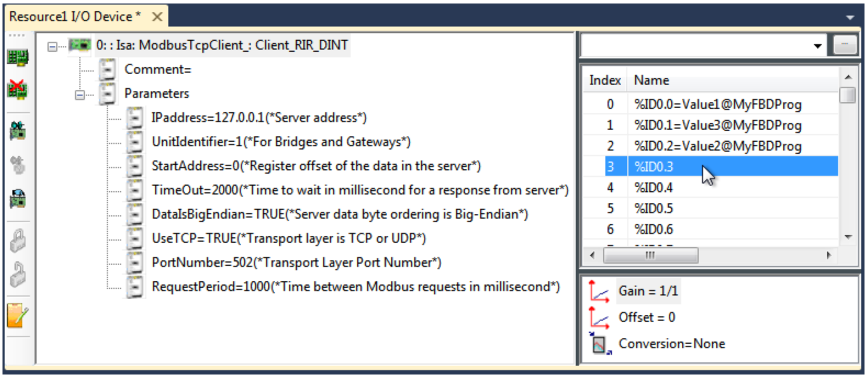



The Device Selector dialog box is displayed.

**c)** From the Device Selector dialog box, select the required I/O device, then click **OK**.

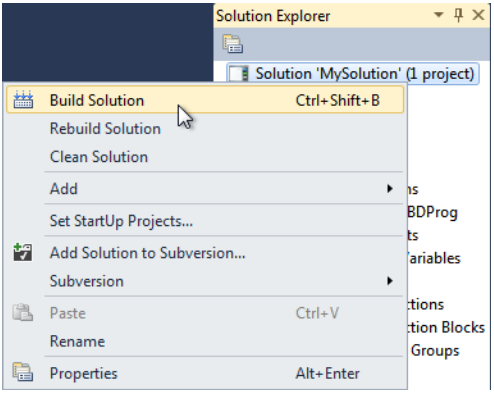



**d)** Double-click an empty channel and from the Variable Selector select the variable to wire, then click **OK**.

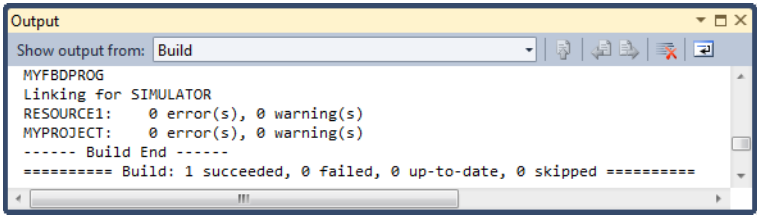



**9.** From the Solution Explorer, build the solution, then view any generated errors, warnings, and messages.

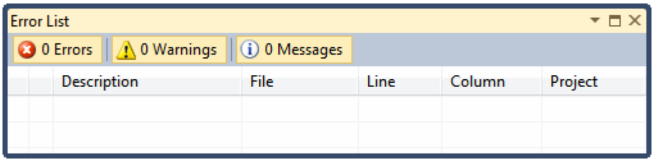
**a)** Right-click the solution element, then click **Build Solution**.




**b)** To view the build information, from the View menu, click **Output**

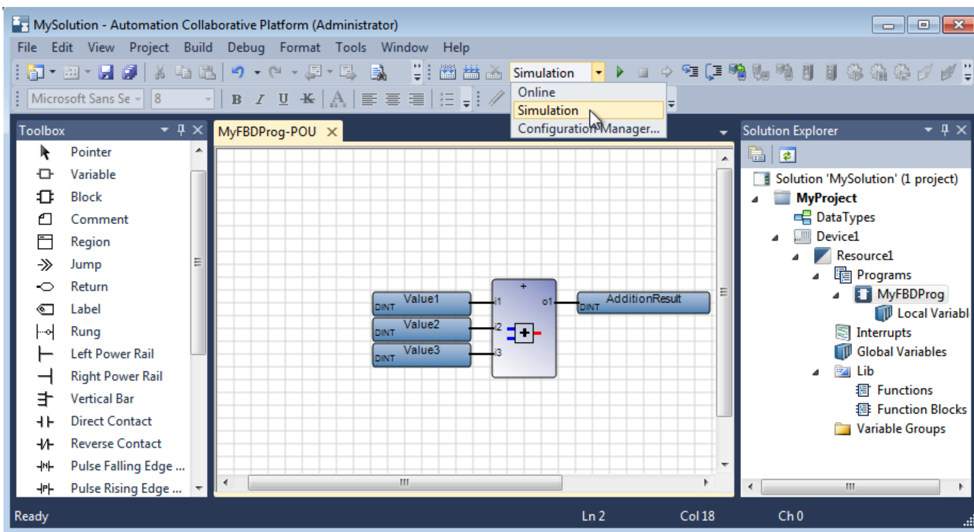


c) To view the errors, warnings, and messages generated during the build, from the View menu, click **Error List**.

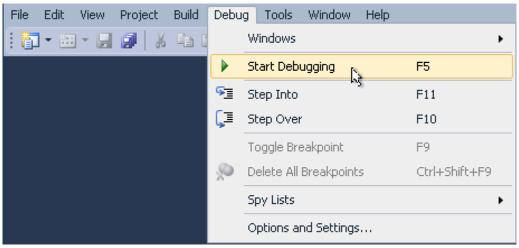



**10.** Begin the debugging process, then view the programs and dictionary variables.

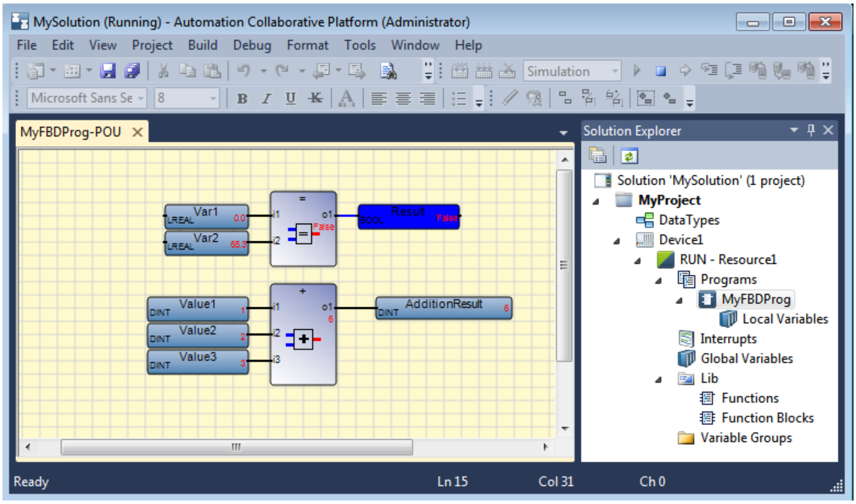
a) From the Target Execution toolbar, in the Solution Configurations drop-down combo-box, select **Simulation**.




b) From the Debug menu, click **Start Debugging**.

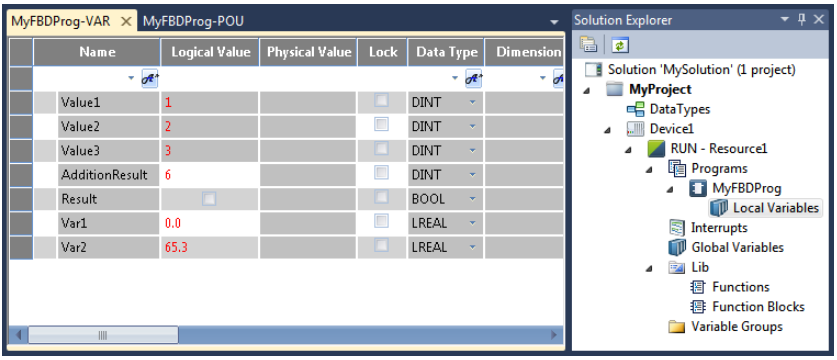



**c)** From the Solution Explorer, view the program by double-clicking the program element.




Note the debugging information regarding boolean variables is displayed using color: red when True and blue when False. Numerical and textual values are displayed in red.

**d)** From the Solution Explorer, view the dictionary variables by double-clicking **Local Variables** for the required program.




Note the logical values are displayed in red. Physical values are only displayed when running online.

11. To stop the debugging process, from the Debug menu, click **Stop Debugging**.

# Importing an Existing Application

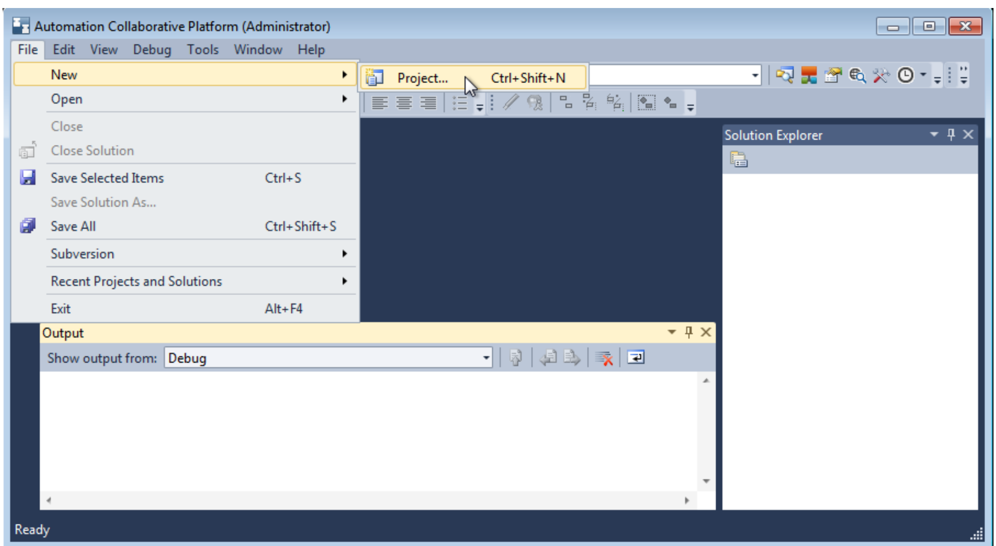
When importing applications created with **ISaGRAF** 5, some features of your projects are converted for use in the current environment.

**To import an ISaGRAF 5 project into ISaGRAF 6**

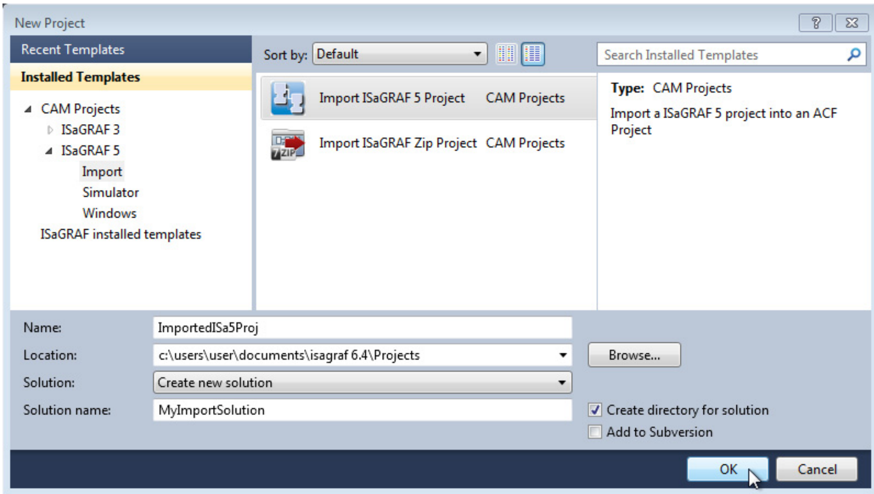
When importing **ISaGRAF 5** projects into **ISaGRAF 6**, the targets associated with the **ISaGRAF 5** projects must be supported by **ISaGRAF 6**.

**1.** Import the **ISaGRAF 5** project into **ISaGRAF 6**.

**a)** From the File menu, point to **New**, then click **Project**.

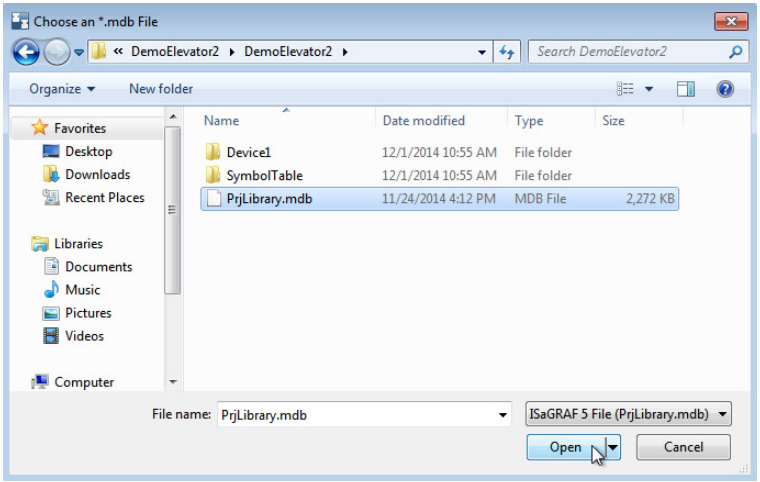


**b)** From the New Project dialog box, expand the **ISaGRAF 5** projects node, select the Import template section, and click **Import ISaGRAF 5 Project**. You then enter the required information in the fields provided and click **OK**.




**c)** From the Choose an .mdb File dialog box, select the **ISaGRAF 5** project file, then click **Open**.

You may encounter a message asking if you want to update the database to the current version. To continue the importation process, click OK.

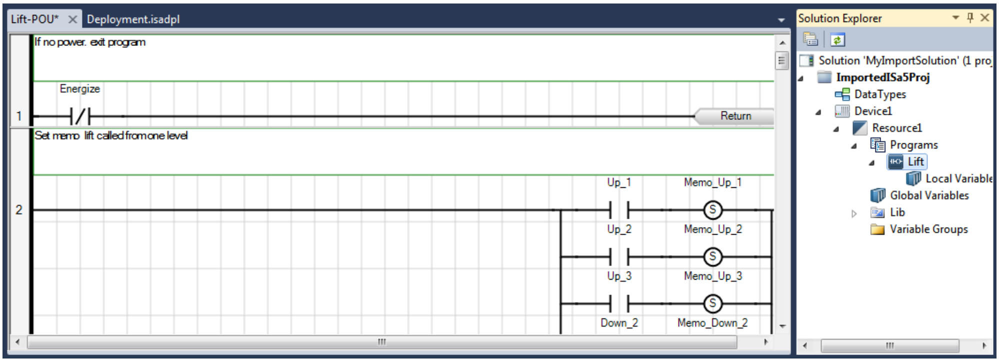



The **ISaGRAF 5** project is imported.

2. View the project in **ISaGRAF 6**.

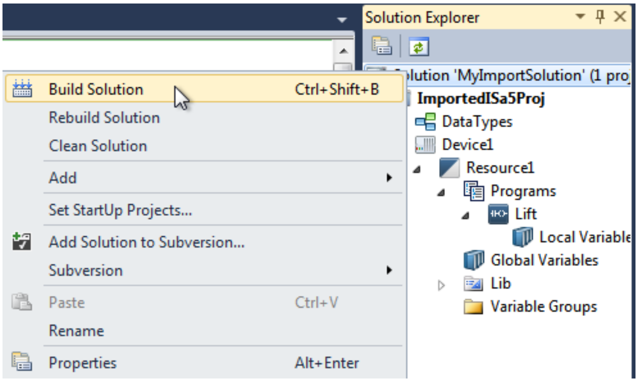
   a) In the Solution Explorer, expand the project, device, resource, and program elements, then view the individual programs by double-clicking the required program instance.

   Opened programs are displayed in the language container.

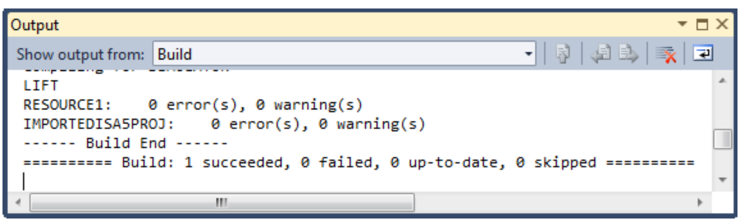


3. Build the solution, then view any generated errors, warnings, and messages.

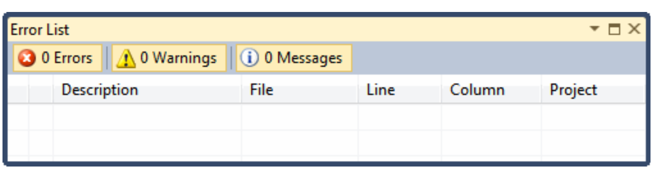
   a) In the Solution Explorer, right-click the solution element, and then click **Build Solution**.




   b) To view the build information, from the View menu, click **Output**.

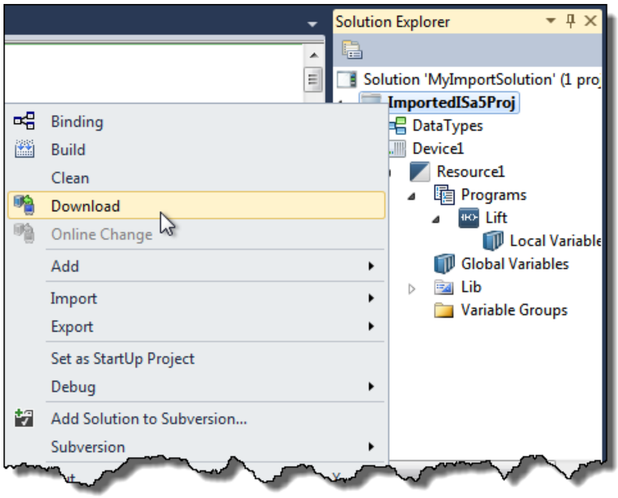



   c) To view errors, warnings, and messages generated during build, from the View menu, click **Error List**.



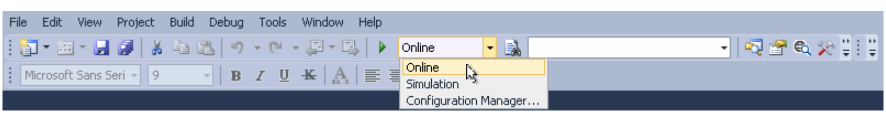

4. Debug the project.

**a)** To download the resource code to the target, in the Solution Explorer, right-click the project element, then click **Download**.

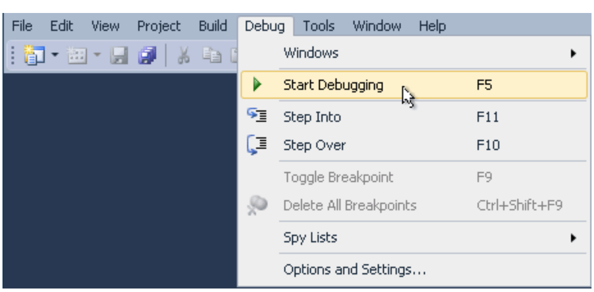



You can monitor the progress of the download operation using the Output window.

**b)** In the Target Execution toolbar, from the drop-down combo-box, select **Online**.




**c)** To begin the debugging process, from the Debug menu, click **Start Debugging**.



5. To stop the debugging process, from the Debug menu, click **Stop Debugging**.

# Version Source Control

You manage the changing versions of **ISaGRAF** elements including solutions, projects, devices, resources, POUs (other than IEC 61499 and SFC with child), and ISaVIEW screens by saving them to a repository. Saving these elements to a repository enables multiple users to work on the same solutions and project elements at the same time as well as retrieve older versions of elements at a later time.

This product includes software developed by * CollabNet (http://www.Collab.Net/) based on the Subversion AnkhSVN source control plug-in for Visual Studio.

When creating a solution, you can choose to commit, i.e., save, the solution into a repository. You should commit changes to elements such as projects, devices, resources, programs, deployment views, and ISaVIEW screens from the pending changes feature (while retaining all items selected). When committing elements to a repository, a default repository is installed in the following location:

%USERPROFILE%\My Documents\ISaGRAF 6.x\Repository

After committing a solution to a repository, you can choose to lock the solution or sub-elements for exclusive access when making changes. Upon making changes to previously committed elements, these elements become locked for your exclusive access. Afterwards, you can commit changes made to these elements into the repository. When committing solutions, all sub-elements are also committed to the repository.

When deleting, renaming, and adding elements in the Solution Explorer, you need to have locked the parent of that element.

| Element | Parent |
|---|---|
| POU | Resource |
| Resource | Device |
| Device | Project |
| ISaView Screen | Project |

When retrieving, i.e., updating, a solution from the repository, you can update the solution to the latest version. For reference purposes, you can access specific revisions.

When using version source control, the following best practices are recommended:

- When committing elements, save all changes, then commit all modified files at once.
- When locking elements, avoid stealing locks unless absolutely required. For instance, when a user having the elements checked out is no longer available. Modifications performed on such elements by the original lock holder are no longer available for committing.
- When getting the latest version, perform the operation from the solution.
- When getting a specific version, place the retrieved files in an empty folder; avoid placing the retrieved files in the current existing working copy.
- When canceling a modification, revert all modified files at once, wait for the completion of the rollback operation, then unlock all files having no more modifications.
- When using libraries, always use an absolute path for binding a library. All other users (different computer or folder) retrieving the library from the repository must also use the same absolute path and get all library solutions before project solutions. Place libraries and projects in different solutions since libraries need to be loaded before projects can access their elements.

The version control status of elements is indicated in the Solution Explorer:

| | |
|---|---|
|  | Unmodified. The solution is unmodified from the current repository version. |
|  | Modified but not saved on local disk. The solution is modified from the current repository version. |
|  | Modified and saved on local disk. The solution is modified from the current repository version. |
|  | New element (solution, project, device, resource, POU, or ISaVIEW screen) having a reserved location in the repository. The element has been added to the Solution Explorer and is in queue for committing to the repository. |
|  | New element. The element (solution, project, device, resource, POU, or ISaVIEW screen) has been added and is in queue for committing to the repository. |

 Locked and unmodified. The element (project, device, resource, POU, or ISaVIEW screen) is locked for exclusive use from the repository and has not been modified; The element is not available to others.

 Locked and modified. The element (project, device, resource, POU, or ISaVIEW screen) is locked for exclusive use from the repository and has been modified; The element is not available to others.

 Read-only. The element (project, device, resource, POU, or ISaVIEW screen) may be available for modification from the repository if no other has locked it for exclusive access. When committing an element, you can choose to keep it locked.

 Renamed. The element (solution, project, device, resource, POU, or ISaVIEW screen) has been renamed from the current repository version.

 Deleted. The element (solution, project, device, resource, POU, or ISaVIEW screen) has been deleted from the local working copy.

 Missing. The element (solution, project, device, resource, POU, or ISaVIEW screen) is missing from the repository.

 Conflicted. A conflict occurred between the element (solution, project, device, resource, POU, or ISaVIEW screen) in the working copy and the repository while performing a get specific version, performing a get latest version, or committing to the repository.

**See also**

Defining a Repository
Using the Repository Explorer
Using the Working Copy Explorer
Committing Pending Changes
Locking and Unlocking Elements
Getting Versions of Elements
Reverting Versions of Elements
Creating a Working Copy from a Repository
Viewing the History of Elements
Comparing Element Versions
Canceling Local Modifications

## Using the Repository Explorer

The Repository Explorer enables viewing the files for the elements committed to a local repository. When committing, the repository includes a folder structure containing the respective source control files for individual elements ranging from the solution to the resources.

| Element Type | Retained in Repository |
|---|---|
| Solution | Solution and deployment files |
| Project | Project and target files |
| Device | Device file |
| Resource | Resource file |
| Programs | Program properties, program coding, and language editor properties files |

You set up repositories within a defined URL

In the Repository Explorer, you can perform management tasks:

- Add a URL
- Remove a URL
- Refresh the contents of repositories
- Copy directories or files to another location within a URL
- Move directories or files to another location within a URL. You can only move items not locked by any user.
- Create directories
- Delete directories or files. You can only move items not locked by any user.

**To access the Repository Explorer**

The Repository Explorer displays the contents of the repository.

- From the View menu, choose Repository Explorer.

### To add a URL in the explorer

You can add multiple URLs for viewing in the Repository Explorer.

1. In the Repository Explorer, select Local Repositories in the tree.
2. Click  on the toolbar
3. In the Browse Repository dialog box, enter the URL or select one from the available URLs.

### To remove a URL from the explorer

1. In the Repository Explorer, select Local Repositories in the tree.
2. To remove a URL, click  on the toolbar.

### To refresh the contents of a URL or repository

You can refresh the contents of a selected URL or repository.

- In the Repository Explorer, select the URL or repository for which to refresh the displayed contents, then click  on the toolbar.

### To copy a directory or file

You can copy a directory or file to another location within a URL.

- In the Repository Explorer, select the folder or file to copy, then click  on the toolbar.

### To move a directory or file

You can move a directory or file to another location within a URL.

- In the Repository Explorer, select the folder or file to copy, then click  on the toolbar.

**To create a directory**

You can create directories within a URL.

- In the Repository Explorer, select the location in which to create the directory, then click  on the toolbar.

**To delete a directory or file**

You can only delete files and directories while these are not locked by any users.

- In the Repository Explorer, select the file or directory to delete, then click  on the toolbar.

# Using the Working Copy Explorer

The Working Copy Explorer enables viewing the directories and files of local working copies from repositories. The current solution is displayed at the topmost of the explorer window. You can also add new roots, i.e., mappings, to the Working Copy Explorer pointing to items such as other solutions under source control. These added roots are temporary and are automatically removed upon closing the Working Copy Explorer.

When using the Working Copy Explorer, you can perform the following tasks:

- Add a new root
- Update to latest version (recursive)
- Show the history viewer for a selected item
- Compare a file to another version
- Open or select a file in another environment
- Export a directory or file
- Delete directories and files

**To access the Working Copy Explorer**

The Working Copy Explorer displays the contents of the currently opened project in the local repository.

- From the View menu, choose Working Copy Explorer.

**To add a new root**

1. In the Working Copy Explorer, click  on the toolbar.
2. In the Browse Working Copy dialog box, enter the path or browse to locate the root.

### To update to the latest version (recursive)

When updating versions of solutions, the current version must be latest committed version. You can update the current solution to the latest version recursively.

- In the Working Copy Explorer, select the current solution, then click on the toolbar.

### To show the History viewer for a selected item

- In the Working Copy Explorer, select the current solution, then click on the toolbar.

### To compare a file to another version

You can compare the current solution to another version in a side-by-side editor. Available versions for comparison include original, latest, committed, previous, and specific revisions.

- In the Working Copy Explorer, select the current solution, then click on the toolbar and choose the version for comparison from the drop-down list.

### To open or select a file in another environment

You can choose to open a file or select a file in an environment other than the Working Copy Explorer. For opening files, available options include: Visual Studio, default application, and text. You can also open the folder containing the file. For selecting files, the available options include: Solution Explorer, Working Copy Explorer, and Repository Explorer.

- In the Working Copy Explorer, locate and select the file from the current solution, then click on the toolbar and choose the required option from the drop-down list.

### To export a directory or file

You can export a directory or file to another folder. You can choose to make exports non-recursive.

1. In the Working Copy Explorer, select the directory or file to export, then click  on the toolbar.

2. In the Export dialog box, specify the version of the directory or file to export, then locate or specify the path of the destination. To export only the selected directory or file, select Non-recursive.

**To delete a directory or file**

- In the Working Copy Explorer, select the directory or file to delete, then click  on the toolbar.

# Defining a Repository

When using version source control for managing changes in **ISaGRAF** element versions, you need a repository. The repository contains all necessary files storing the changes for the elements. When opening the Workbench for the first time, you can define a repository in a location other than the default installation:

%USERPROFILE%\My Documents\ISaGRAF 6.x\Repository

You view the contents of a repository from the Repository Explorer. You can also view the contents of a local solution from the Working Copy Explorer.

**To define a repository in a location other than the default installation**

You can only define a repository at a different location when opening the Workbench for the first time; changing the location of a repository at a later time causes link issues with elements saved in a previous repository. The Repository Explorer displays all local repositories and their contents.

1. From Windows Explorer, copy the complete Repository directory from the default installation and paste this directory in the required location.
2. From the View menu, point to **Repository Explorer**.
3. From the Repository toolbar, click , then specify the location or select one from the list of available repositories in the Browse Repository dialog box.

### See also

Using the Repository Explorer
Version Source Control

# Committing Pending Changes

You can commit pending changes for **ISaGRAF** elements such as solutions, projects, devices, resources, programs, deployment views, and ISaVIEW screens to a repository. Committing pending changes enables managing the changing versions of elements and referencing specific revisions. You can also commit pending changes for elements in previously committed solutions. Committing changes to elements consists of adding elements in a queue for committing, then committing to the repository.

You commit changes from the pending changes window after selecting an element in the Solution Explorer hierarchy:

- Deleting projects, devices, resources, POUs, and ISaVIEW screens
- Renaming projects, devices, resources, POUs, and ISaVIEW screens
- Modifying passwords for projects, devices, resources, and POUs
- Reordering POUs
- Cutting, copying, and pasting projects, devices, resources, and POUs

**Note:** Make sure to commit entire projects and commit changes following a few changes.

From the pending changes window, you can perform the following tasks for elements selected in the Solution Explorer.

- Commit a solution to a repository
- Commit an element to a previously committed solution
- Compare files with other versions
- Refresh the list of files pending changes

**To commit a solution to a repository**

When committing solutions, the necessary files are added to the repository.

1. From the Solution Explorer, right-click the solution to add to the repository.

2. From the contextual menu, point to **Add Solution to Subversion**.

3. In the Add to Subversion dialog box, specify the project name and the repository URL, then click **OK**. The local path is set automatically.

4. When prompted, provide a meaningful comment.

   The solution structure is queuing for commitment to the repository. The solution is displayed with a yellow plus sign and all sub-elements with a blue plus sign.

5. From the File menu, point to **Subversion**, then click **Pending Changes**.

6. From the Pending Changes window, leave all items selected, then click **Commit** in the toolbar.

The solution and all sub-elements are saved in the repository and are displayed with a lock in the Solution Explorer.

**To commit an element to a previously committed solution**

When committing an element to a previously committed solution, you save all changes, then commit the element to the repository. Elements available for adding to a project’s control structure are displayed with a blue plus sign.

1. From the Solution Explorer, select the element to commit.

2. From the File menu, point to **Subversion**, then click **Pending Changes**.

3. From the Pending Changes window, leave all items selected, then click Commit and one of the following options from the drop-down menu:

   - To commit changes while keeping locks on the files, click **Commit Changes**.
   - To commit changes while keeping the files locked for exclusive access, click **Commit Keeping Locks**.

The element is saved in the repository and is displayed with a lock in the Solution Explorer.

### See also

Version Source Control
Locking and Unlocking Elements

Comparing Element Versions

# Getting Versions of Elements

You can get, i.e., retrieve or update, different versions including the latest version, previous version, and a specific revision for solutions, projects, devices, resources, functions, function blocks, programs, and ISaVIEW screens.

When updating versions of elements, the current version must be latest committed version.

When getting versions of elements, you can perform the following tasks:

- Get the latest version for a working copy
- Get a specific version for an element

**To get the latest version for a working copy**

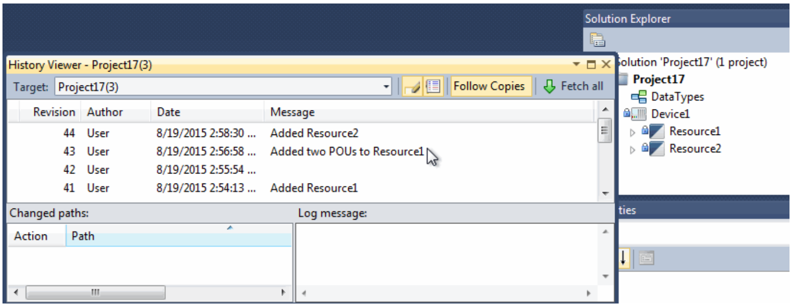
You get the latest versions for a working copy from the solution level.

1. From the Solution Explorer, right-click the solution.
2. From the contextual menu, click **Update Solution to Latest Version**.

**To get a specific version of an element**

You can get a specific version for an element from the repository for reference purposes only. When modifying previous versions, conflicts may arise for different reasons such as elements being no longer available, renamed, or deleted. When getting a specific version, avoid placing the retrieved files in the current existing working copy.

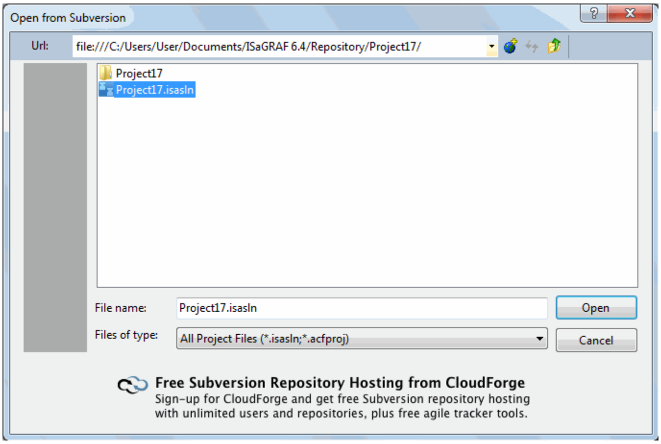
1. In the History Viewer, note the revision number for the specific version of the element to retrieve, then close the viewer.



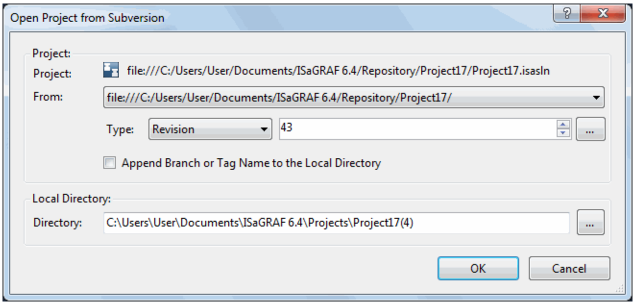

2. Close the solution.

3. From the File menu, point to **Subversion**, then click **Open from Subversion**.

4. In the Open from Subversion dialog box, select the required solution in the repository, then click **Open**.




5. In the Open from Subversion dialog box, verify the project and location information, select the Revision type, specify the revision number for the version to retrieve, and define the empty local directory in which to place the solution, then click **OK**.




The solution files for the specific revision is opened in the Solution Explorer.

**See also**

Reverting Versions of Elements
Viewing the History of Elements
Comparing Element Versions
Version Source Control

# Reverting Versions of Elements

You can revert, i.e., roll back, versions of solutions and all sub-elements to a specific revision to continue making modifications from this revision. Reverting enables duplicating a project from repository (as a branch) either from the latest version or any previous version. Reverting the solution, project, device, or resource to a previous version involves getting a working copy, removing the element from the source control, and reinserting the element in the source control. Reverting POUs (without child) and ISaVIEW screens involves reverting from the element history and locking the element for exclusive use.

When reverting versions of elements, you can perform the following tasks:

- Revert a solution to a specific revision and proceed from this version
- Revert a project, device, resource, or POU to a specific revision and proceed from this version
- Revert a POU (without child) or ISaVIEW screen to a previous version and proceed from this version

**To revert a solution to a specific revision and proceed from this version**

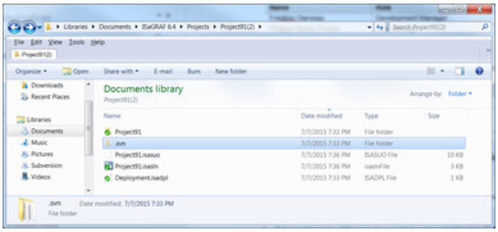
When reverting a solution to a specific revision for duplication in a repository, you need to get a working copy of the required revision from the repository, remove all source control bindings from the version, then duplicate the solution in the repository either using the same name or a different name.

**1.** Get a working copy of the solution to remove from binding to the repository.

**2.** Close the solution.

**3.** Remove source control from the retrieved solution:

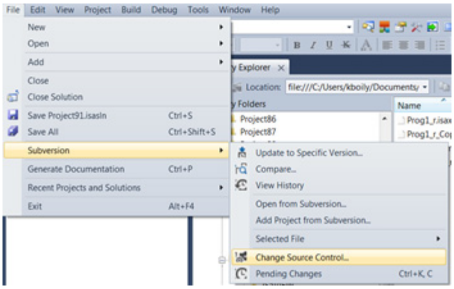
   **a)** From Windows Explorer while displaying hidden files, locate the solution folder and delete the .svn folder.



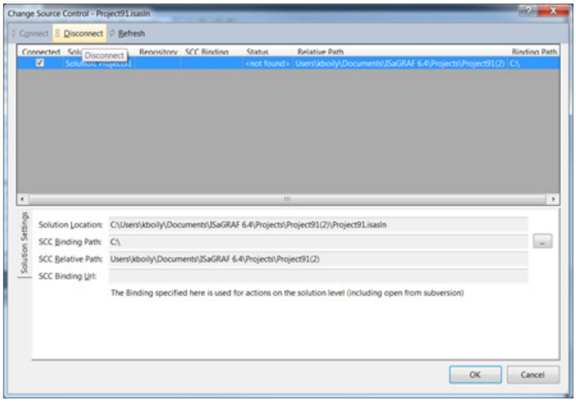
**b)** Re-open the solution in the workbench.

**Note:** You can choose to rename the solution or retain the existing name to prepare a duplicate in the repository.

**c)** From the File menu, point to **Subversion**, then click **Change Source Control**.




**d)** In the Change Source Control dialog box, click **Disconnect**, then **OK**.



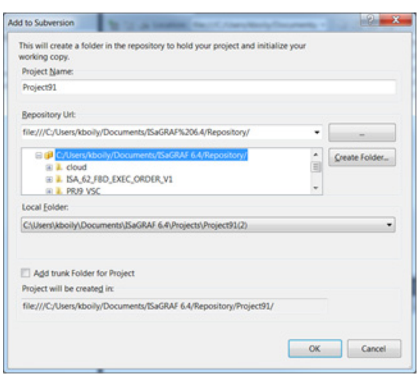

**e)** From the File menu, click **Save All**.

All bindings to the repository are removed.

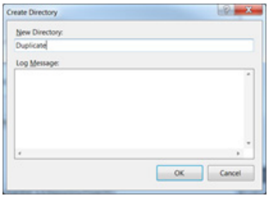
**4.** Duplicate the solution in the same repository:

**a)** In the Solution Explorer, right-click the solution, then click **Add Solution to Subversion**, from the contextual menu.

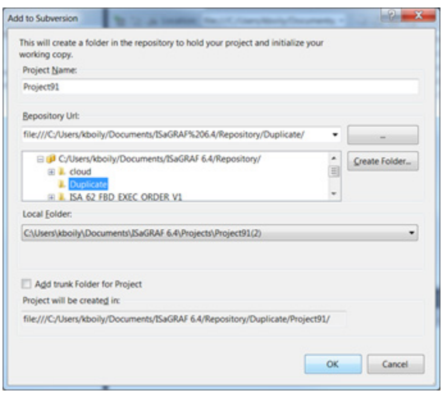
**b)** In the Add to Subversion dialog box, select the repository URL and the location in which to place the solution duplication, then click **Create Folder**.



c) In the Create Folder dialog box, provide a name for the duplicate solution, then click **OK**.

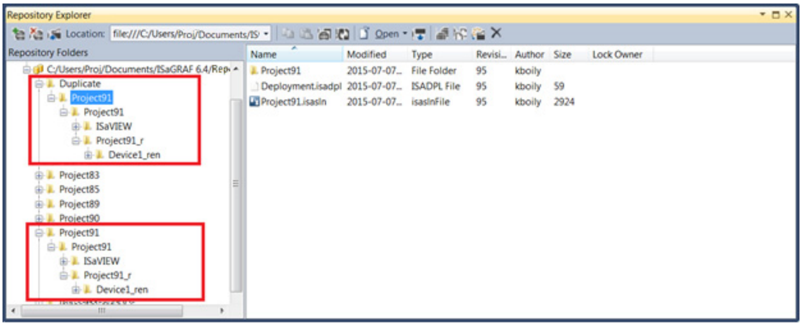



d) In the Add to Subversion dialog box, select the solution folder from the Repository URL list for the duplicate solution, then click **OK**.




e) From the Pending Changes window, commit the entire duplicate solution to the repository.

5. In the Repository Explorer, note the duplicate solution in the tree structure.




Each solution is separate in the repository and follows individual sets of changes.

**To revert a project, device, resource, or POU to a specific revision and proceed from this version**

You can revert, i.e., rollback, elements such as projects, devices, resources, and POUs to a specific revision and continue making modifications from this revision. You can revert element after getting a version of the solution containing the required elements.

1. From the retrieved solution, right-click the desired element and export the element in the *.7z format.
2. Place the *.7z file for the element in a safe location. You can choose to delete the retrieved solution working copy.
3. Create a working copy of the latest solution from the repository.
4. To replace the element from the latest solution with the element exported in the *.7z format. To add the retrieved element without deleting anything, skip this step.
   a) Delete the element to replace from the solution.
   b) From the File menu, click **Save All**.
   c) From the Pending Changes window, commit all changes to the repository.
5. Lock the immediate parent of the element to replace.

6. Import the replacement element *.7z file while making sure not to replace any existing elements. Specify all imported elements as “new”. Incorrect elements must be replaced before importing.

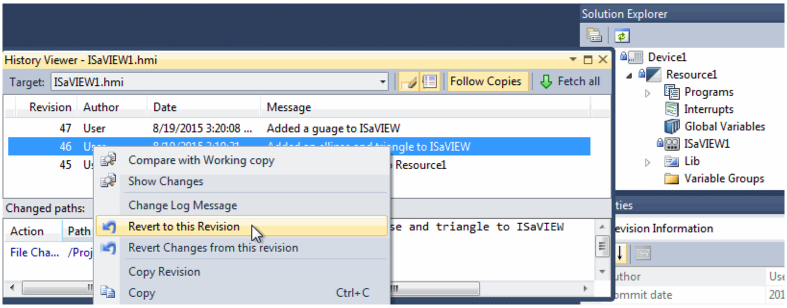
7. From the Pending Changes window, commit all changes to the repository.

The current solution containing the imported (and reverted) element is available as a working copy in which to make modifications.

**To revert a POU or ISaVIEW screen to a previous version and proceed from this version**

You can revert, i.e., rollback, a POU (without child) or ISaVIEW screen to a specific revision from the repository. Reverting a POU or ISaVIEW screen consists of getting a specific revision from the element history. You can also revert elements having been renamed, however, such elements retain the latest name. For example, when reverting POU3 to a previous revision where it was named POU2, the POU retains the POU3 name. Following reversion, you can rename the element to its previous name.

1. In the History Viewer, right-click the required revision number for the specific version of the element to retrieve.




2. From the contextual menu, click **Revert to this Revision**.

   The element is available as defined for the retrieved specific revision.

3. Lock the element and make the required modifications.

4. From the File menu, click **Save All**.

5. From the Pending Changes window, commit all changes to the repository.

**See also**

Getting Versions of Elements
Viewing the History of Elements
Comparing Element Versions
Version Source Control

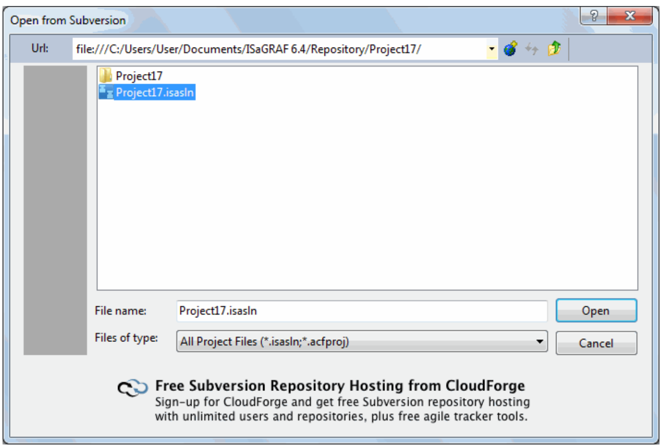
# Creating a Working Copy from a Repository

You can create a local working copy of a solution stored in a repository.

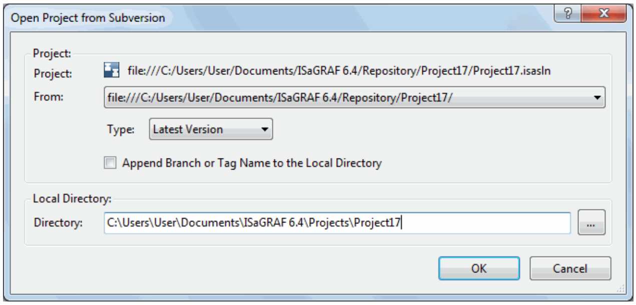
**To create a local working copy of a solution from a repository**

1. From the File menu, point to **Subversion**, then click **Open from Subversion**.

2. In the Open from Subversion dialog box, select the required solution in the repository, then click **Open**.




3. In the Open from Subversion dialog box, verify the project and location information, select the Latest Version type, and specify the local directory in which to create the working copy, then click **OK**.




The local working copy of the solution from the repository is opened in the Solution Explorer.

**See also**

Getting Versions of Elements
Reverting Versions of Elements
Viewing the History of Elements
Version Source Control

## Locking and Unlocking Elements

You can lock elements to establish exclusive access when making changes. When committing changes to locked files, the lock is removed unless you select the Keep locks option. When you complete changes on locked elements, you can unlock these elements to enable access to others.

**Warning:** When opening solutions and projects from a repository, i.e., checking out, you can steal locks from others having locks on these same elements. However, you should avoid performing such operations unless absolutely required, for instance, when a user having the elements checked out is no longer available. Modifications performed on such elements by the original lock holder are no longer available for committing.

**To lock an element for exclusive access**

When locking elements, sub-elements are not locked.

1. From the Solution Explorer, right-click the element to lock for exclusive access.
2. From the contextual menu, click **Lock**.

**To unlock an element to enable access to others**

When unlocking elements, sub-elements are unlocked.

1. From the Solution Explorer, select the element to unlock.
2. From the File menu, point to **Subversion**, then click **Unlock**.

### See also

Committing Pending Changes
Version Source Control

# Viewing the History of Elements

You can view the history of elements such as solutions, projects, devices, resources, POUs, and ISaVIEW screens. The history of an element is usually available from the time of creation.

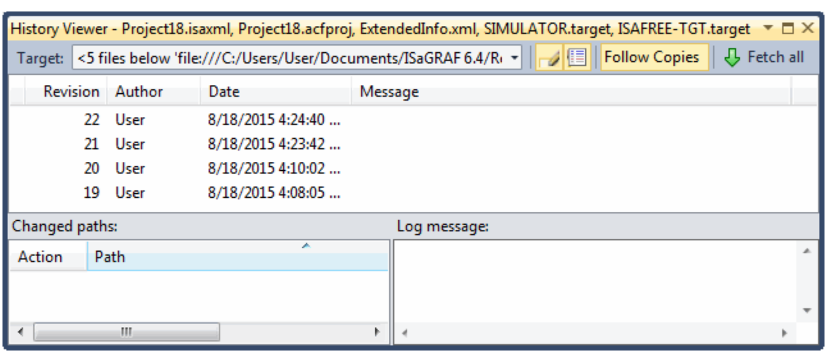
You view the history of elements in the History Viewer. The viewer displays log messages from the repository. The viewer is split into three panes: list of revisions, changed paths, log message. The top pane lists the revisions including the date and time, author and first line of the log message for each revision. The Changed paths pane lists the files and folders associated with a selected revision. The files or folders having changed paths are displayed using colored font: blue font for modifications, red font for deletions, and dark red for additions. The Log message pane displays the complete log message for a selected revision.

In the History viewer, you can fetch all revisions for an element. You can also choose to view the strict node history displaying the equivalent of svn log with the --stop-on-copy option. You can also choose to display the changed paths and log messages.

**To view the history of an element**

1. From the Solution Explorer, right-click the element for which to view the history.

2. From the contextual menu, click **View History**.




The History Viewer displays all revisions for the element. For some element revisions, you can open the XML format in a textual editor.

**To modify the layout of the History viewer**

1. To display the changed paths for revisions, click  .

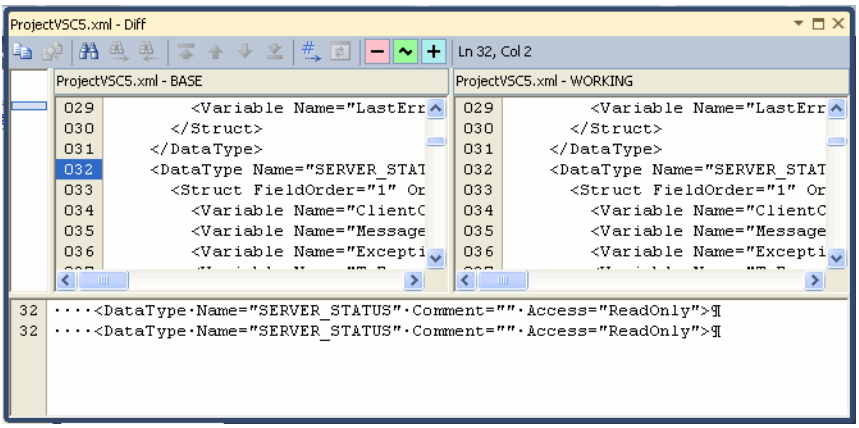
2. To display the complete log messages for revisions, click  .

### See also

Comparing Element Versions
Canceling Local Modifications
Version Source Control

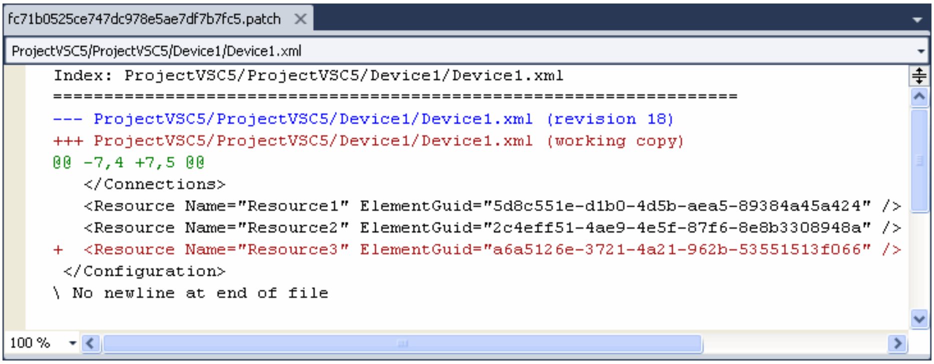
# Comparing Element Versions

You can choose to compare the XML format files for different versions of elements including projects, devices, resources, POUs, and ISaVIEW screens. These different versions can include the latest version, working version, base version, committed version, previous version, and a specific revision. You compare these revisions in a side-by-side textual editor having many basic editing, search, and comparison functions. For a selected line, both versions of text are placed in the lower pane.




You can also perform a unified differences of all items within a project. These different versions can include the latest version, working version, base version, committed version, previous version, and a specific revision version. You view the differences in single-pane window.

For comparisons and unified differences, the differences are displayed using colored font: red font for deletions, green font for changes, and blue font for insertions.




**To compare versions of an element**

1. In the Solution Explorer, select the element for which to build a comparison.
2. From the File menu, point to **Subversion**, then click **Compare**.
3. In the Compare Files dialog box, select the element for the comparison, specify the From type and the To type to compare, then click **OK**.

Each file for the comparison file is displayed in a side-by-side textual editor.

**To perform a unified differences for a solution**

1. In the Solution Explorer, select the solution for which to build a unified differences, then right-click.
2. From the contextual menu, point to **Subversion**, then click **Unified diff**.
3. In the Unified Diff dialog box, select the items for which to view differences, then specify the From version and the To version, then click **OK**.

The differences between the versions for the selected element are displayed in a single window.

## See also

Viewing the History of Elements

Getting Versions of Elements
Canceling Local Modifications
Version Source Control

## Canceling Local Modifications

You can choose to cancel modifications made locally to elements, i.e., revert, to the state prior to local changes since it was last updated.

**To revert an element to the state prior to local changes**

1. In the Pending Changes window, highlight all files, then right-click.
2. From the contextual menu, click Revert.

   The element is returned to the latest committed version state.
3. To unlock the elements, in the Pending Changes window, highlight all files having the Locked state, then right-click.
4. From the contextual menu, point to **Subversion**, then click **Unlock**.

The element is available for all users.

**See also**
Viewing the History of Elements
Getting Versions of Elements
Comparing Element Versions
Version Source Control

## Version Source Control Keyboard Shortcuts

The following keyboard shortcuts are available for use with the Repository Explorer and Working Copy Explorer.

| | |
|---|---|
| Ctrl+K, R | Opens the Repository Explorer |
| Ctrl+K, W | Opens the Working Copy Explorer |
| Ctrl+K, C | Opens the Pending Changes - Source Files window |
| Alt+Left Arrow | Collapses the selected folder |
| Alt+Right Arrow | Expands the selected folder |
| Shift+Home | In the explorer view, moves to the first item.<br>In the folder view, selects from the current file/folder to the first file/folder. |
| Shift+End | In the explorer view, moves to the last item.<br>In the folder view, selects from the current file/folder to the last file/folder. |
| Home | Moves to the first item |
| End | Moves to the last item |

# Dictionary

The Dictionary, i.e., tag editor, is the environment where you manage variables, arrays, structures, and defined words. The Dictionary is made up of multiple grids having different purposes.

- Arrays Grid, enables managing the arrays for a project
- Structures Grid, enables managing the structures for a project
- Defined Words Grid, enables managing the defined words for a project
- Variables Grid, enables managing the variables for resources and programs. Each resource and program has its instance of the grid. For resources, the grid displays global variables. For programs, the grid displays local variables.

The grids each display the properties for the type of element. You can open multiple grid instances simultaneously. When working in a grid, you can navigate the cells using the mouse controls. For complex data types, you can expand fields using Ctrl+PLUS SIGN on numeric keypad (+) and collapse fields using Ctrl+MINUS SIGN on numeric keypad (-).

You access Dictionary instances from the Solution Explorer.

You can customize the Dictionary environment by arranging the columns to display and setting the display colors.

**To access a Dictionary grid instance**

1. From the Solution Explorer, expand the project and device nodes.
2. For the variables of a resource, expand the required resource node, then double-click the **Global Variables** element.

   The Dictionary instance is displayed containing the variables belonging to the resource.
3. For the variables of a program, expand the required program node, then double-click the **Local Variables** element.

   The Dictionary instance is displayed containing the variables belonging to the program.

4. For the data types of a project, expand the required **Project** node, then double-click the **Data Types** element.

   The data types Dictionary instance is displayed with the Arrays, Structures, and Defined Words tabs.

**To arrange the columns to display**

To retain customized display settings, you must save the Dictionary instance before closing.

1. To move a column, drag the column header to another location.

   When dragging a column header, arrows indicate the current position of the header.

2. To hide a column, right-click a column header, then click **Hide Column**.

3. To show a column, right-click on any column header, click **Show Column**, and then select the desired column name.

# Arrays Grid

The Arrays grid of the Dictionary enables managing the arrays for a project. You can perform the following tasks from the Arrays grid:

- Creating arrays
- Editing existing arrays
- Deleting arrays
- Sorting arrays in the grid
- Filtering arrays in the grid

For arrays, the properties are the following:

| Column | Description | Possible Values |
|---|---|---|
| Name | Name of the array | Limited to 128 characters beginning with a letter or underscore character followed by letters, digits, and single underscore characters. These names cannot have two consecutive underscore characters. |
| Data Type | Type of the array | BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, User arrays, Structures |
| Dimension | The dimension of the array | Example: [1..10] for a one dimensional array, [1..4,1..7], for a two dimensional array. The dimension is defined as a positive double integer (DINT) value. |

| Column | Description | Possible Values |
|---|---|---|
| Comment | Comment for the array | Free-format text |
| String Size | If Data Type is STRING, represents the length | String capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string |

You can customize the Dictionary environment by arranging the columns to display.

**To create an array**

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Arrays** tab.
3. In an empty row of the Arrays grid, define the required properties for the array, then press ENTER.

**To edit an existing array**

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Arrays** tab.
3. In the Arrays grid, make the required changes.

**To delete an array**

You can delete arrays from the Arrays grid.

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Arrays** tab.

3. In the Arrays grid, select the array by clicking the left-most column, and then click **Delete**.

**To sort arrays in the grid**

You can sort the arrays in the grid using an ascending or descending order for the individual columns.

1. From the Solution Explorer, expand the project, device, resource, and lib nodes, then double-click the **Data Types** item.
2. From the Data Types instance, click the **Arrays** tab.
3. In the Arrays grid, select the required column header.

   An arrow showing the current order is displayed on the column header.
4. Toggle the column header to switch between ascending and descending order.

**To filter arrays in the grid**

You can filter arrays displayed on the Arrays tab of Data Types instance. When filtering, you create a view displaying only the arrays containing specified characters.

The filter row is the top row of the grid. You can filter arrays by typing alphabetical and numerical characters in the cells of the filter row. You can also select from the drop-down-combo box. Matching arrays are automatically displayed.

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Arrays** tab.

3. In the filter row of the Arrays grid, click the required cell, then do one of the following:
   - Type the characters to use in the filtering operation
   - Select the required array from the drop-down combo-box

### See Also

Dictionary

# Structures Grid

The Structures grid of the Dictionary enables managing the structures for a project. You can perform the following tasks from the Structures grid:

- Creating structures
- Editing existing structures
- Deleting structures
- Sorting structures in the grid
- Filtering structures in the grid

For structures, the properties are the following:

| Column | Description | Possible Values |
|---|---|---|
| Name | Name of the structure | Limited to 128 characters beginning with a letter or underscore character followed by letters, digits, and single underscore characters. These names cannot have two consecutive underscore characters. |
| Data Type | Type of the structure | BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, User arrays, Structures |
| Comment | Comment for the structure | Free-format text |
| String Size | If Data Type is STRING, represents the length | String capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string |

You can customize the Dictionary environment by arranging the columns to display.

**To create a structure**

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Structures** tab.
3. In an empty row of the Structures grid, define the required properties for the structure, then press ENTER.

**To edit an existing structure**

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Structures** tab.
3. In the Structures grid, make the required changes, then press ENTER.

**To delete a structure**

You can delete structures from the Structures grid.

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Structures** tab.
3. In the Structures grid, select the structure by clicking the left-most column, and then click **Delete**.

**To sort structures in the grid**

You can sort the structures in the grid using an ascending or descending order for the individual columns.

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Structures** tab.

3. In the Structures grid, select the required column header.

   An arrow showing the current order is displayed on the column header.

4. Toggle the column header to switch between ascending and descending order.

**To filter structures in the grid**

You can filter structures in Structures grid. When filtering, you create a view displaying only the structures containing specified characters.

The filter row is the top row of the grid. You can filter structures by typing alphabetical and numerical characters in the cells of the filter row. You can also select from the drop-down-combo box. Matching structures are automatically displayed.

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.

2. From the Data Types instance, click the **Structures** tab.

3. In the filter row of the Structures grid, click the required cell, then do one of the following:

   - Type the characters to use in the filtering operation
   - Select the required structure from the drop-down combo-box

## See Also

Dictionary

## Defined Words Grid

The Defined Words grid of the Dictionary enables managing the defined words for a project. You can perform the following tasks from the defined words grid:

- Creating defined words
- Editing existing defined words
- Deleting defined words
- Sorting defined words in the grid
- Filtering defined words in the grid

For defined words, the properties are the following:

| Column | Description | Possible Values |
|---|---|---|
| Word | Name of the defined word | Limited to 128 characters beginning with a letter followed by letters, digits, and underscores. Defined words cannot contain defined words. |
| Equivalent | String replacing the defined word during compilation. For example, the defined word "PI" is replaced by its equivalent "3.14159" | Limited to 128 characters |
| Comment | Comment for the defined word | Free-format text |

You can customize the Dictionary environment by arranging the columns to display.

**To create a defined word**

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Defined Words** tab.
3. In the Defined Words grid, define the required properties, then press ENTER.

**To edit an existing defined word**

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Defined Words** tab.
3. In the Defined Words grid, make the required changes.

**To delete a defined word**

You can delete defined words from the Defined Words grid.

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Defined Words** tab.
3. In the Defined Words grid, select the defined word by clicking the left-most column, and then click **Delete**.

**To sort defined words in the grid**

You can sort the defined words in the grid using an ascending or descending order for the individual columns.

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Defined Words** tab.
3. In the Defined Words grid, select the required column header.

   An arrow showing the current order is displayed on the column header.
4. Toggle the column header to switch between ascending and descending order.

**To filter defined words in the grid**

You can filter defined words in Defined Words grid. When filtering, you create a view displaying only the defined words containing specified characters.

The filter row is the top row of the grid. You can filter defined words by typing alphabetical and numerical characters in the cells of the filter row. You can also select from the drop-down-combo box. Matching defined words are automatically displayed.

1. From the Solution Explorer, access the Dictionary instance for the data types of the project.
2. From the Data Types instance, click the **Defined Words** tab.
3. In the filter row of the Defined Words grid, click the required cell, then do one of the following:
   - Type the characters to use in the filtering operation
   - Select the required defined word from the drop-down combo-box

### See Also

Dictionary

# Variables Grid

The variables grid of the Dictionary enables managing the variables for a resource or program. Each resource and program has its instance of the grid. For resources, the grid displays global variables. For programs, the grid displays the local variables. You can perform the following tasks from the variables grid:

- Creating variables
- Editing existing variables
- Dragging variables
- Deleting variables
- Sorting variables in the grid
- Filtering variables in the grid

For variables of resources or programs, the properties are the following:

| Column | Description | Possible Values |
|---|---|---|
| Name | Name of the variable | Limited to 128 characters beginning with a letter or underscore character followed by letters, digits, and single underscore characters. These names cannot have two consecutive underscore characters. |
| Logical Value | Available while monitoring applications. Displays the value used by code being executed on the virtual machine. You can force the value of variables. | Values are displayed according to the variable data type |
| Physical Value | Available while monitoring applications. Displays the value sent to and received from the drivers. You can force the value of variables. | Values are displayed according to the variable data type |

| Column | Description | Possible Values |
|---|---|---|
| Lock | Available when online. The indication of whether the value of the variable is locked. Locking operates differently for simple variables, array and structure elements, and function block parameters. For simple variables, individual variables are locked directly. For structure and array elements, locking an element locks all the elements of the structure or array. | Yes or No |
| Data Type | Data type of the variable | BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, Array types, Structure types, Function blocks |
| Dimension | The size (number of elements) of an array. | For example: [1..3,1..10] - represents a two-dimensional array containing a total of 30 elements. |
| String Size | For String type variables, indicates the maximum length | String capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string |
| Initial Value | Value held by a variable when the virtual machine starts the execution of the resource code | The initial value of a variable can be the default value, a value given by the user when the variable is defined or the value of the retain variable after the virtual machine has stopped. |
| Direction | For I/O wiring, indicates whether a variable is an input, output, or internal. | VarInput, VarOutput, or Var |

<table>
<tr><th>Column</th><th>Description</th><th>Possible Values</th></tr>
<tr><td>Attribute</td><td>The property of a variable indicating its read and write access rights.</td><td>Read, Write, or Read/Write</td></tr>
<tr><td>Retained</td><td>The indication of whether the value of the variable is saved by the virtual machine at each cycle.</td><td>Yes or No</td></tr>
<tr><td>Comment</td><td>User-defined text</td><td>Free format</td></tr>
<tr><td>Alias</td><td>Any name (for use in LD POUs)</td><td>Limited to 128 characters consisting of letters, digits, and the following special characters: !, #, $, %, &, \, *, +, -, ,/ <, :, =, >, ?, @, \, ^, _, `, |, and ~.</td></tr>
<tr><td>Wiring</td><td>Read-only cell, generated by the I/O wiring tool indicating the I/O channel to which the variable is wired</td><td>Uses the syntax of Directly Represented Variables</td></tr>
<tr><td>Address</td><td>User-defined address of the variable</td><td>The format is hexadecimal and the value ranges from1 to FFFF.</td></tr>
<tr><td>Groups</td><td>Variable groups containing the variable listed in alphabetical order</td><td>User-defined variable group names</td></tr>
</table>

You can customize the Dictionary environment b y arranging the columns to display.

**To create a variable**

1. From the Solution Explorer, access the Dictionary instance for the required resource or program.

2. In an empty row of the variables grid, define the required properties for the variable, then press ENTER.

**To edit an existing variable**

1. From the Solution Explorer, access the Dictionary instance for the required resource or program.

2. In the variables grid, make the required changes.

**To drag a variable**

You can drag variables from a Dictionary instance to multiple locations within a project. These locations include other Dictionary instances as well as elements within a language container.

You drag variables to other locations individually. When dragging a variable to another Dictionary instance, you can place the variable anywhere in the grid. When dragging a variable into a language container, you can place the variable anywhere in the language container. To retain changes made to Dictionary instances and language containers, save the respective instance or POU before closing.

1. From the Solution Explorer, access the Dictionary instance containing the required variable and the destination for the variable.

2. From the Dictionary instance containing the required variable, in the variables grid, select the variable by clicking the cell in the left-most column.

   The selection indicator () is displayed in the leftmost column.

3. Drag , placing the variable in the grid or open language container.

   The variable is displayed at the destination.

**To delete a variable**

You can delete variables from Dictionary instances. Deleting variables from an instance opened for a program element removes the variables from the instance only.

1. From the Solution Explorer, access the Dictionary instance for the required resource or program.

2. In the variables grid, select the variable by clicking in the left-most column, and then click **Delete**.

**To sort variables in the grid**

You can sort the variables in the grid using an ascending or descending order for the individual columns.

1. From the Solution Explorer, access the Dictionary instance for the required resource or program.

2. In the variables grid, select the required column header.

   An arrow showing the current order is displayed on the column header.

3. Toggle the column header to switch between ascending and descending order.

**To filter variables in the grid**

You can filter variables in variables grid instances. When filtering, you create a view displaying only the variables containing specified characters.

The filter row is the top row of the grid. You can filter variables by typing alphabetical and numerical characters in the cells of the filter row.You can also select from the drop-down-combo box. Matching variables are automatically displayed.

1. From the Solution Explorer, access the Dictionary instance for the required resource or program.

2. In the filter row of the variables grid, click the required cell, then do one of the following:

   - Type the characters to use in the filtering operation
   - Select the required variable from the drop-down combo-box

## See Also

Dictionary

# Cross Reference Browser

The cross reference browser provides an overview of the variables, programs, functions, function blocks, and defined words existing in a project including information such as names, various properties, location of usage, and comments. When locating items in the browser, you need to select a context view for the type of elements to locate. You can find specific elements by name or filter the element list. In the browser, some columns from the different context views reflect the respective properties of the items. You can sort the items in the list according to the different column headings in ascending or descending order.

The Cross Reference Browser toolbar contains the following:


Enables locating elements within the element list. You can type the element name in the field or select a previous search from the drop-down combo-box.

Enables locating the element specified in the Find field within the element list


Enables filtering the element list using text. You can type element name in the field or select a previous search from the drop-down combo-box. To remove a filter, select <Remove Filter> from the drop-down combo-box.

Enables selecting a context view for the element list. Possible views include Variable, Programs, Functions, Function Blocks, and Defined Words.

Enables refreshing the element list to include the latest elements from the solution

Enables displaying the previous instance of the selected element within the project

Enables displaying the next instance of the selected element within the project

The properties for the Variables view are the following:

| | |
|---|---|
| Name | Name of the variable |
| Scope | Range of accessibility of a variable in relation to POUs of a resource |
| Alias | Alias name of the variable |
| Type | Data type of the variable |
| Project | Project using the variable |
| Device | Device using the variable |
| Resource | Resource using the variable |
| Comment | Comment of the variable |
| Group | Group containing the variable |

The properties for the Programs view are the following:

| | |
|---|---|
| Name | Name of the program |
| Language | Programming language of the program |
| Project | Project using the program |
| Device | Device using the program |
| Resource | Resource using the program |
| Comment | Comment of the program |

The properties for the Functions and Function Blocks views are the following:

| | |
|---|---|
| Name | Name of the function or function block |
| Category | Type of function or function block. Possible types are standard, user-defined, and native. |
| Language | Programming language of the function or function block |
| Project | Project using the function or function block |
| Device | Device using the function or function block |
| Resource | Resource using the function or function block |
| Comment | Comment of the function or function block |
| From Library | Library containing the function or function block |

The properties for the Defined Words view are the following:

| | |
|---|---|
| Name | Name of the defined word |
| Equivalent | String replacing the defined word during compilation. For example, the defined word "PI" is replaced by its equivalent "3.14159" |
| Project | Project using the defined word |
| Comment | Comment for the defined word |

In the element list, color is used to identify elements used within the project. Element displayed as blue are used at least once within the project. Elements displayed as red are not in use. For elements other than programs, you can select elements displayed as blue to view the location information for each instance of the element within the project. You can jump to the location of individual instances by double-click in the instance list.

When working in the Cross Reference Browser, you can navigate using keyboard and mouse controls.

| | |
|---|---|
| Arrow keys | Enable moving up, down, left and right within the rows of the elements list |
| Tab key | Enables moving left and right within the fields and commands of the toolbar. Also enables moving from the elements list to the list of instances. |
| Esc key | Enables moving from the Cross Reference Browser to the workspace |

**To access the Cross Reference Browser**

- From the View menu, click **Cross Reference Browser** (or press **Ctrl+W, Ctrl+C**).

The Cross Reference Browser is displayed.

**To locate element in the Cross Reference Browser**

When locating elements in the Cross Reference Browser, you refresh the list of cross references by clicking (or pressing **Ctrl+T, Ctrl+R**).

**1.** From the Context View drop-down list, select the type of elements to display.

**2.** To refine the element list, do one of the following:

- To sort the elements list, click the column heading by which to sort in an ascending order. Clicking twice sorts in descending order.
- To filter the elements list, type in the filter field or select a previous filter from the drop-down combo-box.

**3.** Select the required element by performing one of the following:

- Type the name of the element in the Find field or select a previous search from the drop-down combo-box, then click .
- Scroll through the element list.

**To jump to an instance of an element**

In the element list, elements displayed in blue are used at least once within the project. For these elements, selecting in the element list displays all related instances in the instance list.

**1.** In the Cross Reference Browser, locate the required element.

**2.** In the instance list, double-click the required instance of the element (or press **F8** or **Shift+F8**).

The program containing the instance of the element is displayed with the instance selected.

# Device View

The device view is a graphical environment enabling navigation through project elements including devices, resources, and POUs. The navigation consists of vertical links on the left pane and a breadcrumbs trail in the address field. The device view displays information for individual devices and resources. For individual devices, you can access the following information:

- Device information such as the name, comment, target, description, memory size, and online behavior
- The resources contained in a device
- The list of C functions and function blocks available for the device based on the target type. Furthermore, selecting C functions or function blocks enables viewing the parameters.
- The list of target I/O devices available for the device. Information available for the I/O devices includes package, driver, name, data type, direction, channels, parameters, and channel parameters.
- Features of the target type attached to the device, including target name, memory size, and supported characteristics

For individual resources, you can access the following information:

- Resource information such as the name, comment, number, description, and size of code.
- The POUs and interrupts defined in a resource. For programs and interrupts, you can open individual programs or interrupts by double-clicking the required instance. For user-defined functions and function blocks, you can access the parameters view by single-clicking the instance or open the POU by double-clicking the instance.
- The list of interrupts available for a resource
- Other elements attached to resources including ISaVIEW screens and variable groups
- Resource properties such as cycle time, memory size for online changes, code type, compiler options, and extended parameters.

You can open an instance of the device view for each device within your project.

**To access the Device View**

- In the Solution Explorer, right-click the required device, and then click **Open**.

  The device view is displayed in the workspace.

**To view device and resource information**

1. To view device information, click the device instance located in the left pane of the Device View.

   The device information is displayed in the properties pane.

2. To view resource information, in the Device View, click on the device item, then click the required resource instance.

   The resource information is displayed in the properties pane.

**To display device elements**

1. To display the list of available C Functions and Function Blocks including the parameters, perform the following:

   - From the Device View, in the breadcrumbs trail, click and then click **C Functions & Function Blocks**.

2. To display the I/O devices available for the target including information for each I/O device instance, perform the following:

   - From the Device View, in the breadcrumbs trail, click and then click **Target I/O Devices**.

3. To display the features of the target type attached to the device, perform the following:

- From the Device View, in the breadcrumbs trail, click and then click **Target Features**.

**To display resource elements**

1. From the Device View, in the breadcrumbs trail click at the device level, then click the required resource instance.

2. To display programs, click on the **Programs** item, then double-click the required program instance.

   The program is displayed in the language container.

3. For interrupts click on the **Interrupts** item, then perform the following:
   - To display the list of Interrupts, double-click the Interrupts item.
   - To open the interrupt POU in the language container, double-click the interrupt instance.

4. For functions, click on the **Functions** item, then perform the following:
   - To display the Parameters view for a function, click the required function instance.
   - To open the function in the language container, double-click the function instance.

5. For function blocks, click on the **Function Blocks** item, then perform the following:
   - To display the Parameters view for a function block, click the function block instance.
   - To open the function block in the language container, double-click the function block instance.

6. To display ISaVIEW screens, click on the **Others** item, then double-click the required ISaVIEW instance.

7. To display variable groups, click on the **Others** item, expand the Variables Group item, then double-click the required variable group instance.

**To display resource properties**

1. From the Device View, in the breadcrumbs trail click at the resource level, and then click Properties.

The resource properties are displayed

# Controller Status

You can access real-time status information for all controllers, i.e., devices, in a project. The available information is the following:

| | |
|---|---|
| Name | Name of the controller |
| Status | Status of the controller:<br>- Building, indicates that the project, device, or resource (if supported by the CAM) build is in progress<br>- Unable to Connect, indicates that the Workbench is unable to connect with the controller<br>- Need Password, indicates that the Workbench requires the controller password to connect and provide status information<br>- Need Save Status, indicates that the project requires saving then refreshing the controller status<br>- Unavailable, indicates that the Workbench is switching from design to online or simulation mode<br>- Simulator Running, indicates that the Workbench is unable to connect with the controller since it is in simulation mode<br>- Connecting, indicates that the Workbench is retrieving status information or is unable to communicate with the controller<br>- Running, indicates that the controller is running an application while displaying the application version and date<br>- Stopped, indicates that the controller is not running an application<br><br>Controller statuses are also displayed with a system health condition: Healthy, Faulted, or Warning. |
| Locked Variables | Number of locked input and output variables for the controller. |
| Cycle Time | Current cycle time of the controller |

When viewing the Controller Status, the Name column displays icons that indicate the connection status of the controllers.

 Healthy, indicates a healthy, fault-free status

 Faulted, indicates an unhealthy, faulted status

 Warning, indicates a connection problem or a mismatch between the local and running application versions

**To access the status information for controllers**

- From the View menu, click **Controller Status**.

# I/O Wiring

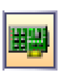
I/O wiring enables the definition of links between variables defined for a project and I/O channels of I/O devices existing on a target system. You perform I/O wiring from an I/O Device instance accessed for a resource. Each resource can instantiate none, one, or multiple I/O device instances.

When performing I/O wiring for a resource, you access an I/O Device instance, add I/O devices, and wire the I/O channels to variables. When defining I/O wiring for the first time, an I/O Device instance is empty.

An I/O Device instance consists of three sections:

- A hierarchical tree-like structure displaying devices, parameters, and comments
- A wiring grid enabling the association of channels with variables. The wiring grid displays the name of all variables. When online, the wiring grid also displays the logical value, physical value, and lock status of all variables.
- A section listing the options for individual channels

The I/O Wiring toolbar enables performing many tasks in I/O Device instances:


Adding an I/O device


Deleting an I/O device


Freeing all channels of an I/O device


Frees individual channels of an I/O device


Toggles an I/O device between real and virtual


While debugging, locks a channel

| | |
|---|---|
| | While debugging, unlocks a channel |
| | Shows or hides I/O device information |

The tree-structure hierarchy is composed of the following items:

| | | |
|---|---|---|
| Simple I/O Device | | |
| | Comment | Comment of the simple I/O device |
| | Parameters | |
| | *Parameter Name* | Individual parameter defined for the device |
| Complex I/O Device | | |
| | Comment | Comment of the complex I/O device |
| | Parameters | |

**To access an I/O Device instance**

- From the Solution Explorer, right-click on a resource, then click **I/O Device**.

I/O Wiring is displayed in the workspace.

## See Also
I/O Devices
I/O Channels

# I/O Devices

I/O devices contain multiple channels having the same type and direction. When adding I/O devices, the Device Selector enables selecting from those available for the target. You specify a device index and a number of channels. You can also include an alias name and comment. The device index value can range from 0 and 65535.

While running online, when devices are set to real, I/O variables are directly linked to the corresponding I/O devices. Input or output operations in the programs correspond directly to the input or output conditions of the actual I/O device fields. When devices are set to virtual, I/O variables are processed as internal variables. The debugger can read or update these to enable simulating I/O processing, but no actual connection is made.

When adding complex devices, the number of channels, i.e., device size, of individual simple devices making up a complex device varies depending on the definition of the complex device in the target.

You manage I/O devices from the hierarchical tree-like structure in an I/O Device instance where these are one of two types:

 Real device

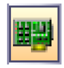
 Virtual device

When selecting I/O devices, their properties are displayed in the Properties window.

From an I/O device instance, you can perform the following tasks when managing the I/O devices of a resource:

- Adding I/O devices
- Toggling I/O devices between real and virtual
- Modifying existing I/O devices
- Deleting I/O devices
- Showing or hiding I/O device information

**To add an I/O device**

1. On the I/O Wiring toolbar, click .

   The Device Selector is displayed.

2. Select the required I/O device from the list of available devices.

   You can sort the listed I/O devices in ascending or descending order by clicking the Name column heading.

3. Specify a device index number, the number of channels, an alias name, and a comment for the I/O device.

**To toggle the real/virtual attribute**

You can toggle between the real and virtual attribute for a selected I/O device.

- From the I/O Wiring toolbar, click .

**To modify an existing I/O device**

1. From the I/O Wiring tree structure, double-click the required I/O device.
2. In the Device Selector, make the required changes to the properties of the device.

**To delete an I/O device**

You can delete devices. You can also disconnect variables attached to selected channels. Note that when deleting devices, all variables are unwired from the device (as with Free I/O device channels).

1. From the I/O Wiring tree structure, click the I/O device element.
2. From the I/O Wiring toolbar, click .

The device is deleted.

**To show or hide I/O device information**

You can toggle between displaying and hiding I/O device information.

- From the I/O Wiring toolbar, click .

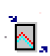
### See Also

I/O Wiring

# I/O Channels

I/O channels represent hardware I/O points. These can be inputs and outputs. A variable is generally connected to a channel to be used in POUs. You can also use directly represented variables in POUs. When adding I/O devices, you specify the number of channels. All I/O channels of a device have the same type and direction.

You wire variables to channels of a device in the wiring grid. When wiring channels, you can apply various operations to individual channels depending on their type. For Boolean channels, you can toggle between the original value (direct) or its negation (reverse). For numerical channels, you can apply gain and offset factors to the value. For all channel types, you can apply conversions available for the target implementation. You apply these operations to individual channels:

| | | |
|---|---|---|
| | Direct | Boolean channels only |
|  | Reverse | Boolean channels only |
|  | Gain | Numeric channels only |
|  | Offset | Numeric channels only |
|  | Conversion | All channel types, depending on target implementation |

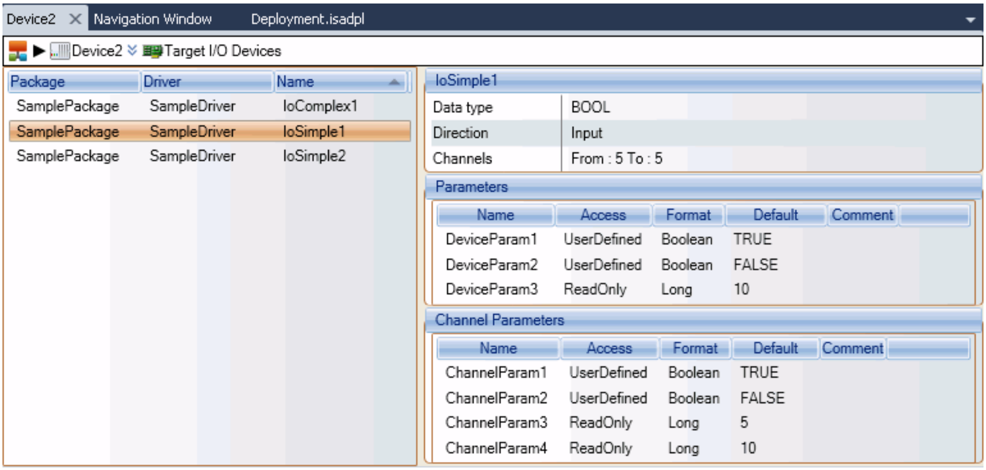
When applying the Gain/Offset factors, the resulting value differs for inputs and outputs. For inputs, the original value (coming from the input device) is multiplied by the gain, and the offset value is added. This results in the value used by the programs of the resource. For outputs, the value of the variable resulting from the execution of the program is multiplied by the gain and the offset value added, before updating the output device. The Gain factor consists of a multiplier factor and a divider factor. The Gain/Offset formula is applied as follows:

```
NewValue = (Value * MultFactor) / DivFactor + Offset
```

Conversions are available for all channel types. However, the available conversions depend on the target implementation.

For details on device-specific implementations of the Gain/Offset factors and conversions, refer to the device documentation.

You can import I/O channel OEM parameters defined in target definition files. I/O channel parameters enable using different settings for individual channels of an I/O device. Individual I/O channel parameter definitions include the parameter name, access type, format, default value, and comment text.




You can use direct variable representation (%IX1.1) to access I/O values when I/O channels have no wiring.

When deleting variables in the Dictionary, channels are automatically unwired. For each deleted variable a new variable is created in the Dictionary. The name of the new variable refers to the name of the channel wired to the deleted variable. For example, deleting a variable wired to channel %IX0.0 creates a variable with the name _IO_IX0_0. Renaming variable _IO_IX0_0 rewires it to channel %IX0.0.

After having wired channels of a device to variables, you can choose to free individual wired channels of a device or free all wired channels of a device.

While debugging, you can choose to force the values of I/O variables.

### To wire the channels of an I/O device

1. Access the I/O Device instance for the required resource.

2. From the tree structure, click the device having the I/O channels to wire. For complex devices, channels are accessed from the simple devices making up the complex device.

3. In the wiring grid, double-click the channel to wire.

4. From the Variable Selector, select the variable for the channel, then click **OK**.

   The channel's Name field indicates the wired variable. For Boolean channels, the default value operations are direct and no conversion.

5. For Boolean channels, to toggle between the direct and reversion operations, select the channel in the grid, then double-click the (Direct) or (Reversion) item below the grid.

6. For numerical channels, to set a Gain and Offset factor, select the channel in the grid and do the following:

   **a)** Double-click the (Gain or Offset) item below the grid.

   **b)** In the I/O Filter dialog box, specify the required values in the Filter section.

7. To apply a conversion to a channel, select the channel in the grid and do the following:

**Note:** Conversions are only available when implemented for a target.

   **a)** Double-click the (Conversion) item below the grid.

   **b)** In the I/O Filter dialog box, select the required conversion.

**To view I/O channel OEM parameters**

1. Import the required target definition file containing the I/O channel parameters.

2. From the Solution Explorer, right-click the required device, and then click **Open**.

3. From the Device View, click , and then click **Target I/O Devices**.

   The Device View displays information on the Target I/O devices.

4. In the left-hand pane, select the required I/O device.

The I/O device definition, I/O device parameters, and I/O channel parameters are displayed in the right-hand pane.

**To free individual channels of an I/O device**

You can free one channel in a device.

1. In the wiring grid, select the wired channel.

2. From the **I/O Wiring toolbar**, click .

**To free all channels of an I/O device**

You can free all channels of a device.

1. In the tree-like structure, select the device having the channels to unwire.

2. From the I/O Wiring toolbar, click .

**To lock and unlock an I/O variable**

You can also lock and unlock variables from a Dictionary instance.

1. Access the I/O Device instance for the required resource.

2. In the tree-like structure, select the device having the channels to lock or unlock.

3. To lock a channel, select the channel in the wiring grid, then click , from the I/O Wiring toolbar.

4. To unlock a channel, select the channel in the wiring grid, then click , from the I/O Wiring toolbar.

**See Also**

I/O Devices

I/O Wiring

# I/O Wiring Keyboard Shortcuts

The following keyboard shortcuts are available for use with I/O wiring. Some shortcuts do not apply or may differ while debugging.

| | |
|---|---|
| Ctrl+N | Adds a device (not available while debugging) |
| Ctrl+F | Frees all channels of a selected device (not available while debugging) |
| Ctrl+R | Frees a channel (not available while debugging) |
| Ctrl+H | Toggles between a real or virtual I/O device (not available while debugging) |
| Ctrl+L | While debugging, locks a channel |
| Ctrl+U | While debugging, unlocks a channel |
| Ctrl+S | Toggles between showing or hiding the full device name |

# Bindings

Bindings are directional links, i.e., access paths, between variables located in different resources. One variable is referred to as the producing variable and the other as the consuming variable. The value stored in the producing variable is transferred to the consuming variable. **ISaGRAF** enables external bindings, which exist between resources belonging to different projects.

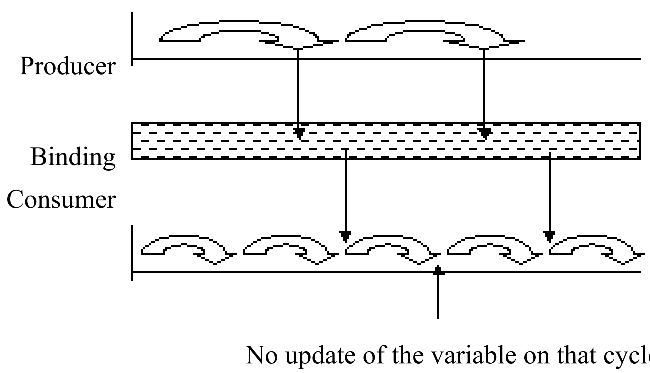
When defining bindings, devices must be connected via a network that supports bindings. For bindings between resources on the same device, the HSD network type must be used. You define network connections using the Deployment View.

**Note:** Online changes are possible as long as binding definitions remain the same.

**Binding** the variable V1 from resource R1 to the variable V2 of resource R2 means that V1 is periodically copied to V2 using memory sharing or network exchanges.

Variables coming from bindings (consumed variables) are refreshed in the resource at the beginning of the cycle, each time the producing resource sends them, i.e. on each end of the producing resource cycle.

The variable is not updated in the consuming resource until the producing resource sends them through the binding media. For example:

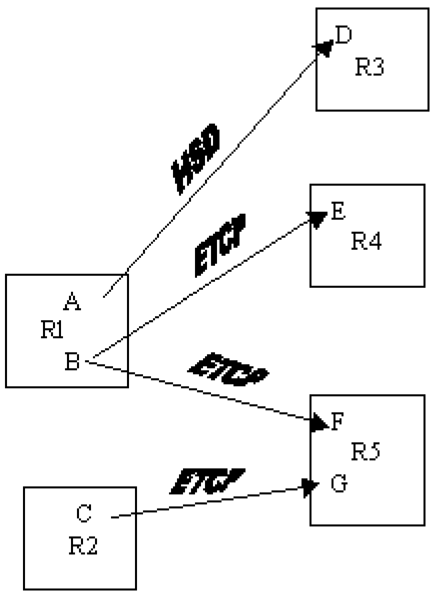



**ISaGRAF** does not impose the read-only accessibility for consumed variables. **However, it is highly recommended to declare consumed variables with read-only attribute in order to avoid conflicts between Binding and execution of POUs.**

## Binding error variables

Binding error variables enable the management of binding errors at the consumer resource level; one error variable for one consumer resource for each resource that produces to this resource. The virtual machine gives specific values to these error variables.

## Example




**Production errors**

The variable 'A' of the R1 resource represents the producer error variable for all binding links starting from R1 and using the HSD driver

(in the example only link from R1 to R3).

The variable 'B' of the R1 resource represents the producer error variable for all binding links starting from R1 and using the ETCP network

(links from R1 to R4 and from R1 to R5).

**Consumption errors**

The variable 'F' of the R5 resource represents the consumer error variable for the unique binding link that comes from R1 and using ETCP.

The variable 'G' of the R5 resource represents the consumer error variable for the unique binding link that comes from R2 and using ETCP.

Depending on the driver used the error variables can take different values with different meanings.

**Warning:** Once the error variable is set to a non-zero value, it has to be reset to 0 by user or by Programs.

To test globally that there is a binding error, you can test the value of the following system variables:

- __SYSVA_KVBPERR: for a production error. It is a Boolean variable. If it is true it means there is a production error.
- __SYSVA_KVBCERR: for a consumption error. It is a Boolean variable. If it is true it means there is a consumption error.

You access the Bindings View from the contextual menu for the projects, devices, and resources in the Solution Explorer.

### For HSD:

To test values of one binding error variable, you should create the following defined words in the dictionary of your project:

The 0 value in the error variable indicates there is no error.

| | | |
|---|---|---|
| ISA_HSD_KVB_ER_MUTEX | 1 | An error occurred with semaphore management |
| ISA_HSD_KVB_ER_SPACE | 2 | An error occurred with memory space access |
| ISA_HSD_KVB_ER_NOKERNEL | 3 | The bound producer is stopped (not running). This error happens only for consumer resources. |
| ISA_HSD_KVB_ER_TIMEOUT | 4 | Variable was not refreshed within the maximum time allowed (ValidityTime). This error happens only for consumer resources. |
| ISA_HSD_KVB_ER_BAD_CRC | 5 | Producer and consumer have different CRC. |
| ISA_HSD_KVB_ER_INTERNAL | 6 | Internal error |

## For ETCP:

To test values of binding error variables, you should create the following defined words in the dictionary of your Project:

A value of 0 in the error variable indicates no error.

| | | |
|---|---|---|
| ETCP_KVB_ERR_BINDING_IN_PROCESS | 1 | The binding initialization process is on its way. |
| ETCP_KVB_ERR_NO_PRODUCER | 2 | The remote producer is not currently runnin g. This error happens only for consumer resources. |
| ETCP_KVB_ERR_BAD_CRC | 3 | Producer and consumer have different CRC. |
| Obsolete error value | 4 | The producer has been stopped. This error happens only for consumer resources. |
| ETCP_KVB_ERR_DATA_DIFFUSSION | 5 | Error during diffusion process. |
| ETCP_KVB_ERR_TIMEOUT | 6 | ETCP server does not answer quickly enough (TimeOut). This error happens only for consumer resources. |
| ETCP_KVB_ERR_IMPOSSIBLE_TO_BIND | 7 | Impossible to bind. |

## External Bindings

External variable bindings are bindings between the variables of resources belonging to different projects. When defining external variable bindings, you need to define groups of producer variables in the producer project, then create bindings by defining groups of consuming variables from a consumer project.

You can define external bindings using the Bindings View.

**To open the Bindings View**

The Bindings View is accessed from the Solution Explorer.

- In the **Solution Explorer**, right-click the project, device, or resource, then click **Bindings**.

The Bindings View is displayed.

### See Also

Bindings View

# Bindings View

Bindings are defined using the Binding View. In the Solution Explorer, you can access the Bindings View from the contextual menus for the project and devices, as well as resources with targets that support bindings. You can open multiple instances of the Bindings View. However, each instance of the Bindings View must have a different scope.

When working in the Bindings View, you can navigate the cells using the mouse controls.

| Column | Description |
|---|---|
| Producing Groups | Displays a hierarchical view of projects, devices, resources, and defined producing groups. |
| Producing Variables | Displays the list of producer variables included in a selected producing group |
| Consuming Groups | Displays a hierarchical view of the projects, devices, resources, and consuming groups |
| Consuming Variables | Displays the list of consumer variables included in a selected consumer group |

The Bindings View toolbar contains the following:


Creates a group of producer variables


Creates a group of consumer variables

Enables selecting from a list of the most recently defined bindings. Selected bindings are displayed automatically.

In the Bindings View, you can define groups of producer variables from the resources of your project. Individual variables of a resource can belong to a one or more producing groups. You can connect a producing group to consuming groups belonging to different resources. You can also edit the contents of producing groups from their originating resource.

You can add variables from a Dictionary by dragging them into the Bindings View and place them Producing Variables column. You can also drag variables from the Variable Selector into the Bindings View and place them Producing Variables column. System variables and those belonging to function block instances cannot be used in bindings.

Consuming groups are automatically updated to reflect changes made to producing groups. For example, deleting a producer variable automatically removes the associated consumer variable.

You can delete producing groups having producer variables used in bindings. However, deleting such producing groups causes the bound variables to display errors.

In the Bindings View, you can define a group of consumer variables from local or external projects by identifying the project, the resource, and the producing group. When creating a consuming group, a link to an existing producing group is created. The Bindings View displays the linked producer and consumer variables at the same level within their respective columns. The link between producer and consumer variables flows in one direction, from producer to consumer. You can also choose to use binding error variables.

When linking structure and array variables, these must have the same byte order and alignment. Variables having the elementary data types are automatically adjusted for differences in byte order and alignment.

Using the Bindings View, you can edit the contents of consuming groups. You can also delete groups of consumer variables.

For consuming and producing variable groups and external bindings, indicates errors that can occur for different situations such as the following:

| | |
|---|---|
| Variable groups | - The project of a variable group cannot be found |
| | - The variable group cannot be found within the specified project |
| | - A conflict exists between the consumer and producer resources |
| | - One of the bound variables no longer exists |
| External bindings | - The variable used in the binding no longer exists |
| | - The project holding the variable cannot be accessed |

### To define a group of producer variables

You define a producing group by adding producer variables within the Producing Variables column and by dragging them from the Dictionary. When defining a producing group, the consuming group column is empty.

1. In the **Producing Groups** column, select the required resource, then click 

.

   The producing group is displayed in the Producing Groups column.

2. Click the producing group.

3. To add producer variables, do one of the following:

   - In the **Producing Variables** column, click 

, then select the required variable from the drop-down combo-box.
   - From the **Dictionary**, select the required variable, then drag the selection indictor (  ) into the **Producing Variables** column, placing it on 

.

4. To add subsequent variables, from the **Producing Variables** column, click below the existing producer variable, then repeat step 3.

### To edit an existing group of producer variables

You can edit an existing producing group by replacing or deleting its variables individually. Note that local producing groups are modifiable. However, the properties of external producing groups are not editable using the Bindings View.

1. In the **Producing Groups** column, click the producing group.

2. In the **Producing Variables** column, click the variable, then do one of the following:

- To replace the variable, right-click the variable, click **Edit**, then select another variable from the drop-down combo-box.
- To delete the variable, right-click the variable, then click **Delete**.

**To delete a producing group**

You can delete producing groups from the Bindings View.

- In the **Producing Groups** column, right-click the producing group, then click **Delete**.

The producing group is permanently deleted.

**To define a group of consumer variables**

You can define a group of consumer variables by accessing the consuming resource of a project. When defining consumer variables, a link is established between the producing resource and the consuming resource.

**1.** In the **Consuming Group** column, select the required resource, then click 

.

**2.** From the **Add consuming group** dialog:

   **a)** Click 

, select the project library file containing the required consumer variables, then click **Open**.

   **b)** In the **Resource Number** field, select the required resource number from the drop-down combo-box.

   The drop-down combo-box contains the resource numbers of the resources within the library file selected in step 2a.

   **c)** In the **Group ID** field, select the group ID from the drop-down combo-box.

   The drop-down combo-box contains the group IDs for the producing groups located the Producing Groups column of the Bindings View.

   The Group Comment field displays the name of the producing group corresponding to the group ID selected.

d) In the **Binding Error Variables** section, select the binding error variable from the drop-down combo-box (optional), then click **OK**.

3. In the **Consuming Groups** column, click the consuming group.

4. To add variables to the **Consuming Variables** column, do one of the following:

   - Click …, then select the required variable from the drop-down combo-box.
   - From the **Dictionary**, select the required variable, then drag the selection indictor ( ) into the **Consuming Variables** column, placing it on … .

5. To add more consumer variables, in the **Consuming Variables** column, click below the existing variable, then repeat step 4.

**To edit an existing consuming group**

You can edit an existing consuming group by individually replacing or deleting the consumer variables within the Consuming Variables column.

1. In the **Consuming Groups** column, click the consuming group.

2. In the **Consuming Variables** column, select the variable, then do one of the following:

   - To replace the variable, right-click the variable, click **Edit**, then select another variable from the drop-down combo-box.
   - To delete the variable, right-click the variable, then click **Delete**.

**To delete a consuming group**

- In the **Consuming Groups** column, right-click the consuming group, then click **Delete**.

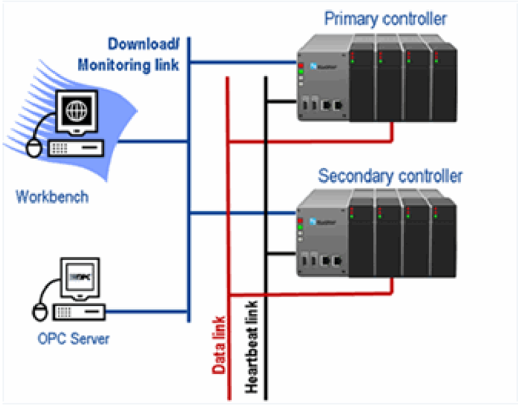
The consuming group is permanently deleted.

## See Also

Bindings

# Failover Mechanism

The failover mechanism is a secondary backup operational mode which is an essential part of mission-critical systems where availability is without compromise. The failover mechanism provides a more fault-tolerant industrial application. When a failure occurs in the primary components of an industrial system or when a scheduled down time is performed, the functions of the industrial system are backed by the failover mechanism running on a secondary industrial system.




The failover mechanism consists of two devices (primary and secondary) executing the same application. At the end of each cycle, both devices synchronize their data by exchanging CRCs (data link). This parallel execution allows a bumpless switch-over between both devices. During execution, only one device is active. This device updates the output while the other device remains on standby. A heartbeat signal is sent between devices to ensure availability.

The failover mechanism sequence is executed as follows:

**1.** Download the application onto the active device.

The workbench downloads onto active devices. When the primary is not active, the workbench automatically switches to the secondary.

2. The application is automatically transferred to the standby device.

   The workbench always performs only one download to the active devices. When a device is in error and is being replaced, the active device downloads the application to the standby device turned active upon reconnecting.

3. Both devices execute the same application code in parallel.

4. Before each execution cycle, input values are transferred from the primary device to the secondary device.

   The performance of the failover mechanism is directly affected by the quantity of input values transferred during data synchronisation between the primary device and the secondary device.

5. At the end of each cycle, a check sum mechanism runs to ensure the integrity of the data and results. In case of a mismatch, the full data space of the active device is transferred to the standby device.

6. The active device generates the heartbeat, then verifies that the standby device can receive it before sending the heartbeat across the link.

7. In case of failure on the active device, the standby device becomes the active one and controls the process.

When the standby device (secondary device) does not detect activity during the time determined in the *FailoverHeartbeatDeactivationTimeMs* property, it becomes active. If the now active secondary device detects activity, it means the primary device is also active. You can only have one active device, so the primary active device is forced into standby for the duration of time specified in the *FailoverHeartbeatDeactivationTimeMs* property.

When a device contains multiple resources, each resource is executed independently and performs data synchronization based on its defined cycle time. To optimize data synchronization, a high priority resource should use a minimal amount of memory space.

The following system variables provide information for a failover system:

| | | | |
|---|---|---|---|
| _SYSVA_FO_ISENABLE | BOOL | READ | Activation status of the failover system for the device |

| | | | |
|---|---|---|---|
| _SYSVA_FO_ERRCODE | UDINT | READ | Operational status of the failover system. The associated bits of the variable represent the following errors:<br>0 = No error<br>1 = The standby device failed to read the heartbeat from the active device<br>2 = The devices are unable to establish communication across the data link<br>3 = System mismatch between both devices making up the failover mechanism - each device has a different type<br>4 = Capability mismatch between the devices making up the failover mechanism - each device supports different features |
| _SYSVA_FO_ISPRIMARY | BOOL | READ | Indication of whether the active device is the primary for the failover system |
| _SYSVA_FO_ISACTIVE | BOOL | READ | Indication of whether the device is active |
| _SYSVA_FO_DATASYNCTIME | UDINT | READ | Time between data synchronizations from the active to the standby device |
| _SYSVA_FO_DATASYNCCNT | UDINT | READ | Number of full data synchronizations since starting the target (available from ISaGRAF 5.40 targets) |
| _SYSVA_FO_HBEATSYNCTIME | UDINT | READ | Time between heartbeat synchronization from the standby to the active device |

The failover mechanism executes only the most recent application code. When downloading this code to the active device, the compilation date is used to determine whether the downloaded code is older than the existing code on the device. To execute an older version of an application, you must recompile the code before downloading to the active device.

Since the failover mechanism executes the same application code on the active and standby devices, the code must contain conditions that can be executed by both devices. For example, the __SYSVA_FO_ISACTIVE system variable can only be true for the active device. Therefore, when __SYSVA_FO_ISACTIVE is true, the standby device does not execute the following code. As a result, a CRC mismatch occurs and the failover mechanism requires a full data synchronization, causing extended cycle time.

```
if (__SYSVA_FO_ISACTIVE = TRUE) THEN
  (...)
end_if;
```

### See Also

Limitations for Failover Mechanisms

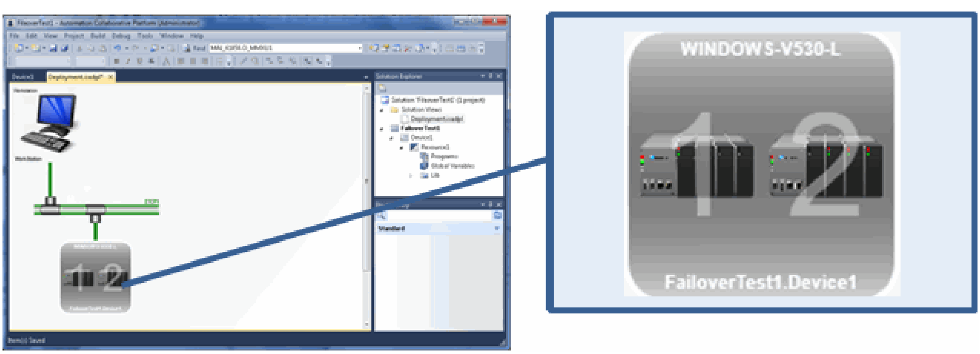
# Configuring a Failover Mechanism

The workbench provides tools where engineers see one device during the programming phase and both devices during monitoring and debugging. To emphasize the concept of primary, secondary, and active devices, the workbench always remains focussed on active devices:

- Only one project with one device to manage
- Only one compilation to perform
- Only one download to perform to the active devices
- Only one connection to establish to the active devices

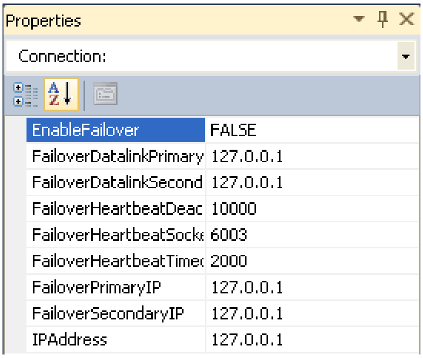
Failover mechanisms for devices are available for projects created using the failover project template and in which was imported a failover *.TDB file.

A project with a failover mechanism has one device representing two devices. In the deployment view, the failover mechanism is represented as a standalone device with the graphical representation showing two devices where one represents the primary device and the other represents the secondary device.

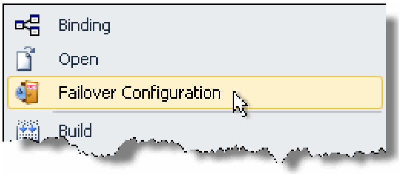


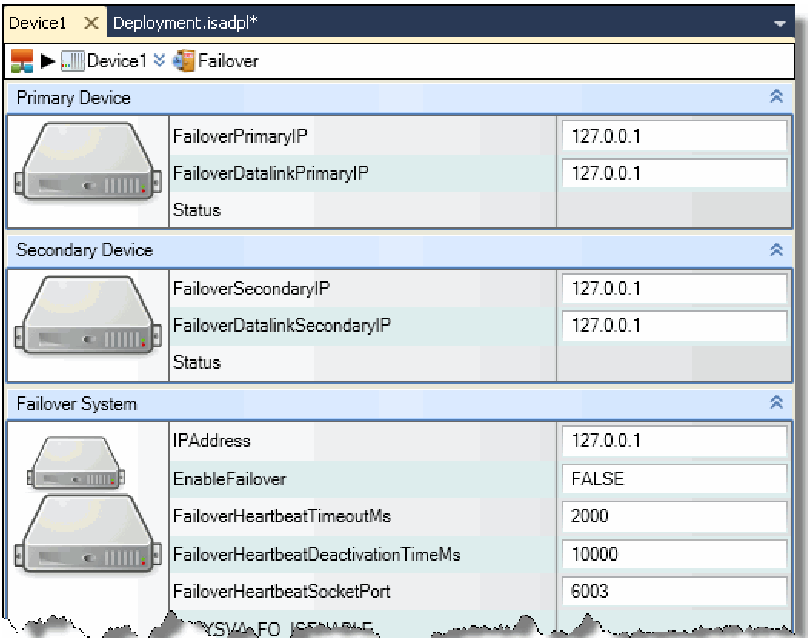

From the deployment view, you can configure a failover mechanism using one of two methods:

- In the properties window for a connection link accessed by selecting the connection link between the device and the ETCP network, then right-clicking and then clicking **Properties** (or pressing the F4 key).




- In the failover configuration graphical environment accessed by selecting a device, then right-clicking and then clicking **Failover**.






Configuring a failover mechanism consists of setting parameters defined by the OEM where some may differ depending on whether it uses an Ethernet, serial, or another link type. Since a failover mechanism requires Ethernet communication for the dialog between devices and the workbench, the configuration is similar for all implementations:

**Primary Device**

| | | |
|---|---|---|
| FailoverPrimaryIP | STRING | IP address of the primary device on the network used for communication with the workbench. This value is the same as IPAddress. |
| FailoverDatalinkPrimaryIP | STRING | IP address of the primary device on the data link used for communication between the active and standby devices |

**Secondary Device**

| | | |
|---|---|---|
| FailoverSecondaryIP | STRING | IP address of the secondary device on the network used for communication with the workbench |
| FailoverDatalinkSecondaryIP | STRING | IP address of the secondary device on the data link used for communication between the active and standby devices |
| **Failover System** | | |
| EnableFailover | BOOL | Activates the failover mechanism for the device |
| FailoverHeartbeatTimeoutMs | UDINT | Time delay, in milliseconds, before the standby device takes over from the active device |
| FailoverHeartbeatDeactivationTimeMs | UDINT | Time delay, in milliseconds, before the standby device takes over from the active device following a loss of communication. This delay does not apply for actual breakdowns of the active device. |

You can configure the timeout of the data synchronization for a resource using the following extended parameter. This is accessed from the properties for the individual resources.

| | | |
|---|---|---|
| **Extended Parameters** | | |
| FailoverDatalinkTimeoutMs | UDINT | Maximum time, in milliseconds, that the active device waits for a reply from the standby device before resuming the control cycle. When the timeout is reached, the communication is re-initialized. |

## See Also

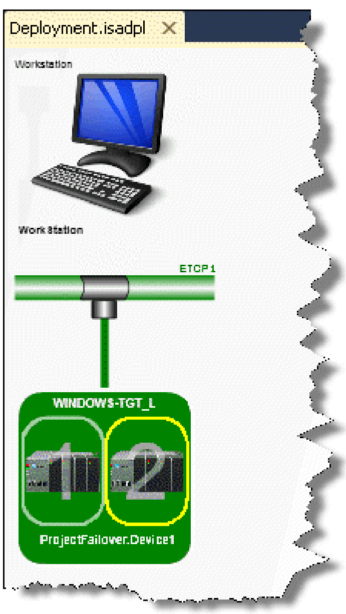
Monitoring the Failover Mechanism
Implementing Failover Mechanisms on a Windows Platform
Limitations for Failover Mechanisms

# Monitoring the Failover Mechanism

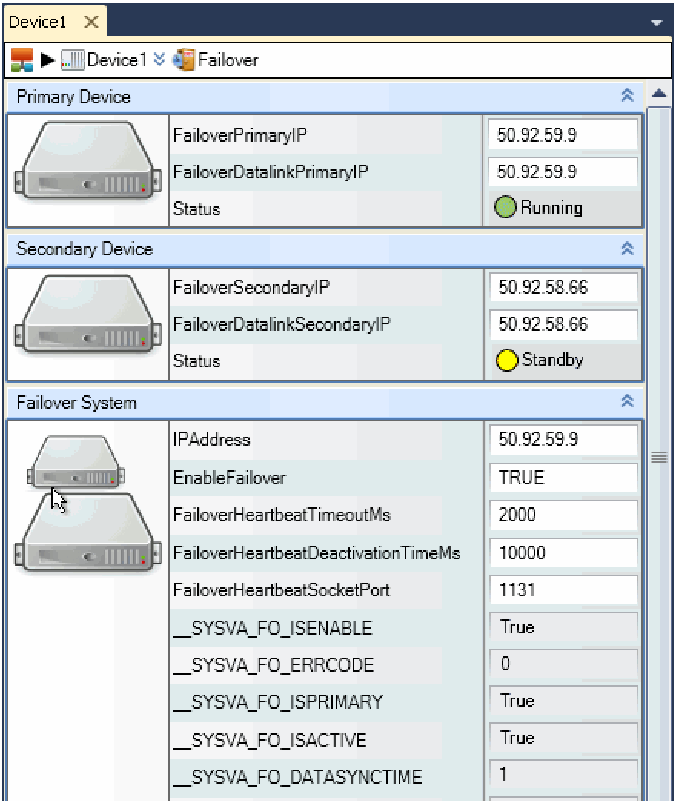
You can access information for the failover mechanism from the Device and Deployment views while running online or debugging. From the navigation view, you can access the failover mechanism information for the primary device, the secondary device, general system, and system variables.

From the deployment view, the real-time status of the primary and secondary devices is displayed using color:

| | |
|---|---|
| Green | Device is active |
| Yellow | Device is on standby |
| Red | Device is in error |




While running online, you can also monitor the failover mechanism including system variables from the Device View. The status color displays are similar to those for the Deployment View.




## See Also

Failover Mechanism

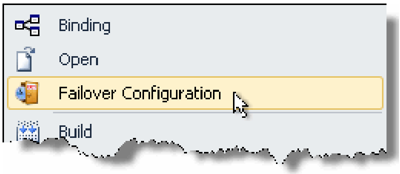
Implementing Failover Mechanisms on a Windows Platform

# Implementing Failover Mechanisms on a Windows Platform

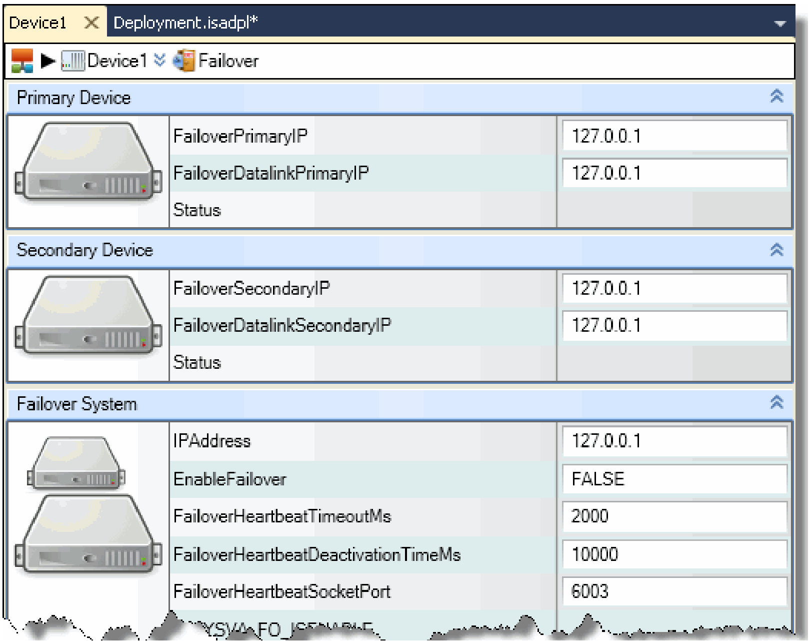
You can only implement failover mechanisms for systems running with targets having the failover feature.

1. Create a project using the Win32_L_Failover_TPL template.

2. From the Deployment View, right-click the device for which to define a failover mechanism, and then click **Failover Configuration**.




3. In the Device view, set EnableFailover to TRUE, then define the remaining properties for the primary device, secondary device, and failover system. When not using a separate data link connection for the devices, the respective data link properties must use the same IP address as the corresponding failover IP address properties.




**4.** Set up the **ISaGRAF** targets for the primary and secondary devices.

**a)** From the PRDK, install the **ISaGRAF** targets on the respective computers.

**b)** From a command prompt (Start menu > All Programs > Accessories > Command Prompt), launch the targets using the following command lines to identify which will run the primary and secondary systems.

| Primary or Secondary | Window (MonoTask) | Windows (MultiTask) |
|---|---|---|
| Primary system | ISa.exe -PR | ISaGRAF.exe -PR |
| Secondary system | ISa.exe -SE | ISaGRAF.exe -SE |

**Note:** Before proceeding to download the application, make sure firmware (definition of C functions, I/O drivers, etc.) is identical for the primary and secondary systems.

5. Build the application and perform a download.

   The application is downloaded onto the active device and automatically duplicated on the standby device. When launching the failover mechanism, the active device is the primary and the standby device is the secondary.

6. Switch the application to run online by choosing Start Debugging from the Debug menu.

7. From the Device View, note the status information for the primary and secondary devices as well as the values for the failover system variables.

**See Also**

Failover Mechanism
Monitoring the Failover Mechanism

## Limitations for Failover Mechanisms

While using a failover mechanism, systems have no particular limitations. Most workbench features are supported while the code behind the failover mechanism is highly portable enabling it to run on any hardware platform that meets the requirements for ISaGRAF firmware. The main limitations are the following:

- Failover requires using Ethernet communication between the workbench and firmware
- The OPC server and OPC gateway support automatic switching only on ETCP (Ethernet TCP/IP)
- Failover supports the bindings feature from the ISaGRAF 5.40 targets
- Failover does not yet support the interrupt feature enabling to control the moment of execution of cyclic programs (ST, LD, FBD, and SAMA)
- Failover does not support sending custom files, placed in the *To Download* folder of a device directory, to the target when downloading onto the target platform

### See Also

Failover Mechanism

# IEC 61499 Language

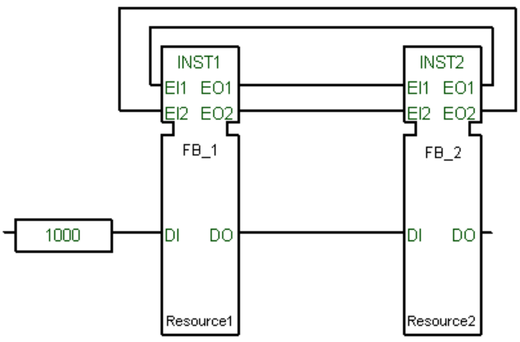
The IEC 61499 language is a distribution method enabling the distribution of individual IEC 61499 function blocks belonging to an IEC 61499 program across multiple resources. The IEC 61499 standard function blocks are available with the IEC 61499 library.

In an IEC 61499 project, you create programs into which you insert IEC 61499 basic function blocks and composite function blocks.

**Note:** The IEC 61499 implementation is based on the *Function blocks - Part 1: Architecture* and *Function blocks - Part 2: Software Tools Requirements* documents available from the ANSI webstore.

# IEC 61499 Program Main Format

In IEC 61499 programs, IEC 61499 function blocks are distributed across resources. Inputs and outputs from these function blocks distributed between resources are connected with bindings. These bindings are automatically created. Inputs and outputs between function blocks must respect data types. For IEC 61499 function blocks, identifiers can only be literals or defined words.

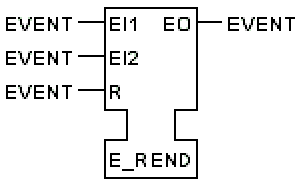
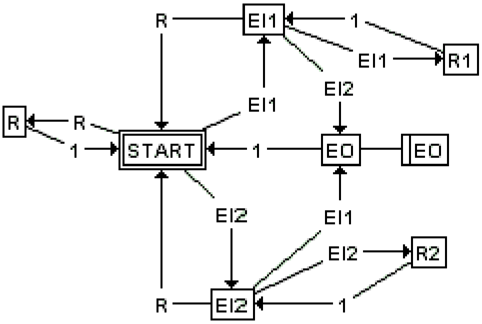



Insertion of an IEC 61499 basic function block or composite function block into a program is enabled following its creation in the project library.

When splitting an IEC 61499 function block output to connect with two inputs, **ISaGRAF** automatically performs the split. Therefore, use of the E_SPLIT function block is not required.

Resources having an instance of an IEC 61499 function block display the IEC 61499 program in which the function block is defined. Therefore, a given IEC 61499 program can appear in multiple resources. Bindings between resources are displayed in the Binding View.

IEC 61499 function blocks are distinct from IEC 61131-3 function blocks; An execution control chart handles the events and algorithms handle the data. IEC 61499 is implemented as either ECC (basic function blocks) or IEC 61499 FBD (composite function blocks). IEC 61499 function blocks have specific parameter types, for instance, event input and event output.

| Basic function block type | Execution control chart |
|---|---|
|  |  |





In an execution control chart, individual items represent SFC elements:

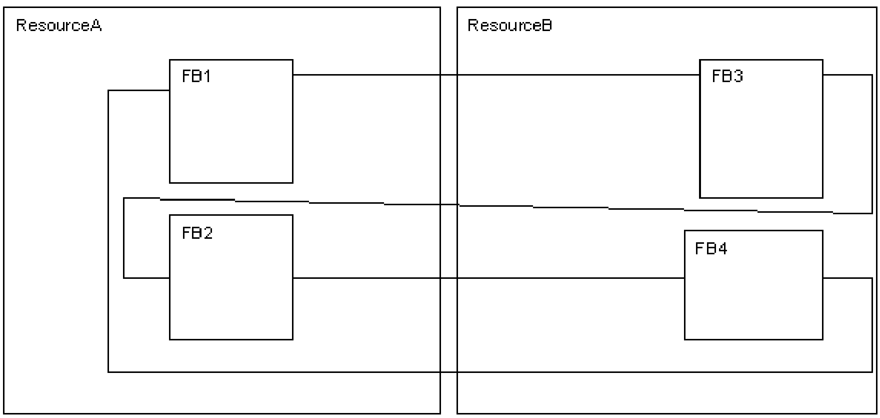
- a box with a double outline indicates the initial step
- arrows indicate transitions
- boxes with a single outline indicate steps
- double boxes indicate generated outputs. The space on the left indicates an algorithm name when one is defined.

An IEC 61499 program is built with blocks from the IEC 61499 library and user-defined IEC 61499 function blocks. The language editor displays IEC 61499 programs. The following elements are available for IEC 61499 programs:

- Function Blocks
- Variables
- Regions
- Links
- Comments

# Cycle Execution Time in IEC 61499 Programs

In IEC 61499 programs, total execution time depends on the cycle execution of multiple resources and the individual IEC 61499 function blocks. For instance, when using basic IEC 61499 function blocks, the diagram consisting of FB1, FB2, FB3, and FB4 completes execution after a minimum of four complete cycles of each resource. Each resource cycle executes the steps of an event control chart until reaching a false transition.

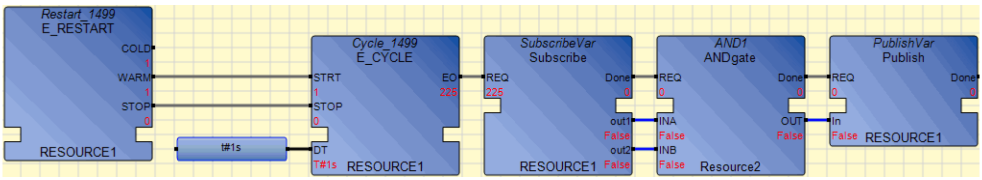



The following formula expresses the minimum total time required to execute one cycle of the above program:

Total time = cycle time (ResourceA) X 2 + cycle time (ResourceB) X 2

# Debugging IEC 61499 Programs

When debugging IEC 61499 programs, you can monitor the output values of elements. These values are displayed using color, numeric, or textual values according to their data type:



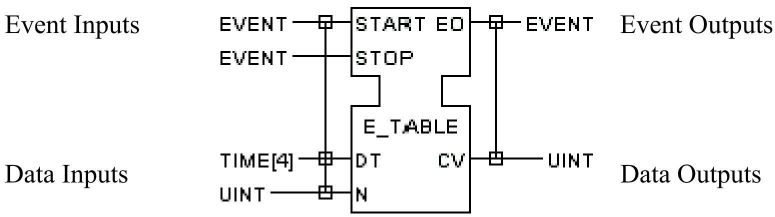

- Output values of boolean type are displayed using color. The output value color continues to the next input. When the output value is unavailable, boolean elements remain black. The colors are red when True and blue when False.

- Output values of SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, and STRING type are displayed as a numeric or textual value in the element. When the output is a structure type, the displayed value is the selected member.

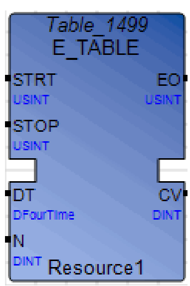
When the output value for a numeric or textual value is unavailable, the *WAIT* text is displayed in the output label. Values are also displayed in the corresponding dictionary instance.

# IEC 61499 Function Block Main Format

IEC 61499 function blocks are made up of event inputs and outputs as well as data inputs and outputs:




An IEC 61499 function block is represented by a box having an upper section representing the event control chart and a lower section representing the data process. Three names are indicated in the block: the instance name at the top, the function block name just below, and the resource in which it is declared at the bottom.




The parameters for standard IEC 61499 function blocks are displayed in the local variables instance with their equivalent direction attribute. The E_TABLE function block shows the following directions for its inputs and outputs:

E_TABLE-VAR

| Name | Data Type | Dimension | String Size | Initial Value | Direction | Attribute | Comment |
|---|---|---|---|---|---|---|---|
| DTime | DFourTime | | | ... | VarInput | Read | time interval (ms) |
| N | DINT | | | | VarInput | Read | occurances |
| START | USINT | | | | EventInput | Read | start event |
| STOP | USINT | | | | EventInput | Read | stop event |
| CV | DINT | | | | VarOutput | Write | current value |
| EO | USINT | | | | EventOutput | Write | Event Output |
| CTRL | E_TABLE_CTRL | | | ... | Var | Read/Write | |
| DLY | E_DELAY | | | ... | Var | Read/Write | |
| LocalEventInput_START | LocalEventInput | | | ... | Var | Read/Write | |
| LocalEventInput_STOP | LocalEventInput | | | ... | Var | Read/Write | |
| latch1 | E_LATCH_DINT | | | ... | Var | Read/Write | |

- DT and N data inputs having the VarInput direction
- START and STOP event inputs having the EventInput direction.
- CV data output having the VarOutput direction
- EO event output having the EventOutput direction
- LocalEventInput_START and LocalEventInput_STOP are created for the START and STOP arguments of the function block

## See Also

Basic IEC 61499 Function Blocks
Composite IEC 61499 Function Blocks

# Basic IEC 61499 Function Blocks

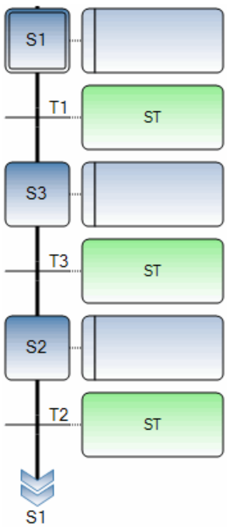
Basic IEC 61499 function blocks are defined using SFC elements to develop their execution control chart.

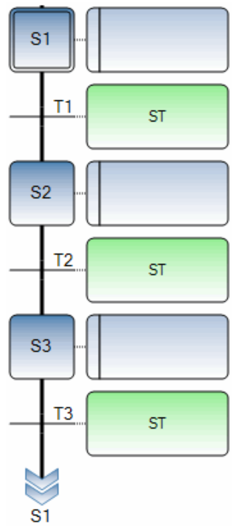
- Steps (States )
- Transitions
- Sequence Controls
- Jumps to Steps

**ISaGRAF** automatically implements the WITH qualifier to ensure synchronization between data inputs and event inputs.

When inserting steps and transitions, these are assigned a default naming convention including numbering. For steps, the default naming is S*n* where S indicates a step and *n* indicates the numbering for the step. For transitions, the default naming is T*n* where T indicates a transition and *n* indicates the numbering for the transition. You can rename steps and transitions. However, when renaming steps and transitions using the default naming convention and changing only the numbering, you can renumber these steps and transitions to a numbering scheme starting from top to bottom, then from left to right.

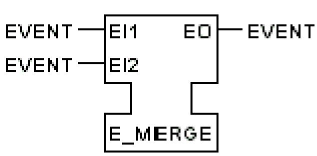
| **Before Renumbering** | **After Renumbering** |
| --- | --- |

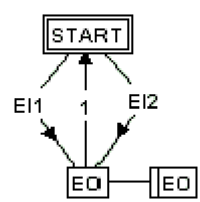


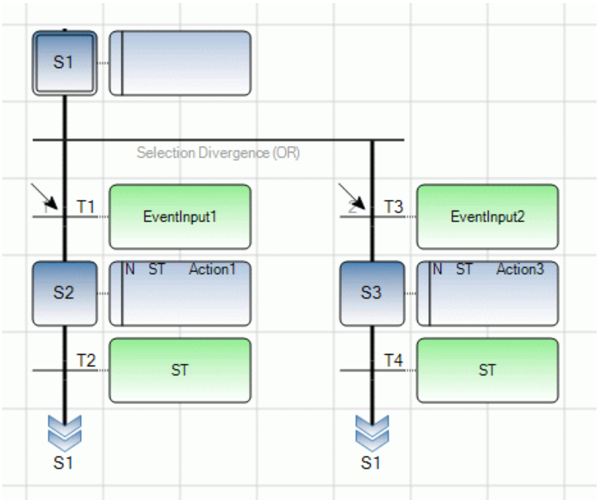

The following example shows the E_Merge function block made up of an initial step, two transitions, and a step:

| E_MERGE | Event Control Chart |
| --- | --- |



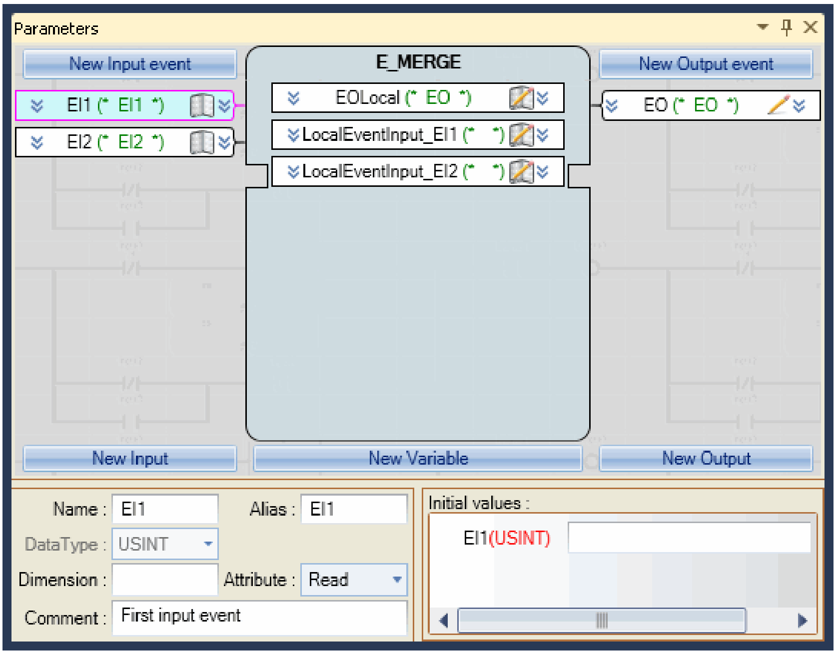



**SFC Equivalent**




```
(* gets the events *)
LocalEventInput_EI1(EI1);
LocalEventInput_EI2(EI2);

(*tests for an event *)
LocalEventInput_EI1.Trigger or
LocalEventInput_EI2.Trigger;

(*processes the algorithm *)
EOLocal:=EOLocal+1;
EO:=EOLocal;
```

When defining the parameters of basic IEC 61499 function blocks, for each argument having the event input type, a local variable having the Local_ prefix is automatically created. Also, for each argument having the event output type, a defined word having the Generate_ prefix is automatically created. From actions, you can call defined words for these event outputs.




Typically, an event transition in an SFC diagram is made up of LD statements.

**To add parameters to a basic function block**

1. In the Solution Explorer, locate the basic function block by expanding the Function Blocks section.
2. Right-click the block, then choose Parameters from the contextual menu.

   A graphic representation of the function block is displayed in the Parameters window.
3. To add a new input, output, or variable, click the respective option below the function block representation and define the arguments for the block.

**To renumber steps and transitions**

Renumbering ignores steps and transitions using a naming convention other than the default S*n* for steps and T*n* for transitions.

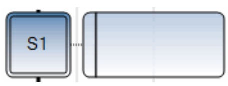
1. Open the basic IEC 61499 function block for which to renumber the steps and transitions.
2. From the Tools menu, choose **Multi-language Editor**, then **Renumber Steps and Transitions**.

# States

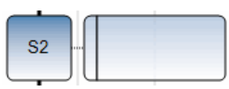
For an IEC 61499 ECC, an initial step corresponds to an execution control initial state (EC initial state) and a step corresponds to an execution control state (EC state).

Intial steps express the initial situation of an SFC program. Whereas, steps are placed throughout an SFC program. An SFC program must contain at least one initial step. Initial steps and steps are referenced by a name, written in their square symbol. This information is the level 1 of the step.

An initial step has a double bordered graphic symbol.

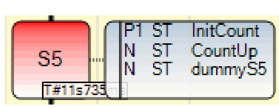



A step is represented by a single square.

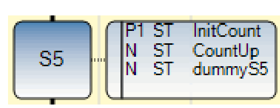



At run time, a token indicates that the step is active. For initial steps, a token is automatically placed in each when the program is started.

Active Step




Inactive Step




Steps have attributes. These can be used in any of the other languages.

StepName.x activity of the Step (Boolean value)
StepName.t activation duration of the Step (time value)

(where StepName is the name of the step)

Activity of a step is an attribute of a step which is activated by an SFC token.

For SFC function blocks, when reading a child active step or duration from a father:

ChildName.__S1.x activity of the Step (Boolean value)
ChildName.__S1.t activation duration of the Step (time value)

(where ChildName is the name of the child. Note that S1 is preceded by two underscore (_)characters)

**To insert an initial step**

- From the Toolbox, drag the initial step element into the language container.

The initial step is displayed in the language container.

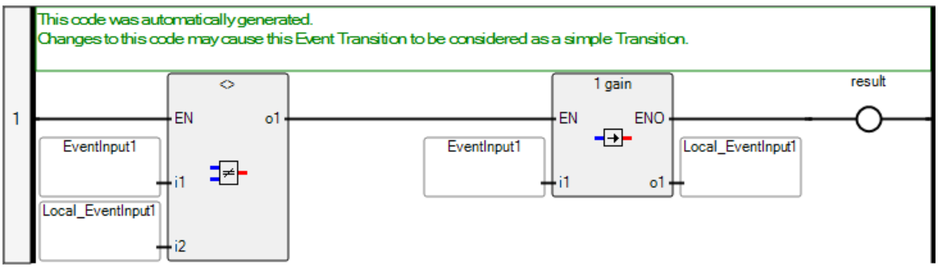
**To insert a step**

- From the Toolbox, drag the step element into the language container.

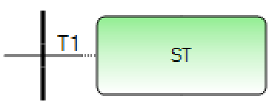
The step is displayed in the language container.

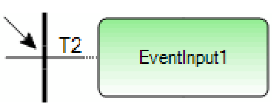
## Transitions

For an IEC 61499 ECC, transitions and event transitions correspond to an execution control transition (ECC transition). Event transitions are transitions programmed to trigger from an event input. Event transitions are pre-programmed in LD.




Transitions are represented by a small horizontal bar that crosses the connection link. Event transitions are displayed with an additional arrow pointing towards the connection link. Each transition is referenced by a name, displayed next to the transition symbol.




### To insert a transition

- From the Toolbox, drag the transition element into the language container.

The transition is displayed in the language container.

### To insert an event transition

- From the Toolbox, drag the event transition element into the language container.

The event transition is displayed in the language container.

## Sequence Controls

Sequence controls are divergences or convergences. These elements adjust automatically to the context of the SFC diagram. For instance, the editor automatically inserts the type of sequence control required according to the elements at the insertion point. Moreover, when adding a parallel element below a sequence control, the sequence control automatically branches out to the added element. Also, when a sequence control is placed erroneously within a diagram, the editor displays it as red.

- Selection Divergences, a multiple link from a step to multiple transitions
- Selection Convergences, a multiple link from multiple transitions to a single step
- Simultaneous Divergences, a multiple link from a transition to multiple steps
- Simultaneous Convergences, a multiple link from multiple steps to a single transition

Divergences are multiple links from one SFC element (step or transition) to multiple SFC symbols. Convergences are multiple connections from more than one SFC symbol to one other symbol.

When inserting a sequence control, the type is determined logically according to the number of SFC elements of a same type (whether multiple) located initially above then below the control.

**To insert a sequence control**

- From the Toolbox, drag the sequence control to the desired location in the language container.

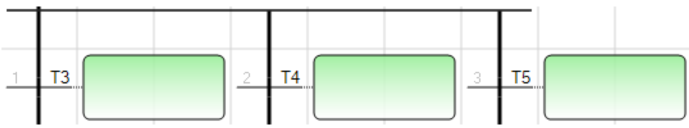
The sequence control element is displayed in the language container.

## Selection Divergences

A selection divergence (OR) is a multiple link from one step to multiple transitions. The selection divergence enables an active token to pass into one of a number of branches.

Conditions attached to the different transitions at the beginning of a selection divergence are not implicitly exclusive. Exclusivity of transitions is defined by the priorities set to those transitions following the divergence.

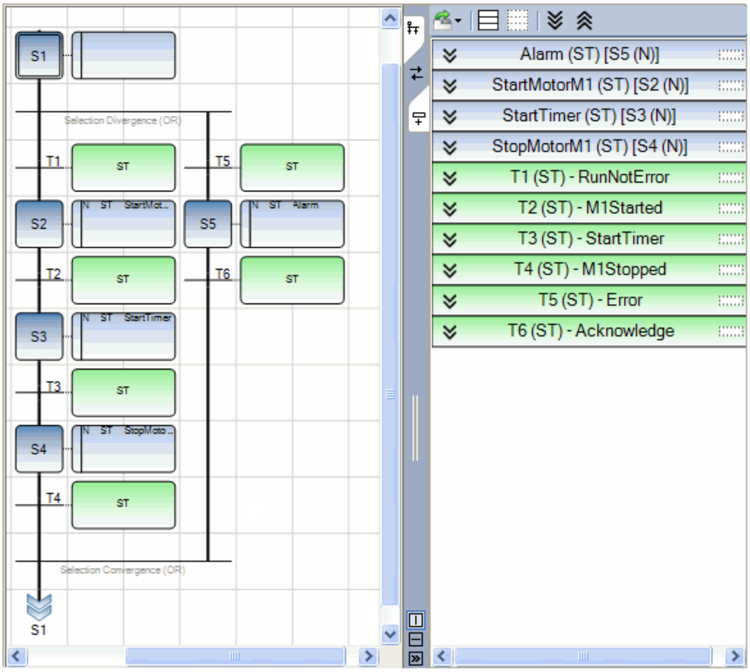
Selection divergences are represented by single horizontal lines.




The first transitions following a single divergence are set in a group to define their priority of execution. The workbench automatically assigns the priority of transitions, displayed on the left, in the order of creation of the divergence branch. You can specify a different priority for a transition in the properties. The possible priority values range from 1 to 255.

### Example

(* ECC with selection divergence and convergence *)

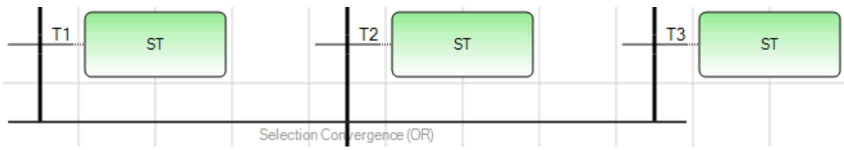



## See Also

Selection Convergences
Simultaneous Divergences

## Selection Convergences

A selection convergence (OR) is a multiple link from multiple transitions to a single step. Selection convergences are generally used to group branches which were started using selection divergences. Selection convergences are represented by single horizontal lines.

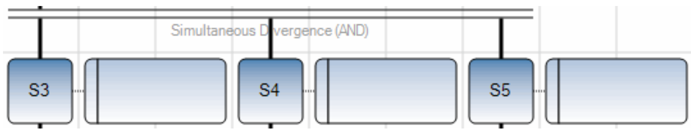



### See Also

Selection Convergences
Simultaneous Convergences

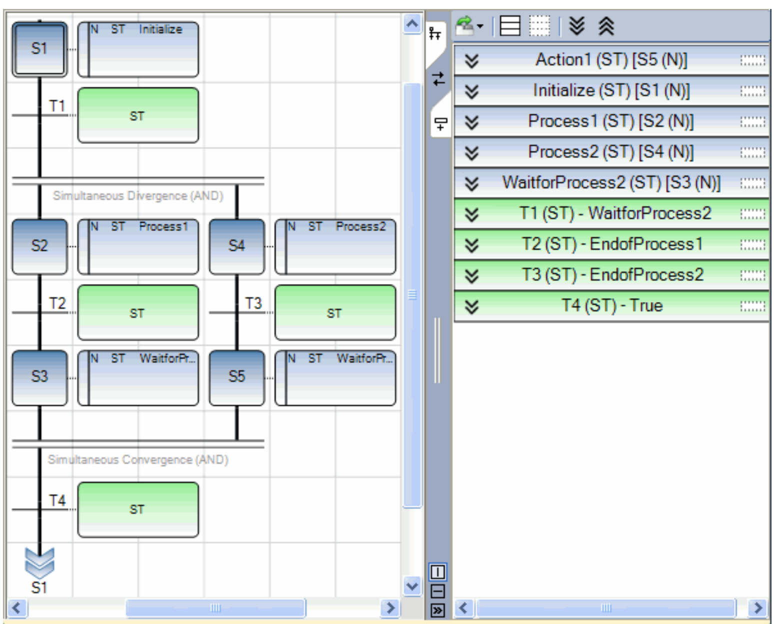
## Simultaneous Divergences

A simultaneous divergence (AND) is a multiple link from one transition to multiple steps. A simultaneous divergence corresponds to parallel operations of a process. Simultaneous divergences are represented by double horizontal lines.




### Example

(* Program with simultaneous divergence and convergence *)

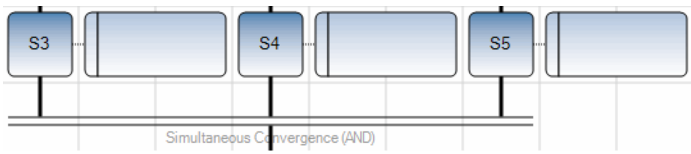



### See Also

Simultaneous Convergences
Selection Divergences

## Simultaneous Convergences

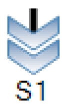
A simultaneous convergence (AND) is a multiple link from multiple steps to a single transition. Simultaneous convergences are generally used to group branches which were started using simultaneous divergences. Simultaneous convergences are represented by double horizontal lines.




**See Also**
Simultaneous Divergences
Selection Convergences

## Jumps to Steps

Jump symbols are available to indicate a connection link from a transition to a step, without having to draw a connection line. The jump symbol must be referenced with the name of the destination step. A jump symbol cannot represent a link from a step to a transition.



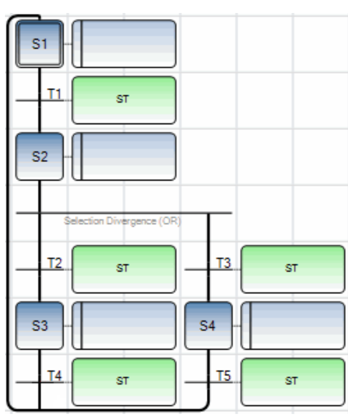

Jump to Step S1

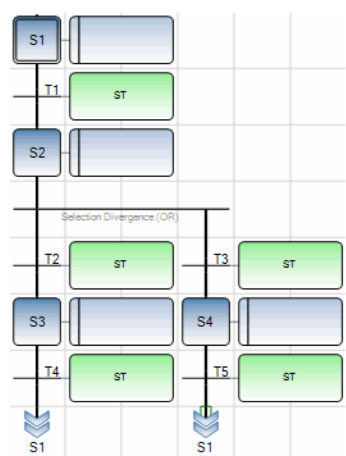
**To insert a jump to a step**

1. From the Toolbox, drag the jump element into the language container and place it directly below the existing transition.
2. In the language container, click the jump element.
3. In the drop-down combo-box, click the desired step.

### Example

The following charts are equivalent. The chart on the left uses links to return from the bottom to the top of the chart while the chart on the right uses jumps to return to the top of the chart.



## Coding Action Blocks for Steps

Action blocks are operations executed when a step is active. Steps can contain multiple action blocks of the same or different type. You add action blocks to the level 1 of a step. Depending on the action block type, you may need to program the level 2 for the block. You program level 2 code for an action block in a level 2 window, displayed to the right of the POU. The available action block types are the following:

- Boo where the action block name is automatically associated to Boolean variable selected from the variable selector. Possible qualifiers are Action (N), Reset (R), and Set (S).
- LD where you program an LD diagram in the level 2 window. Possible qualifiers are Action (N), Reset (R), Set (S), Pulse on Deactivation Action (P0), and Pulse On Activation Action (P1).
- SFC where the action block name is automatically associated to the SFC child. Possible qualifiers are Action (N), Reset (R), and Set (S).
- ST where you define ST code in the level 2 window. Possible qualifiers are Action (N), Reset (R), Set (S), Pulse on Deactivation Action (P0), and Pulse On Activation Action (P1).
- Event Action where an ST action is automatically generated following the selection of an event output. The ST action is named using the event output name preceded by the *Generate_* prefix.

Individual SFC steps are executed in the following order:

1. Step activation - beginning when the previous transition is cleared. During this period, defined action blocks are executed in the order of appearance.
2. Step cycle - beginning when the step becomes active and ending when the step completes deactivation. During this period, defined action blocks are executed in the order of appearance.
3. Step deactivation - ending when the following transition becomes active. During this period, defined action blocks other than Boolean (Boo) action blocks having the N qualifier are executed in the order of appearance. Boolean (Boo) action blocks are executed after all other action blocks.

**To add action blocks to steps**

1. Select the step for which to define operations.

2. Right-click the step, then from the contextual menu choose Add, then the required action block type.

3. Specify the required properties for the action block from the Properties window by clicking the action block definition on the step.

   a) To rename the action block, type the required text in the Name field.

**Note:** The names for Boo and SFC action blocks are automatically associated to their respective assignation (Boolean variable or SFC child).

   b) To specify the qualifier for the action block, choose the required type in the Qualifier field.

   c) To include a comment, type the required text in the Comment field.

4. For a Boo action block, double-click the action block name, then from the Variable Selector, select the variable for use in the block.

5. For an ST or LD action block, access the level 2 for the block by double-clicking the action block name on the step, then program the required level 2 operations in the level 2 window displayed to the right of the POU.

6. For an Event Action block, select an event output from the Select Output Event window.

**To rearrange the order of action blocks for a step**

1. On the step, select the action block to displace.

2. Right-click the action block, the choose **Move Up** or **Move Down** from the contextual menu.

**To delete an action block**

1. On the step, select the action block to remove.

2. Right-click the action block, the choose **Delete** from the contextual menu.

## Boolean Actions

Boolean Actions assign a Boolean Variable with the activity of the Step. The Boolean Variable can be a VarInput or VarOutput variable. It is assigned each time the Step activity starts or stops. This is the meaning of the basic Boolean Actions:

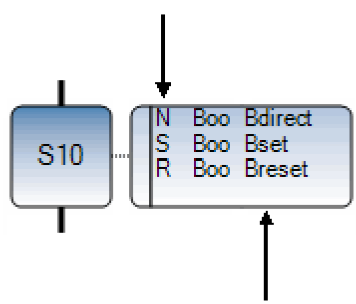
**N on a Boolean Variable** assigns the Step activity signal to the variable

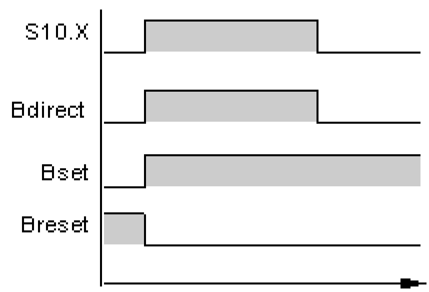
**S on a Boolean Variable** sets the variable to TRUE when the step activity signal becomes TRUE

**R on a Boolean Variable** resets the variable to FALSE when the step activity signal becomes TRUE

The Boolean variable must be VarInput or VarOutput. The following SFC programming leads to the indicated behavior:

Qualifier

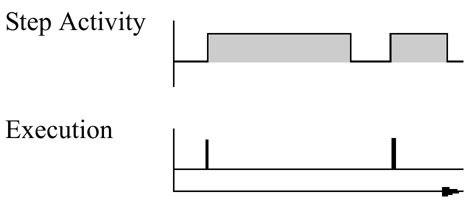



Variable Name (S10.X is the activity of Step S10)

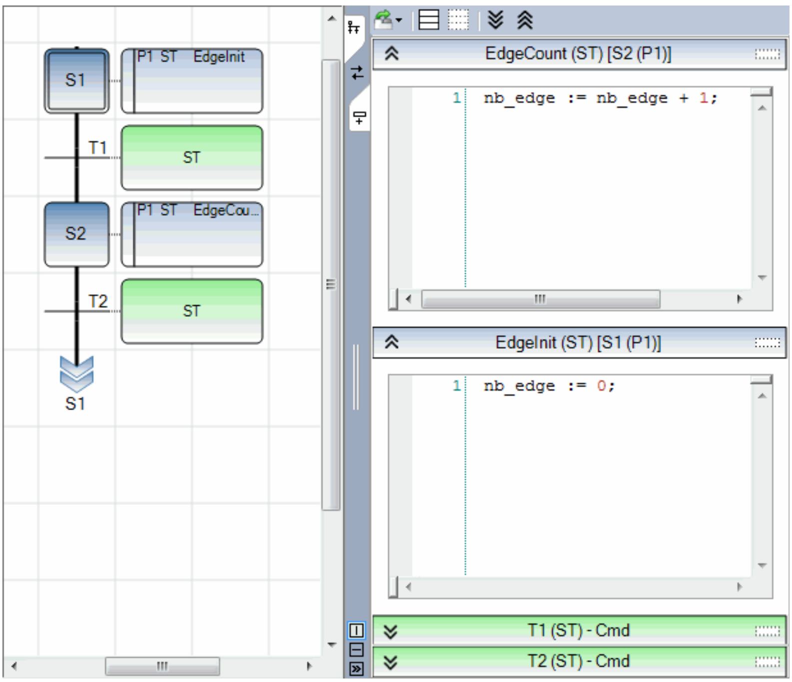
## Pulse Actions

A pulse action is a list of instructions, which are executed only once at the activation of the Step: P1 Qualifier, or executed only once at the deactivation of the Step: P0 Qualifier. Instructions are written using the ST or LD syntax. The following shows the results of a pulse Action with the P1 Qualifier:



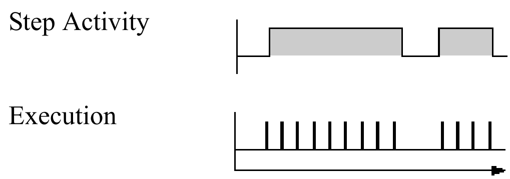
## Example

In the following program, step S1 is assigned an ST action named EdgeInit having the P1 qualifier and S2 is assigned an ST action named EdgeCount having the P1 qualifier. The code for these actions is programmed in their respective level 2 window.



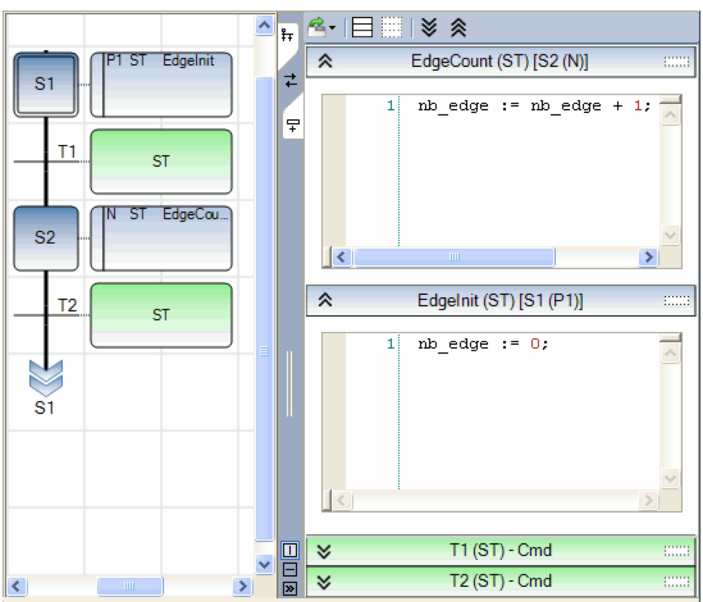
## Non-Stored Actions

A non-stored (normal) action is a list of ST or LD instructions which are executed at each cycle during the whole active period of the step. Instructions are written according to the used language syntax. Non-stored actions have the "N" qualifier. The following are the results of a non-stored Action:




### Example

In the following program, step S1 is assigned an ST action named EdgeInit having the P1 qualifier and S2 is assigned an ST action named EdgeCount having the N qualifier. The code for these actions is programmed in their respective level 2 window.



## Coding Conditions for Transitions

You code conditions for the clearing of transitions by programming these in the level 2 window. When defining the properties of conditions, you indicate a name, a comment (optional), and the programming language (type). The available programming languages for transitions are LD and ST.

When no expression is attached to the Transition, the default condition is TRUE.

**To code conditions for transitions**

1. Select the transition for which to code a condition.
2. Right-click the transition, then from the contextual menu choose **Properties**.
3. Specify the required properties for the transition from the Properties window.
   a) To rename the transition, type the required text in the Name field.
   b) To specify the type (programming language) for the transition condition, choose the required type in the Type field.
   c) To include a comment, type the required text in the Comment field.
4. In the Level 2 window, program the required condition.

## Conditions Programmed in ST

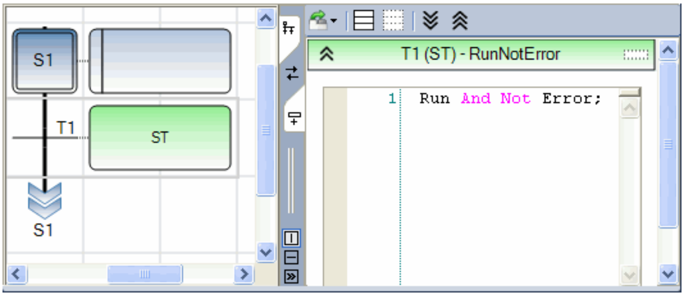
The Structured Text (ST) language can be used to describe the condition attached to a Transition. The complete expression must have Boolean type and may be terminated by a semi colon, according to the following syntax:

< boolean_expression > ;

The expression may be a TRUE or FALSE constant expression, a single input or an internal Boolean Variable, or a combination of Variables that leads to a Boolean value.

### Example

(* Program with ST programming for Transitions *)

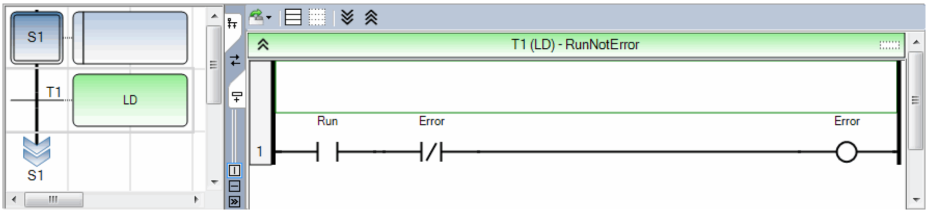


## Conditions Programmed in LD

The Ladder Diagram (LD) language can be used to describe the condition attached to a transition. The initial diagram is composed of a rung.

### Example

(* Program with LD programming for transitions *)



## Calling Functions from Transitions

Any Function (written in ST, LD, or FBD), or a "C" Function can be called to evaluate the condition attached to a Transition, according to the following syntax in ST:

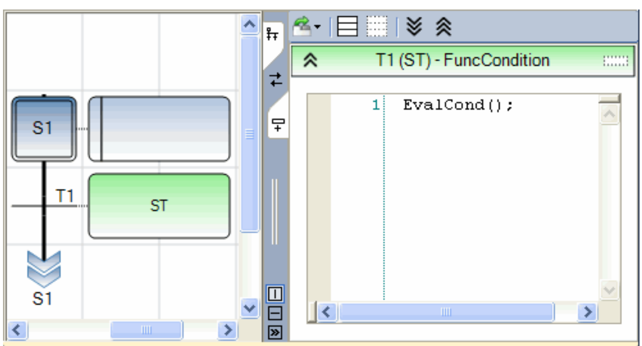
< function > ( ) ;

The value returned by the Function must be Boolean and yields the resulting condition:

| | | |
|---|---|---|
| return value = **FALSE** | -> | condition is **FALSE** |
| return value = **TRUE** | -> | condition is **TRUE** |

### Example

(* Program with function call for transitions *)



## Calling Function Blocks from Transitions

It is not recommended to call a function block in an SFC condition for the following reasons:

- A function block should be called at each cycle, typically in a cyclic program.
- An SFC condition is evaluated only when all of its preceding steps are active (not at each cycle)

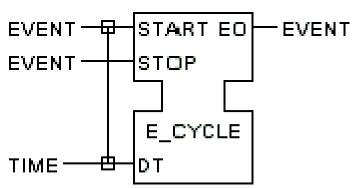
# Composite IEC 61499 Function Blocks

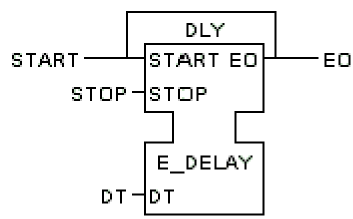
Composite IEC 61499 function blocks are defined using IEC 61499 FBD calling standard IEC 61499 function blocks, basic function blocks, and composite function blocks to perform the required operations.

A composite IEC 61499 function block is like a function block network where nodes are basic and/or composite function blocks and their parameters and where branches are data connections and event connections. **ISaGRAF** automatically implements the WITH qualifier to ensure synchronization between data inputs and event inputs.

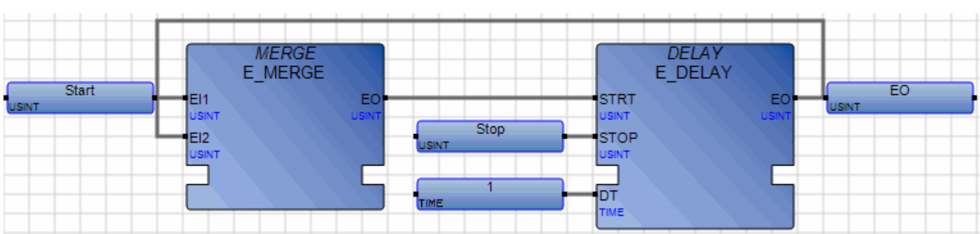
The following example shows the E_CYCLE composite function block:

**E_CYCLE** **Algorithm**




**IEC 61499 FBD Equivalent**




The following elements are available for composite IEC 61499 function blocks:

- Function Blocks
- Variables
- Comments
- Links
- Regions

## Function Blocks

In IEC 61499 programs and composite function blocks, you can include standard IEC 61499 or user-defined function blocks. You include functions blocks by inserting block elements into the language container then selecting the function block from the block selector. Following insertion, you connect inputs and outputs to variable blocks (literals or defined words) or other block inputs or outputs. Formal parameter short names are displayed inside the blocks.

**To insert a block element**

1. From the **Toolbox**, drag the block element into the language container.

   The Block Selector is displayed.

2. In the **Block Selector**, choose the required function block, then click **OK**. You can sort the block list according to the columns by setting these in ascending or descending order.

The selected function block is displayed in the language container.

# Variables

In IEC 61499 programs and composite function blocks, variable blocks can only be literals or defined words.

To connect a new symbol to an existing one (a block input or output), drag the element until its connecting line on the left (or right) overlaps an existing connecting point. When the mouse is released, the new symbol is automatically created and linked.

When entering variable blocks, you need to enter a literal or select a defined word by double-clicking the variable element. Available defined words are displayed in a drop-down list.

**To insert a variable element**

1. From the toolbox, drag the variable element to the required input or output.
2. Double-click the variable element, then do one of the following:
   - To specify a literal, type the required value in the text box.
   - To specify a defined word, select a defined word from the drop-down list.

The variable element is displayed in the language container with the specified value.

## Links

You draw connection links between block inputs and outputs. For variable elements, the links are automatically drawn when the element approaches an input or output.

Negation connection links are equivalent to placing a NOT block on a direct link.

Links are always drawn from an output to an input point (following the direction of the data flow).

**To insert a link between outputs and inputs in programs**

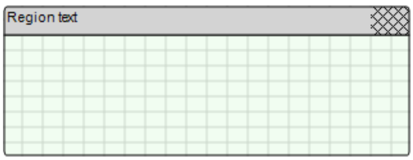
1. Click an output, then drag while holding the mouse depressed to the required input.
2. To set the link to negation, right-click the link and choose Properties from the contextual menu, then in the Properties window, set the Is Negation property to True.

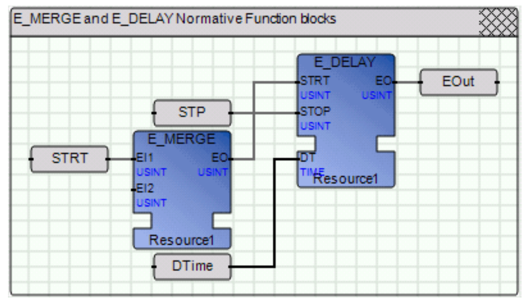
**To insert a link between outputs and inputs in composite function blocks**

- Right-click an output and from the contextual menu choose Connect To, then the required function block and input with which to connect.

## Regions

Regions delineate and group together areas of an IEC 61499 POU. A region consists of a header and a delineated zone grouping together elements.The header section enables entering free-format text. After entering text in the header, click elsewhere in the region to exit editing mode. When moving the location of a region in the language container, you can also move all the content grouped within. You can resize regions.




### To insert a region

- From the Toolbox, drag the region element into the language container.

The region element is displayed in the language container.

### To move a region

1. In the language container, left-click the top right corner of the region element and hold the mouse button.
2. Drag the region element to the required location and release the mouse button.

The region and the elements contained inside have moved location in the language container.

### See Also

Comments

# Comments

Comments are free format text inserted anywhere in the POU, for documentation purposes only. After entering text, click elsewhere in the workspace to exit editing mode.

**To insert a comment**

1. From the toolbox, drag the comment element to the required location in the language container.
2. Double-click the comment element, then type the required text within the space provided.

The comment is displayed in the language container.

# Execution Control Chart Behavior

The execution control chart behavior consists of three states: initial situation (start), code execution, and end. Each virtual machine cycle consists of determining all clearable transitions and executing as many active steps as possible. Execution ends upon reaching unclearable transitions, the end of the control chart, or a previously executed step.

Within the execution cycle, the dynamic behaviors of the SFC language are the following:

**Initial situation**

The Initial Situation is characterized by the initial steps which are, by definition, in the active state at the beginning of the operation. At least one initial step must be present in each SFC program.

**Clearing of a transition**

A transition has three properties: enabled/disabled, active/inactive, and clearable/non-clearable. A transition is enabled when all immediately preceding steps linked to its corresponding transition symbol are active, otherwise, the transition is disabled. A transition is active if its condition is True.

A transition is clearable if it is enabled and active at the same time. When a transition is clearable, the steps immediately preceding it become inactive and those immediately following it become active. When transitions follow a divergence, multiple transitions may become clearable.

**Changing of state of active steps**

The clearing of a transition simultaneously leads to the active state of the immediately following steps and to the inactive state of the immediately preceding steps. The code within a step is only executed if the step is active.

**Simultaneous clearing of transitions**

All transitions (of all SFC programs) that can be cleared (enabled and active), are simultaneously cleared.

**End**

The End is characterized by reaching the end of clearable transitions, the end of the control chart, or a previously executed step.

# IEC 61499 Keyboard Shortcuts

The following keyboard shortcuts are available for use with IEC 61499. Some shortcuts do not apply or may differ while debugging.

| | |
|---|---|
| Ctrl+A | Selects all elements (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while debugging) |
| Ctrl+V | Pastes elements saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl+X | Cuts the selected elements to the clipboard (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Ctrl+Alt+G | Enables/disables the grid in the language container |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+R | Toggles between Auto-Input and Manual-Input. Auto-Input automatically opens the Block Selector and Variable Selector (not available while debugging). |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctrl+I | Italicizes selected comment text (not available while debugging) |
| Ctrl+U | Underlines selected comment text (not available while debugging) |
| Enter | When a function block is selected, opens the Block Selector (not available while debugging).<br><br>When a comment is selected, starts editing it (not available while debugging). |
| Ctrl+Enter | When a variable is selected, opens the drop-down list of available variables (not available while debugging).<br><br>When editing a comment, confirms the text (not available while debugging). |
| Ctrl+- | Decreases the magnification |
| Ctrl+= | Increases the magnification |

| | |
|---|---|
| Ctrl+0 | 100% magnification |
| Ctrl+1 | Inserts a variable (not available while debugging) |
| Ctrl+2 | Inserts a function block (not available while debugging) |
| Ctrl+3 | Inserts a comment (not available while debugging) |
| Shift+Up Arrow | Reduces the height of the selected element (not available while debugging) |
| Shift+Down Arrow | Increases the height of the selected element (not available while debugging) |
| Shift+Left Arrow | Reduces the width of the selected element (not available while debugging) |
| Shift+Right Arrow | Increases the width of the selected element (not available while debugging) |
| Ctrl+Up Arrow | Moves the selection to the next element located higher in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Down Arrow | Moves the selection to the next element located lower in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Left Arrow | Moves the selection to the next element located to the left in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Right Arrow | Moves the selection to the next element located to the right in the diagram without keeping the previous element selected (not available while debugging) |
| Alt+Shift+Up Arrow | When a function block is selected, navigates up the different inputs and outputs (not available while debugging) |
| Alt+Shift+Down Arrow | When a function block is selected, navigates down the different inputs and outputs (not available while debugging) |
| Alt+Shift+Left Arrow | When a function block is selected, navigates left across the different inputs and outputs (not available while debugging) |
| Alt+Shift+Right Arrow | When a function block is selected, navigates right across the different inputs and outputs (not available while debugging) |
| Ctrl+Page Up | Jumps to the top of the language container |

| | |
|---|---|
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Alt+Up Arrow | Scrolls up |
| Alt+Down Arrow | Scrolls down |
| Alt+Left Arrow | Scrolls left |
| Alt+Right Arrow | Scrolls right |
| Up Arrow | Moves selected elements up the language container. While debugging, scrolls up. |
| Down Arrow | Moves selected elements down the language container. While debugging, scrolls down. |
| Left Arrow | Moves selected elements left across the language container. While debugging, scrolls left. |
| Right Arrow | Moves selected elements right across the language container. While debugging, scrolls right. |
| Delete | Removes the selected elements (not available while debugging) |

# FBD Language

The Functional Block Diagram (FBD) is a graphic language enabling programmers to build complex procedures by taking existing functions from the standard library, function section, or function block section.

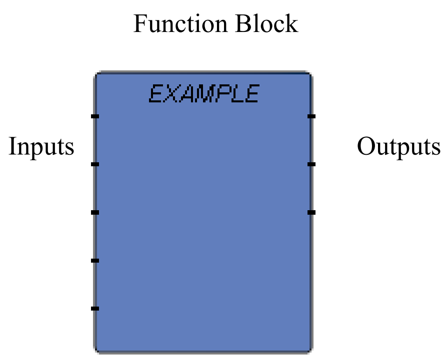
In FBD containers, you can also include LD elements such as coils, contacts, jumps, labels, and returns. However, in contrast to LD elements usage in LD containers where these elements follow strict graphical positioning regulations, LD elements within FBD container are independent of these regulations.

**See Also**

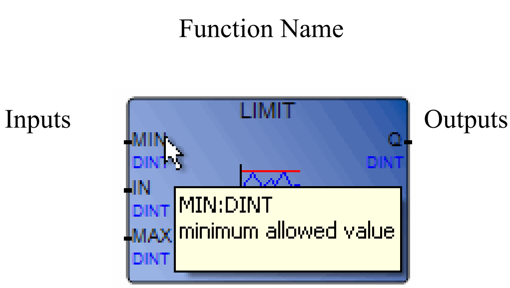
FBD Diagram Main Format
Debugging FBD Programs

# FBD Diagram Main Format

FBD diagrams describe a process between input variables and output variables. A process is described as a network of basic elements. Input and output variables are connected to blocks by connection lines. Outputs of blocks can also be connected to inputs of other blocks.




An entire process represented by an FBD program is built using the available variables, operators, functions, and function blocks. Each block has either a fixed or defined number of input and output connection points. A block is represented by a single rectangle. The inputs are connected on its left border. The outputs are connected on its right border. An elementary block performs a single function between its inputs and its outputs. The name of the function to be performed by the block is written inside its rectangular shape. Each input or output of a block is labeled and has a well defined type.



Input variables of an FBD program must be connected to input connection points of blocks. The type of each variable must be the same as the type expected for the associated input. An input for FBD diagram can be a literal, any internal or input variable, an output variable, or a block output.

Output variables of an FBD program must be connected to output connection points of blocks. The type of each variable must be the same as the type expected for the associated block output. An output for FBD diagram can be any internal or output variable, or the name of the function (for functions only). When an output is the name of the currently edited function, it represents the assignment of the return value for the function (returned to the calling program).

Input and output variables, inputs and outputs of the blocks are wired together with connection lines, or links. Single lines can be used to connect two logical points of a diagram:

- An input variable and an input of a block
- An output of a block and an input of another block
- An output of a block and an output variable

The connection is oriented, meaning that the line carries associated data from left to right. The left and right ends of the connection line must be of the same data type.

Multiple right connections, also called divergences can be used to broadcast information from their left end to each of the right ends. All ends of the connections must be of the same data type.

**See Also**

Execution Order of FBD Programs

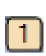
# Execution Order of FBD Programs

When editing FBD programs, you can display the execution order of elements and networks. Within a program, a network is a sequence of connected blocks. The execution order for elements and networks can be defined automatically or manually. When using the manual definition for the execution order, a region is considered a sub-network where the rules of execution order apply to the elements inside the region. The execution order is displayed for the following elements in the form of numerical tags:

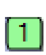
- blocks
- variables (where a value is assigned from another variable)
- coils
- contacts
- vertical bars
- returns
- jumps
- labels (manual definition only)

For manual definition of the execution order, a numerical tag is displayed with a red outline when the specified tag order for an element presents a possible malfunction.

Numerical tags are displayed using different colors depending on the type of execution order:




Beige. Automatic execution order.




Green. Manual definition for the execution order.

When using manual definition, you can perform the following task:

- reset the manual definition order to the default execution order

For the execution order of a program, a block is any object in the diagram, a network is a set of blocks linked together, and the position of a block is based on its top-left corner. The following rules apply to the execution order of the program:

- Networks are executed from top to bottom, left to right. During execution, a grouping is entirely processed before moving to the next grouping.
- All inputs must be resolved before executing a block. When the inputs of two or more blocks are resolved at the same time, the decision for the execution is based on the position of the block.
- The outputs of a block are assigned following execution

You can perform execution order operations from the menu bar, the toolbar, or keyboard shortcuts.

**To display the execution order in an FBD diagram**

- From the Tools menu, click **Execution Order** (or press **Ctrl+W**).

Numerical tags in the individual elements display the default execution order.

**To manually define the execution order in an FBD diagram**

Before manually defining the execution order, you need to display the execution order for a diagram. You specify manual definition of the execution order from the contextual menu, accessed by right-clicking in the language container.

1. To specify manual definition of the execution order, right-click in the language container, point to **Execution Order**, and then click **Manual Definition** (or press **Ctrl+Alt+M**).
2. To start redefining the execution order of individual elements, right-click in the language container, point to **Execution Order**, and then click **Start Renumbering**.

   The numerical tags displaying the execution order are available for renumbering.
3. Click the individual elements in the required order of execution.

When renumbering is complete, right-click in the language container, point to **Execution**

**Order**, and then click **Stop Renumbering**.

**To reset the manual definition to the default execution order rules**

You reset the execution order from the contextual menu, accessed by right-clicking in the language container.

1. To reset the execution order to use the default rules, right-click in the language container, point to **Execution Order**, and then click **Reset**.

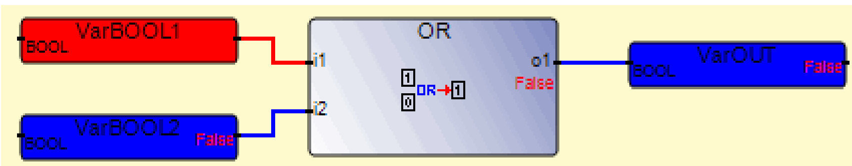
## See Also

FBD Diagram Main Format
Regions

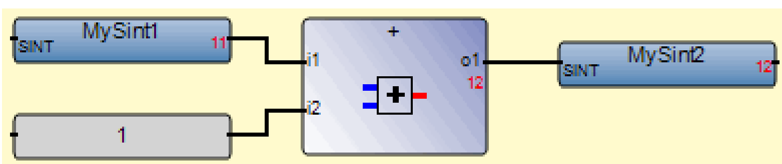
# Debugging FBD Programs

When debugging FBD programs, you can monitor the output value of elements. These values are displayed using color, numeric, or textual values according to their data type:

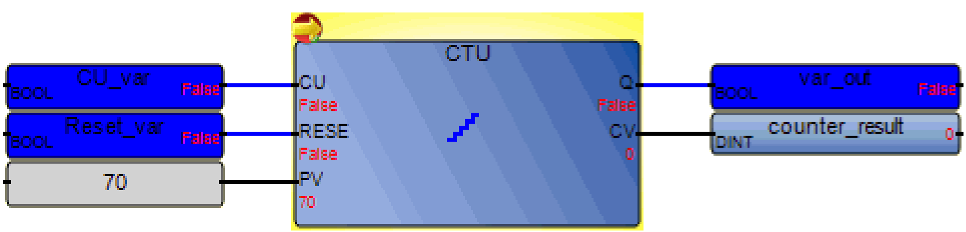
- Output values of boolean type are displayed using color. The output value color continues to the next input. When the output value is unavailable, boolean elements remain black. The colors are red when True and blue when False.




- Output values of SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, and STRING type are displayed as a numeric or textual value in the element. When the output is a structure type, the displayed value is the selected member.




When the output value for a numeric or textual value is unavailable, the *WAIT* text is displayed in the output label. Values are also displayed in the corresponding dictionary instance.



For FBD programs, you enable step-by-step execution by generating debug information for individual POUs. When debug information is generated for FBD programs in a resource, the resource automatically switches to step-by-step execution when the application encounters a breakpoint. You instantiate step-by-step execution by setting breakpoints to functions, function blocks, operators, user-defined functions, and user-defined function blocks. In the language editor, breakpoints appear as red circles located at the top left corner of blocks. When a breakpoint is encountered, a yellow arrow is displayed on the breakpoint and the block is highlighted in yellow. When debugging, the application stops when it encounters a breakpoint. At this time, the resource is in the DEBUGGING state and you can choose to perform one of the following operations:

- Step into the highlighted block (available for user-defined functions and function blocks), executes the highlighted block then steps to the subsequent block.
- Step over the highlighted block, skips the highlighted block then steps to the subsequent block.
- Switch execution to real-time mode
- Switch execution to cycle-to-cycle mode
- Execute one cycle

**Note:** You can only set breakpoints for TIC POUs; you cannot set breakpoints for C source code POUs.

**To generate debug information for an FBD POU**

Generate debug information for FBD POUs enables step-by-step debugging within the POU.

1. In the Solution Explorer, select the FBD POU for which to generate debug information.
2. In the Properties for the POU, set *Generate Debug Info* to True.

**To set a breakpoint in an FBD POU**

- Right-click the block on which to add a breakpoint, then click **Add Breakpoint**.

A breakpoint is displayed as a red circle at the top left corner of the block.

**To disable a breakpoint**

Disabling a breakpoint prevents the block execution.

- Right-click the block on which to disable a breakpoint, then click **Disable Breakpoint**.

**To enable a breakpoint**

Enabling a breakpoint allows block execution.

- Right-click the block on which to enable a breakpoint, then click **Enable Breakpoint**.

**To delete a breakpoint**

- Right-click the block having a breakpoint to remove, then click **Delete Breakpoint**.

The breakpoint is deleted from the block.

**To step into the user-defined function or function block**

- From the Debug menu, click **Step Into** (or press **F11**).

The POU executes the highlighted user-defined function or function block then steps to the next block.

**To step over the block**

- From the Debug menu, click **Step Over** (or press **F10)**.

The POU skips the highlighted block then steps to the next one.

**To switch execution to real-time mode**

- From the Target Execution toolbar, click .

The POU executes in real-time mode.

**To switch execution to cycle-to-cycle mode**

- From the Target Execution toolbar, click .

The POU executes in cycle-to-cycle mode.

**To execute one cycle**

- From the Target Execution toolbar, click .

Executes the remaining POUs until the next cycle.

# FBD Elements

When programming in FBD, you place elements in the workspace by dragging them from the Toolbox into the language container. For FBD POUs, the following elements are available:

- Blocks
- Variables
- Vertical Bars
- Labels
- Jumps
- Returns
- Rungs
- Left Power Rails
- Right Power Rails
- Coils
- Contacts
- Regions
- Comments

**See Also**

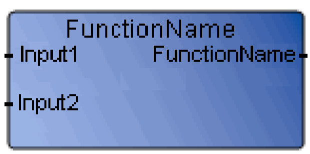
FBD Diagram Main Format
Execution Order of FBD Programs

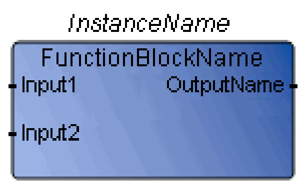
## Blocks

Block elements can be operators, functions, or function blocks. You connect block inputs and outputs to variables, contacts or coils, or other block inputs and outputs. You insert block elements in language containers.

Functions and function blocks are represented by a box displaying the name of the function, function block, or operator, and the parameter names. For function blocks, the instance name is displayed in italics.

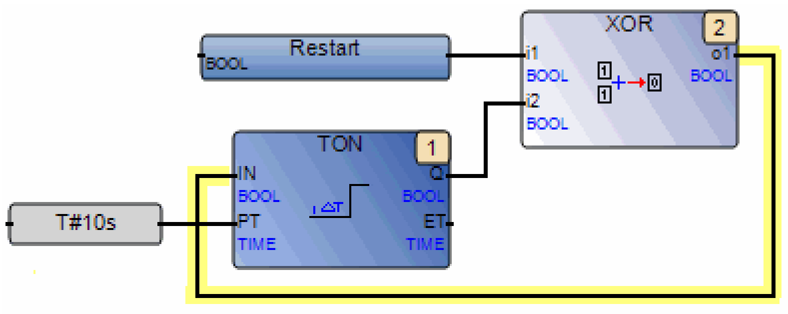
For functions, the return parameter is the only output. For function blocks, multiple return parameters can provide multiple outputs. The return parameter of a function has the same name as the function. The return parameters of a function block can have any name.




You define the parameters of POUs in the Parameters view.

For loops in blocks, you need to use local variables since these are initialized with a value. When using loops, the first execution may produce incorrect outputs due to the execution order of elements in the diagram or the initial values of temporary variables. For example, the following diagram produces a warning when compiling since the TON block is executed before the XOR operator. Whereas, moving the XOR operator to the upper left corner of the diagram eliminates the warning since the XOR operator becomes first in the execution order.




You can resize blocks elements.

**To access the parameters view**

The parameters view is available from function or function block instances located in the Solution Explorer.

1. In the Solution Explorer, right-click the required function or function block, then click **Parameters**.

   The Parameters view is displayed.

2. To define the parameters of a function or function block, in the Parameters view, enter the required information in the fields provided.

**To insert a block element**

1. From the Toolbox, drag the block element into the language container.

   The Block Selector is displayed.

2. In the Block Selector, choose the required function block, then click **OK**. You can sort the block list according to the columns by setting these in ascending or descending order.

The selected block is displayed in the language container.

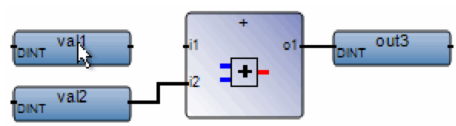
### See Also

FBD Diagram Main Format

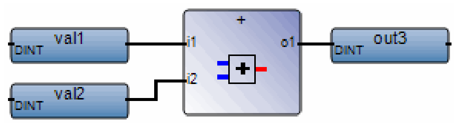
## Variables

To connect a new symbol to an existing one (another variable, a block input, or a block output) in the workspace, keep the mouse button depressed (the cursor becomes a "ghost" symbol) and drag the element until its connecting line on the left (or right) overlaps an existing connecting point. When the mouse is released, the new symbol is automatically created and linked.

Drag to place the existing element:

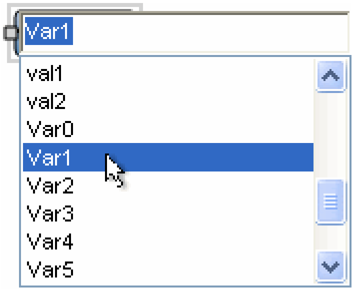



Release the mouse button. The variable is automatically connected:

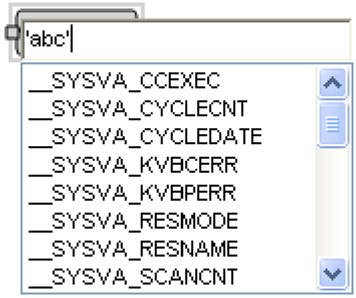



You replace existing variables in POUs by double-clicking them to access the Variable Selector or single-clicking them to select from a drop-down combo-box containing the global and local variables. Also, you can single-click a variable, then type a literal value in the text box provided. When inserting literal values that being with a letter or an underscore, enclose these in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:

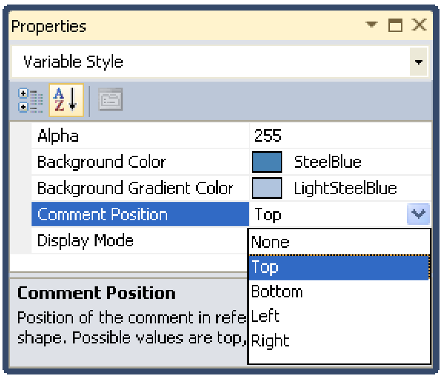


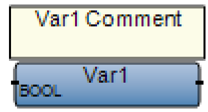

Type a literal value in the text box:




When selecting items such as local variables, global variables, system variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items. You can focus on listed items by typing letters, digits, and specific special characters: !, #, $, %, &, \, *, +, -, ./ <, :, =, >, ?, @, \, ^, _, `, |, and ~.

For input and output variables, you can choose to display comments entered in the dictionary. From the View menu, you can access the Properties window where you can define the *Comment Position* property.




You can resize variables displayed in the workspace.

### To insert a variable

1. From the Toolbox, drag the variable element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**

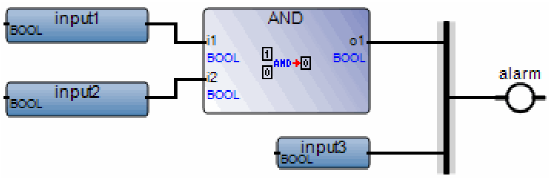
The variable is displayed in the language container.

## See Also

FBD Diagram Main Format

## Vertical Bars

Vertical bars are graphic components of FBD programs enables closing multiple parallel links. More than one horizontal links on the left side of a vertical bar are connected to one link on the right side. The Boolean state of the right end is the logical OR between all the left extremities.




**To insert a vertical bar**

- From the Toolbox, drag the vertical bar element into the language container.

The vertical bar is displayed in the language container.

## Labels

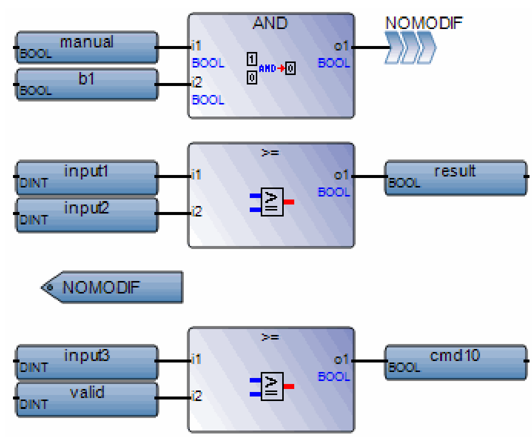
Labels can be placed anywhere in an FBD diagram. These are used as a target for jump instructions, to change the execution order of the diagram. Labels are not connected to other elements.

Place labels on the left of the diagram in order to increase diagram readability.

Labels are used to control the execution of the diagram. No other object may be connected on the right of a label symbol.

If the connection line on the left of the jump symbol has the Boolean state TRUE, the execution of the program directly jumps to after the corresponding label symbol.

### Example




**To insert a label**

**1.** From the Toolbox, drag the label element into the language container.

**2.** In the language container, click the label, then type a label name in the space provided.

The label is displayed in the language container.

**See Also**

Jumps

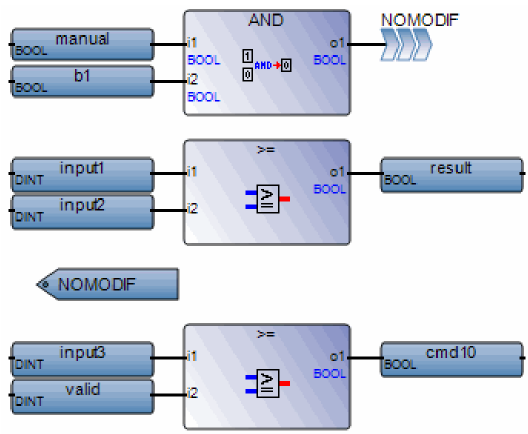
# Jumps

A Jump symbol must be linked to a Boolean point. When this Boolean (left) connection is TRUE, the execution of the diagram Jumps directly to the target Label.

Jumps are used to control the execution of the diagram. No other object may be connected on the right of a jump symbol.

If the connection line on the left of the jump symbol has the Boolean state TRUE, the execution of the program directly jumps to after the corresponding label symbol.

## Example




**To insert a jump to a label**

Before inserting jumps, define one or more labels within the program.

**1.** From the Toolbox, drag the jump element into the language container.

**2.** In the language container, click the jump element, then select the required label name from the drop-down combo-box.

The jump is displayed in the language container with the required label name.

**See Also**

Labels

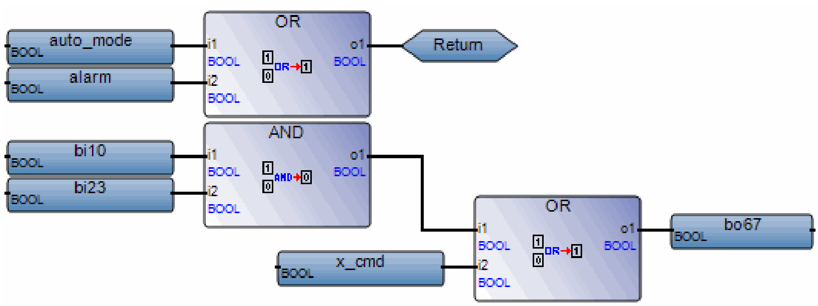
## Returns

If the connection line (to the left of the Return symbol) has the Boolean state TRUE, the Program ends - no further part of the diagram is executed.

No connection can be put on the right of a RETURN symbol.

The "<RETURN>" keyword may occur as a diagram output. It must be connected to a Boolean output connection point of a block. The RETURN statement represents a Conditional End of the program: if the output of the box connected to the statement has the Boolean value TRUE, the end (remaining part) of the diagram is not executed.

### Example




(* ST equivalence: *)

```
If auto_mode OR alarm Then
Return;
End_if;
bo67 := (bi10 AND bi23) OR x_cmd;
```

**To insert a return**

- From the **Toolbox**, drag the return element into the language container.

The return is displayed in the language container.

## Rungs

Rungs are graphic components of FBD programs and represent a group of circuit elements leading to the activation of a coil. Dragging the rung element into the workspace inserts a left power rail linked to a right power rail. Also, the rung contains a direct contact and a direct coil. Error symbols (  ) indicate that the direct contact and direct coil are undefined.

**To insert a rung**

- From the Toolbox, drag the rung element into the language container.

The rung is displayed in the language container.

## Left Power Rails

Left Power Rails are graphic components of FBD programs that represent the left boundary of a rung. Any horizontal link connected to a left power rail has the boolean state TRUE.

You can link left power rails to right power rails as well as many FBD and LD elements, including variables, blocks, jumps, returns, vertical bars, coils, and contacts.

**To insert a left power rail**

- From the Toolbox, drag the left power rail element into the language container.

The left power rail is displayed in the language container.

## Right Power Rails

Right Power Rails are graphic components of FBD programs that represent the right boundary of a rung. The right power rail is optional; ending the rung with a coil also produces the correct code.

You can link right power rails to left power rails as well as many FBD and LD elements, including variables, blocks, vertical bars, coils, and contacts.

**To insert a right power rail**

- From the Toolbox, drag the right power rail element into the language container.

The right power rail is displayed in the language container.

## Coils

Coils are graphic components of LD programs that you can use in FBD programs representing the assignment of Boolean outputs. A coil represents an action. It must be connected on the left to a Boolean symbol, such as a contact or the Boolean output of a block.

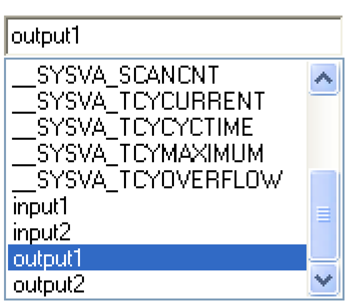
The following types of coils are available from the FBD toolbox:

- Direct Coil
- Reverse Coil
- Set Coil
- Reset Coil

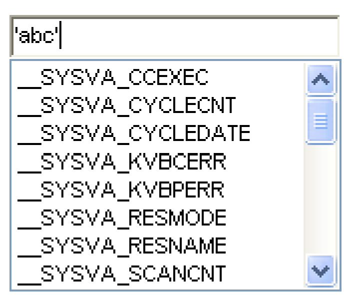
You can change the type of a coil at any time following its insertion.

When inserting coils in POUs, you assign variables using the Variable Selector. Names of assigned variables are displayed above the coil elements within POUs. You replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, you can single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:




Type a literal value in the text box:



**To insert a coil**

You can insert coils from the Toolbox.

1. From the Toolbox, drag the desired coil type into the language container and place it on the rung.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.

The coil element and its associated variable name are displayed in the language container.

**To insert a parallel coil**

1. From the Toolbox, drag a contact element into the language container while placing it parallel to the existing contact.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.
3. Drag the left and right connections to the respective connection points on the rung.

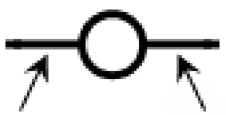
The contact and its associated variable name are displayed on the branch.

**To change the type of a coil**

- In the language container, select the coil, then select the required type in the Modify property of the Properties window.

## Direct Coil

Direct Coils enable a Boolean output of a connection line Boolean state.

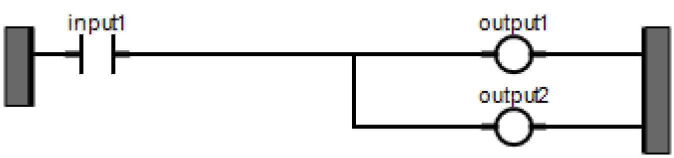


Left Connection    Right Connection

The associated variable is assigned with the Boolean state of the left connection. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

### Example




(* ST Equivalence: *)

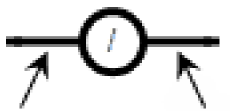
```
output1 := input1;
output2 := input1;
```

### See Also

Coils

## Reverse Coil

Reverse coils enable a Boolean output according to the Boolean negation of a connection line state.

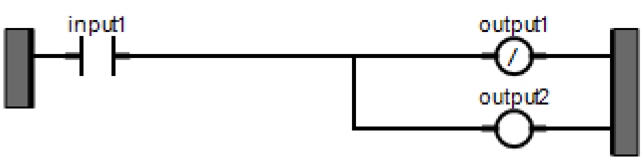


Left Connection   Right Connection

The associated variable is assigned with the Boolean negation of the state of the left connection. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

### Example



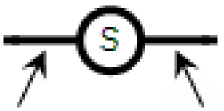

(* ST Equivalence: *)

```
output1 := NOT (input1);
output2 := input1;
```

### See Also
Coils

## Set Coil

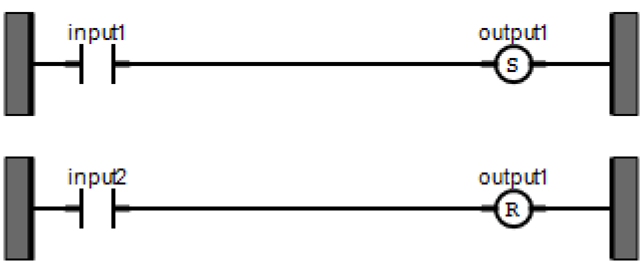
Set coils enable a Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is set to TRUE when the boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a RESET coil. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

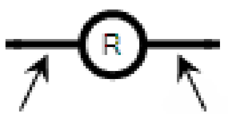
```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

**See Also**

Coils

## Reset Coil

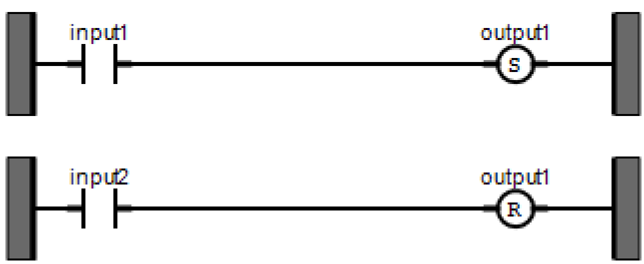
Reset coils enable Boolean output of a connection line Boolean state.




| | |
|---|---|
| Left Connection | Right Connection |

The associated variable is reset to FALSE when the Boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a SET coil. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

**See Also**

Coils

## Contacts

Contacts are graphic components of LD diagrams that you can use in FBD programs. Depending on the type of contact, it represents the value or function of an input or internal variable.

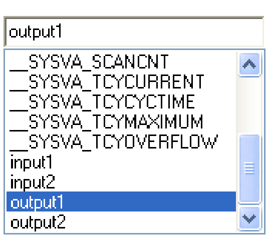
The following contact types are available from the FBD toolbox:

- Direct Contact
- Reverse Contact
- Pulse Rising Edge Contact
- Pulse Falling Edge Contact

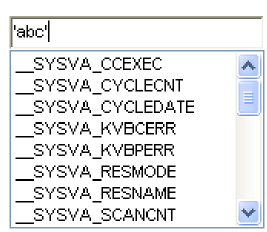
You can change the type of a contact at any time following its insertion.

When inserting contacts in POUs, you assign variables using the Variable Selector. Names of assigned variables are displayed above the contact elements within POUs. You replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, you can single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:




Type a literal value in the text box:



**To insert a contact**

You can insert contacts from the Toolbox.

1. From the Toolbox, drag the desired contact type into the language container and place it on the rung.

   The Variable Selector is displayed.

2. From the Variable Selector, select the required variable, then click **OK**.

The contact and its associated variable name are displayed in the language container.

**To insert a parallel contact**

1. From the Toolbox, drag a contact element into the language container while placing it parallel to the existing contact.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.
3. Drag the left and right connections to the respective connection points on the rung.

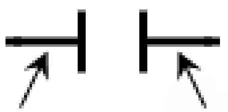
The contact and its associated variable name are displayed on the branch.

**To change the type of a contact**

- In the language container, select the contact, then select the required type in the Modifier property of the Properties window.

## Direct Contact

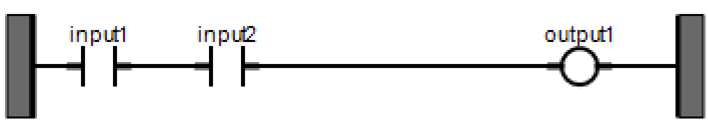
Direct contacts enable a Boolean operation between a connection line state and a Boolean variable.



Left Connection    Right Connection

The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the value of the variable associated with the contact.

### Example




(* ST Equivalence: *)

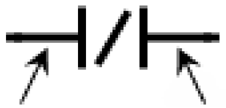
```
output1 := input1 AND input2;
```

### See Also

Contacts

## Reverse Contact

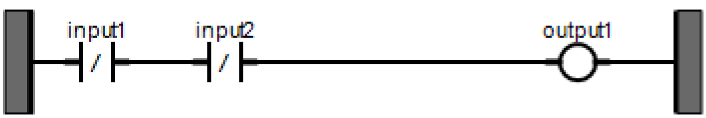
Reverse contacts enable a Boolean operation between a connection line state and the Boolean negation of a Boolean variable.



| | |
|---|---|
| Left Connection | Right Connection |

The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the Boolean negation of the value of the variable associated with the contact.

### Example




(* ST Equivalence: *)

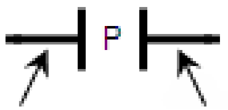
```
output1 := NOT (input1) AND NOT (input2);
```

### See Also

Contacts

## Pulse Rising Edge Contact

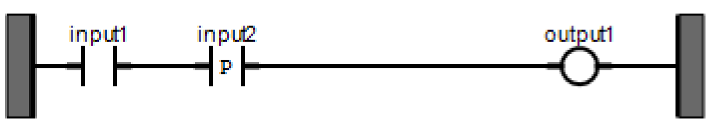
Pulse rising edge (positive) contacts enable a Boolean operation between a connection line state and the rising edge of a Boolean variable.




Left Connection　Right Connection

The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable rises from FALSE to TRUE. The state is reset to FALSE in all other cases.

### Example




(* ST Equivalence: *)

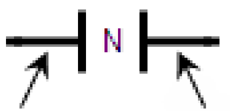
```
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)
```

### See Also

Contacts

## Pulse Falling Edge Contact

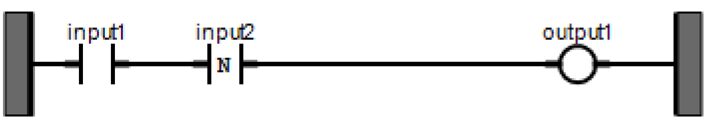
Pulse falling edge (negative) contacts enable a Boolean operation between a connection line state and the falling edge of a Boolean variable.




Left Connection    Right Connection

The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable falls from TRUE to FALSE. The state is reset to FALSE in all other cases.

### Example




(* ST Equivalence: *)

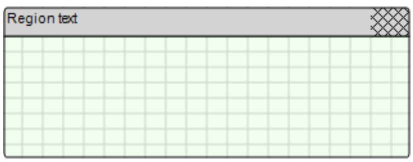
```
output1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev is the value of input2 at the previous cycle *)
```

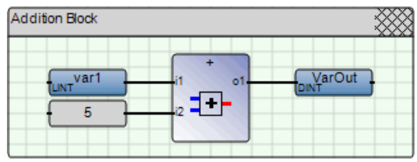
### See Also

Contacts

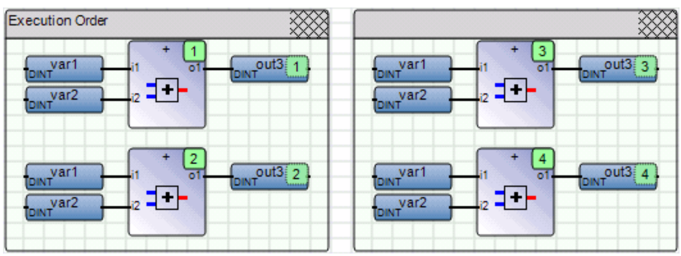
## Regions

Regions delineate and group together areas of an FBD POU. A region consists of a header and a delineated zone grouping together elements.The header section enables entering free-format text. After entering text in the header, click elsewhere in the region to exit editing mode. When moving the location of a region in the language container, you can also move all the content grouped within. You can resize regions.




The region element affects the left to right, top to bottom manual execution order. During manual execution, a grouping is entirely processed before moving to the next grouping.




### To insert a region

- From the Toolbox, drag the region element into the language container.

The region element is displayed in the language container.

### To move a region

1. In the language container, left-click the top right corner of the region element and hold the mouse button.
2. Drag the region element to the required location and release the mouse button.

The region and the elements contained inside have moved location in the language container.

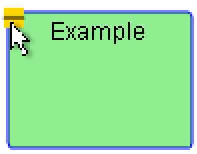
**See Also**
Comments
Execution Order of FBD Programs

## Comments

Comments are free format text inserted anywhere in the FBD POU, for documentation purposes only. After entering text, click elsewhere in the workspace to exit editing mode.

You can expand and collapse comment elements displayed in the workspace by clicking the maximize and minimize buttons. You can also resize comments.

Minimize




Maximize

**To insert a comment**

You can apply text formatting options including bold, italic, underline, strikethrough, and justify from the Description Editor toolbar. You can also define the foreground color.

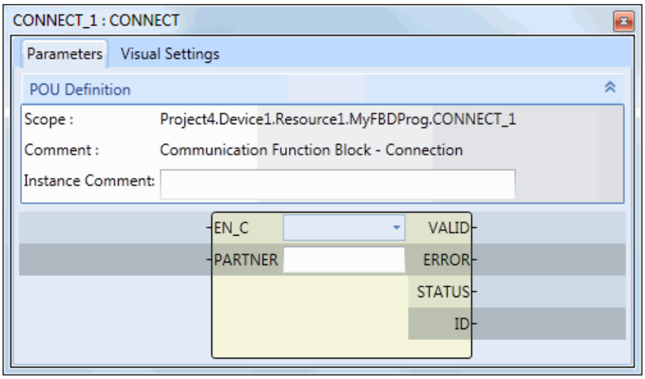
1. From the Toolbox, drag the comment element into the language container.
2. In the language container, double-click the comment, then type the required text within the space provided.

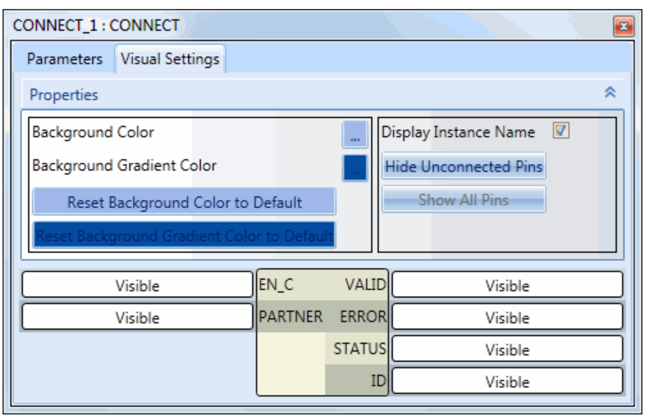
The comment is displayed in the language container.

**See Also**
Regions

# Configuring Function Block Instances

For individual function block instances in FBD, a block configurator provides an integrated environment in which to modify parameters and visual settings. You can perform the following tasks for a function block instance from a block configurator:




- Visualizing information such as the scope for the instance and comment for the block
- Specifying a comment for an instance
- Assigning initial values to unconnected inputs

- Setting background and gradient colors for an instance
- Displaying instance names
- Choosing the pins to display for the instance: hiding unconnected pins, showing all pins, or specifying individual pins. You can only hide unconnected pins.

When aliases are defined for variables, the aliases are displayed in the instance. For function block instances having hidden pins, the Display All Pins button , enables showing all pins.

**To access information and modify parameters for a function block instance**

1. In the POU, click in the upper-right corner of the block instance.

   The block configurator window for the block instance is displayed.

2. Click the Parameters tab.
3. To visualize the scope of the instance, the comment for the block, or specify a comment for the instance, expand the POU definition by clicking .
4. To specify an initial value for an input, click in the field or click ... (for inputs with multiple fields) alongside the required item, then provide the necessary values.

**To define visual settings for a function block instance**

1. In the POU, click in the upper-right corner of the block instance.

   The block configurator window for the block instance is displayed.

2. Click the Visual Settings tab.
3. To set the background color or background gradient color for the instance, click the color swatch for the respective item, then from the color picker, choose or specify the required color.

   You can also reset the background color or background gradient color for the instance.

4. To display the instance name in the block, select *Display Instance Name*.

5. Choose pins to display for the instance:
   - To mask unconnected pins, click **Hide Unconnected Pins**.
   - To display all connected and unconnected pins, click **Show All Pins**.
   - To specify individual pins to make visible, on the block representation, click the required pins to toggle from *Hidden* to *Visible*. You can only hide unconnected pins.

## FBD Keyboard Shortcuts

The following keyboard shortcuts are available for use with the FBD language. Some shortcuts do not apply or may differ while debugging.

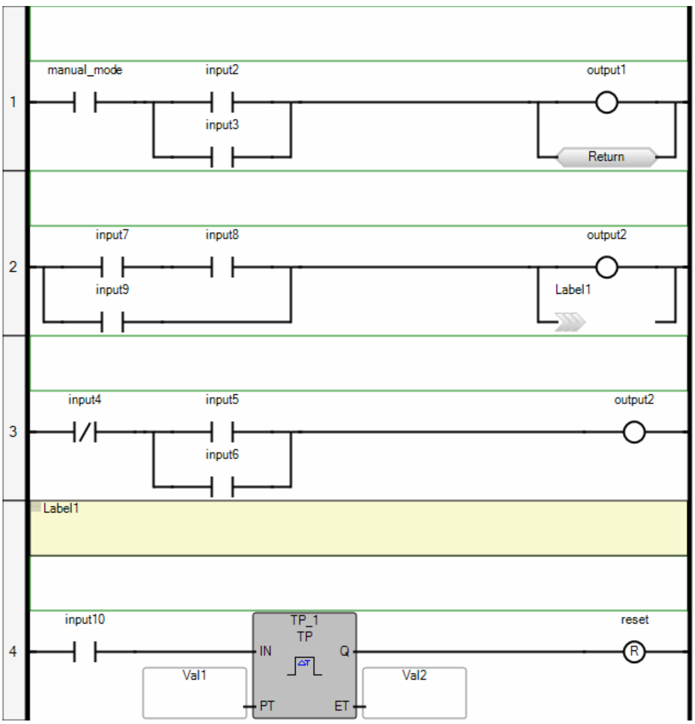
| | |
|---|---|
| Ctrl+A | Selects all elements (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while debugging) |
| Ctrl+V | Pastes elements saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl+X | Cuts the selected elements to the clipboard (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Ctrl+Alt+G | Enables/disables the grid in the language container |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+R | Toggles between Auto-Input and Manual-Input. Auto-Input automatically opens the Block Selector and Variable Selector (not available while debugging). |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctrl+I | Italicizes selected comment text (not available while debugging) |
| Ctrl+U | Underlines selected comment text (not available while debugging) |
| Enter | When a function block is selected, opens the Block Selector (not available while debugging).<br>When a variable is selected, opens the Variable Selector (not available while debugging).<br>When a comment is selected, starts editing it (not available while debugging). |
| Ctrl+Enter | When a variable is selected, opens the drop-down list of available variables (not available while debugging).<br>When editing a comment, confirms the text (not available while debugging). |
| Ctrl+- | Decreases the magnification |

| | |
|---|---|
| Ctrl+= | Increases the magnification |
| Ctrl+0 | 100% magnification |
| Ctrl+1 | Inserts a variable (not available while debugging) |
| Ctrl+2 | Inserts a function block (not available while debugging) |
| Ctrl+3 | Inserts a comment (not available while debugging) |
| Shift+Up Arrow | Reduces the height of the selected element (not available while debugging) |
| Shift+Down Arrow | Increases the height of the selected element (not available while debugging) |
| Shift+Left Arrow | Reduces the width of the selected element (not available while debugging) |
| Shift+Right Arrow | Increases the width of the selected element (not available while debugging) |
| Ctrl+Up Arrow | Moves the selection to the next element located higher in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Down Arrow | Moves the selection to the next element located lower in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Left Arrow | Moves the selection to the next element located to the left in the diagram without keeping the previous element selected (not available while debugging) |
| Ctrl+Right Arrow | Moves the selection to the next element located to the right in the diagram without keeping the previous element selected (not available while debugging) |
| Alt+Shift+Up Arrow | When a function block is selected, navigates up the different inputs and outputs (not available while debugging) |
| Alt+Shift+Down Arrow | When a function block is selected, navigates down the different inputs and outputs (not available while debugging) |
| Alt+Shift+Left Arrow | When a function block is selected, navigates left across the different inputs and outputs (not available while debugging) |
| Alt+Shift+Right Arrow | When a function block is selected, navigates right across the different inputs and outputs (not available while debugging) |

| | |
|---|---|
| Ctrl+Page Up | Jumps to the top of the language container |
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Alt+Up Arrow | Scrolls up |
| Alt+Down Arrow | Scrolls down |
| Alt+Left Arrow | Scrolls left |
| Alt+Right Arrow | Scrolls right |
| Up Arrow | Moves selected elements up the language container. While debugging, scrolls up. |
| Down Arrow | Moves selected elements down the language container. While debugging, scrolls down. |
| Left Arrow | Moves selected elements left across the language container. While debugging, scrolls left. |
| Right Arrow | Moves selected elements right across the language container. While debugging, scrolls right. |
| Delete | Removes the selected elements (not available while debugging) |
| Ctrl+D | Only available in debug mode for the date data type. When the Write Logical Value dialog box is open, enters the current date. |
| Ctrl+W | Displays the execution order of elements within the diagram (not available while debugging) |
| Ctrl+Alt+M | While displaying the execution order of elements, enables manually defining the execution order of individual elements and networks (not available while debugging) |

# LD Language

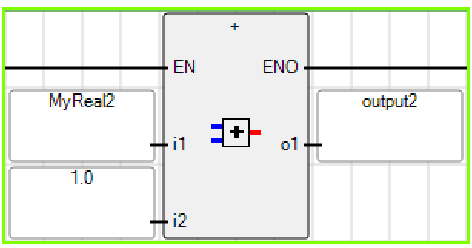
Ladder Diagram (LD) is a graphic representation of Boolean equations, combining contacts (input arguments) with coils (output results). The LD language enables the description of tests and modifications of Boolean data by placing graphic symbols into the program chart. LD graphic symbols are organized within the chart as an electric contact diagram. Thus, the term "ladder" coming from the concept of rungs connected to vertical power rails at both ends where each rung represents an individual circuit.




You can adjust editor and view settings for individual or all Ladder Diagrams. When working in a Ladder Diagram, you set the properties for the diagram from the Container properties in the Properties window. You set the properties for all Ladder Diagrams using the options available from the Tools menu. Some of the available properties include the following:

- background and gradient colors for operators, functions, and function blocks

- displaying the grid as well as the height and width of grid cells, in pixels
- the height and width of elements, in grid cells. Basic elements are blocks without inputs or outputs, coils, and contacts. For blocks, each input and output adds a basic element dimension. For example, note the contact using the default settings of one grid cell high by four grid cells wide. The following block uses a basic element width for the inputs, another for the block, and another for the outputs. The block uses a basic element height for the EN/ENO level, another for the first input and the output, and another for the second input.


- the font type, size, style, and color applied to the text displayed in elements
- various options such as displaying comments and labels, aligning coils, and setting the colors for variables, labels, comments, power rails, and rung headers

**See Also**

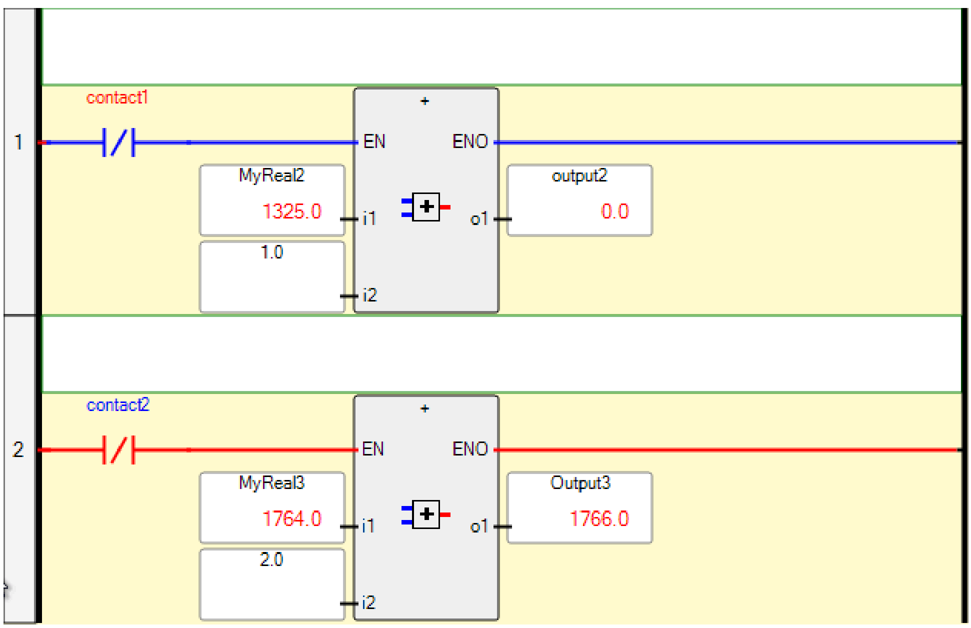
Debugging LD Programs

## Debugging LD Programs

When debugging LD programs, you can monitor the output values of elements. These values are displayed using color, numeric, or textual values according to their data type:

- Output values of boolean type are displayed using color. The output value color continues to the next input. When the output value is unavailable, boolean elements remain black. The default colors are red when True and blue when False. You can customize the colors used for boolean items.

- Output values of SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, and STRING type are displayed as a numeric or textual value in the element. When the output is a structure type, the displayed value is the selected member.



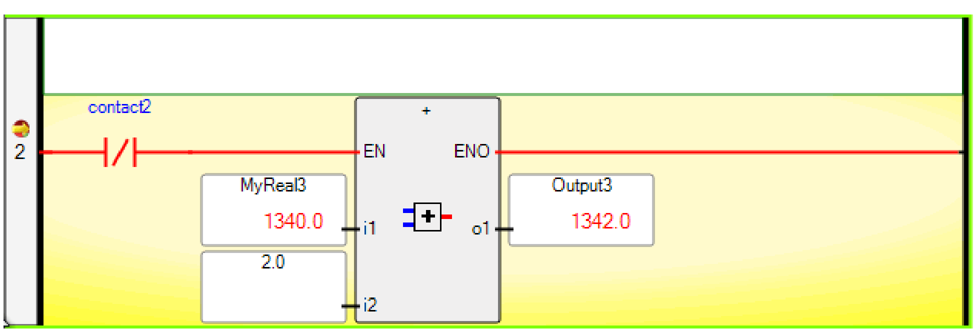

When the output value for a numeric or textual value is unavailable, the *WAIT* text is displayed in the output label. Transitional elements such as Pulse rising edge (positive) contacts, having an unstable state, remain black. Values are also displayed in the corresponding dictionary instance.

For LD programs, you enable step-by-step execution by generating debug information for individual POUs. When debug information is generated for LD programs in a resource, the resource automatically switches to step-by-step execution when the application encounters a breakpoint. You instantiate step-by-step execution by setting breakpoints to rungs. When debugging, the application stops when it encounters a breakpoint. At this time, the resource is in the DEBUGGING state and you can choose to perform one of the following operations:

- Step into the highlighted rung (available for user-defined functions and function blocks), executes the highlighted rung then steps to the subsequent rung.When the highlighted rung includes a call to a function, stepping continues in the called function then returns to the subsequent rung in the POU.
- Step over the highlighted rung, skips the highlighted rung then steps to the subsequent rung.
- Switch execution to real-time mode
- Switch execution to cycle-to-cycle mode
- Execute one cycle

**Note:** You can only set breakpoints for TIC POUs; you cannot set breakpoints for C source code POUs.

In the language editor, breakpoints appear as red circles to the left of the rung. When a breakpoint is encountered, a yellow arrow is displayed on the breakpoint and the next rung is highlighted in yellow.



When stepping passes beyond the last rung of a POU, the arrow points downward.

**To generate debug information for an LD POU**

Generate debug information for LD POUs enables step-by-step debugging within the POU.

1. In the Solution Explorer, click the LD POU for which to generate debug information.
2. In the Properties for the POU, set *Generate Debug Info* to True.

**To set breakpoints in an LD POU**

1. In the Properties for the POU, set *Generate Debug Info* to True.
2. Select the rung or rungs requiring breakpoints, right-click the rung area, and then click **Add Breakpoint**.

Breakpoints are displayed as red circles to the left of rungs.

**To remove breakpoints**

- Select the rung or rungs requiring the removal of breakpoints, right-click the rung area, and then click **Remove Breakpoint**.

The breakpoints are removed from the selected rungs.

**To step into the highlighted rung**

- From the Debug menu, click **Step Into** (or press **F11**).

The POU executes the highlighted rung then steps into the next one and stepping continues in any called function before returning to the next rung of the POU.

**To step over the highlighted rung**

- From the Debug menu, click **Step Over** (or press **F10)**.

The POU executes the current rung then steps to the next one.

**To switch execution to real-time mode**

- From the Target Execution toolbar, click .

The POU executes in real-time mode.

**To switch execution to cycle-to-cycle mode**

- From the Target Execution toolbar, click .

The POU executes in cycle-to-cycle mode.

**To execute one cycle**

- From the Target Execution toolbar, click .

Executes the remaining POUs until the next cycle.

# LD Elements

When editing an LD POU, you can place elements in a language container by dragging them from the LD Toolbox. An element is inserted at the current position in the diagram. When inserting subsequent elements, these are placed to the right of the selected element on the rung, then onto the next rung. For LD POUs, the following elements are available:

- Rungs
- Blocks
- Coils
- Contacts
- Jumps
- Returns
- Branches

## Rungs

Rungs are graphic components of LD programs and represent a group of circuit elements leading to the activation of a coil. Rungs have labels to identify them within the diagram. Labels along with jumps enable controlling the execution of a diagram. The label and jump must have the same name. When the connection on the left of the jump element has the TRUE Boolean state, the diagram execution proceeds at the label element. Comments are free format text inserted above the rung, for documentation purposes only.

**To insert a rung**

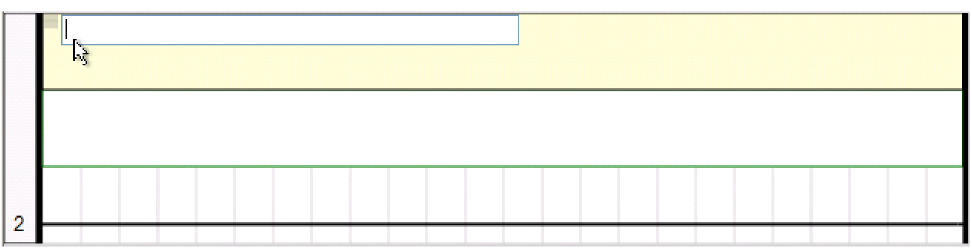
You can insert rungs from the Toolbox or using keyboard shortcuts.

- From the Toolbox, drag the rung element into the language container.

The rung is displayed in the language container.

**To define the label for a rung**

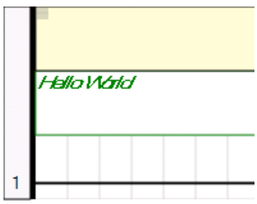
1. Right-click a rung, then click **Add Label**.
2. In the upper left-hand corner, click in the text area beside the grey square and type the required label text.



**To define the comment for a rung**

You place comments in the space above the rung. After entering text, click elsewhere in the workspace to 'validate' the comment. Text formatting options including bold, italic, underline, strikethrough, and justify, are available from the Format menu. Using the Format menu, you can also define the foreground color.

- In the language container, click the rectangular space above the rung, then type the required text.



## Blocks

In a language container, you connect blocks to Boolean lines. Blocks can be operators, functions, or function blocks. Boolean inputs and outputs are not always contained within blocks. Boolean inputs connecting blocks to rungs are always executed each cycle. Boolean outputs connecting blocks to rungs control the remaining rung power flow. When inserting blocks in a diagram, the EN and ENO parameters are added to some block interfaces. You can also force the inclusion of the EN and ENO parameters for blocks having either one Boolean input, one Boolean output, or no Boolean input and output. You activate the *Enable EN/ENO* and *Display Instance Names* options from the Ladder Diagram options.

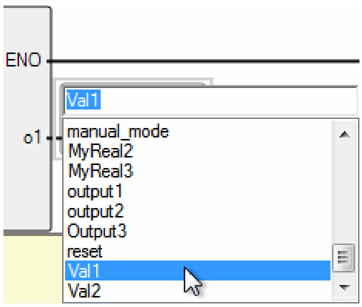
For functions and function blocks, you set the value of return parameters using coils. The return parameter of a function has the same name as the function. The return parameters of a function block can have any name.

When working with different resources, you can define parameters of POUs for multiple resources by navigating the tabs for individual resources displayed in the Parameters view.

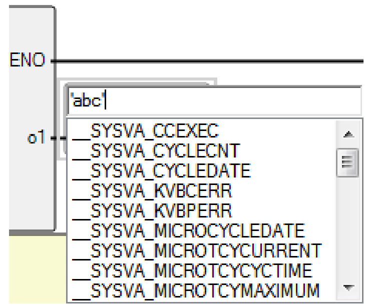
You insert blocks from the LD Toolbox. You can set the type of a block using the Block Selector at any time following insertion. When you set the type of block, variables are automatically displayed and are connected to the inputs and outputs of the block.

You replace input and output variables by double-clicking them to access the Variable Selector or single-clicking them to select from a drop-down combo-box containing the global and local variables. Also, you can single-click a variable, then type a literal value in the text box provided. When inserting literal values that being with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:



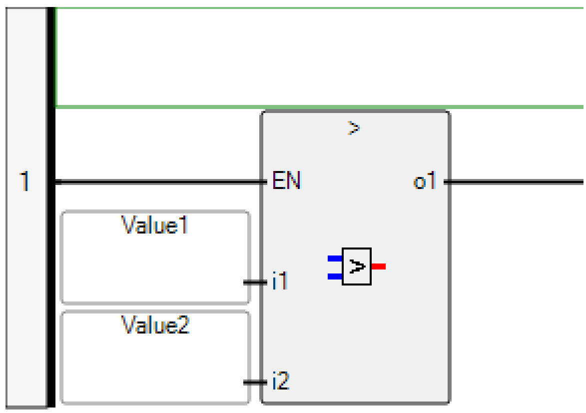

Type a literal value in the text box:




When selecting items such as local variables, global variables, system variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items. You can focus on listed items by typing letters, digits, and specific special characters: !, #, $, %, &, \, *, +, -, ,/ <, :, =, >, ?, @, \, ^, _, `, |, and ~.

### EN Input

For operators, functions, and function blocks where the first input is not a Boolean data type, another input called EN is automatically inserted at the first position since the first input is always connected to the rung. The block is executed only when the EN input is TRUE. The following example shows a comparison operator and its equivalent code expressed in ST.

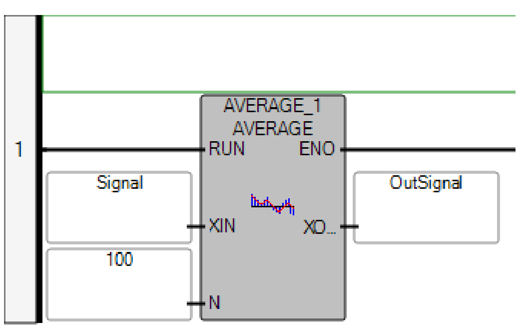



```
IF rung_state THEN
q := (value1 > value 2);
ELSE
q := FALSE;
END_IF;
```

(* continue rung with o1 state *)

### ENO Output

For operators, functions, and function blocks where the first output is not a Boolean data type, another output called ENO is automatically inserted at the first position since the first output is always connected to the rung. The ENO output always has the same state as the first input of the block. The following example shows the AVERAGE function block and its equivalent code expressed in ST.

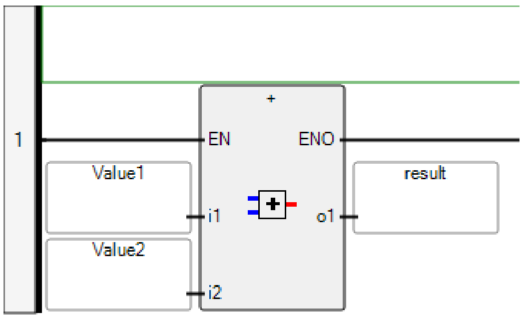



```
AVERAGE(rung_state, Signal,
100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
```

(* continue rung with eno state *)

### EN and ENO Parameters

In some cases, both **EN** and **ENO** parameters are required. The following example shows an arithmetic operator and its equivalent code expressed in ST.




```
IF rung_state THEN
result := (value1 + value2);
END_IF;
eno := rung_state;
```

(* continue rung with eno state *)

**To access the Parameters view**

The Parameters view is available from function or function block instances located in the Solution Explorer.

1. In the Solution Explorer, right-click the required function or function block, and then click **Parameters**.

   The Parameters view is displayed.

2. To define the parameters of a function or function block, enter the required information in the Parameters view.

**To insert a block**

You can insert blocks from the Toolbox or using keyboard shortcuts.

1. From the Toolbox, drag the block element into the language container and place it on the rung.

   The Block Selector is displayed.

2. In the Block Selector, locate the required block. You can sort the block list according to the columns by setting these in ascending or descending order.

- To force the inclusion of the EN/ENO parameters, select *Enable EN/ENO*.

**3.** Click **OK**.

The selected block is displayed on the rung.

**To insert a parallel block**

**1.** From the Toolbox, drag the branch element onto the existing block in the language container.

**2.** To place a block element on the branch, do the following:

   **a)** From the Toolbox, drag the block element into the language container, placing it on the branch.

   The Block Selector is displayed.

   **b)** In the Block Selector, locate the required block. You can sort the block list according to the columns by setting these in ascending or descending order.

   - To force the inclusion of the EN/ENO parameters, select *Enable EN/ENO*.

   **c)** Click **OK**.

The selected block is displayed on the branch.

## Coils

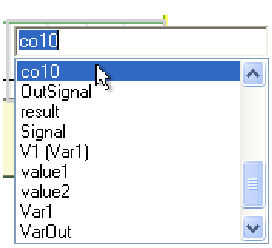
Coils are graphic components of LD programs and represent the assignment of Boolean outputs. In an LD program, a coil represents an action. It must be connected on the left to a Boolean symbol, such as a contact or the Boolean output of a block.

The following types of coils are available from the LD toolbox:

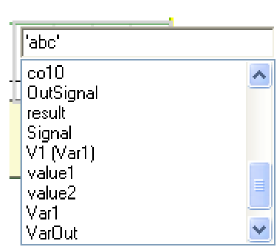
- Direct Coil
- Reverse Coil
- Pulse Rising Edge Coil
- Pulse Falling Edge Coil
- Set Coil
- Reset Coil

You can change the type of a coil at any time following its insertion. When inserting coils in POUs, you assign variables using the Variable Selector. Names of assigned variables are displayed above the coil elements within POUs. You replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, you can single-click existing variables, then type literal values in the provided text boxes. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:




Type a literal value in the text box:



When selecting items such as local variables, global variables, system variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items. You can focus on listed items by typing letters, digits, and specific special characters: !, #, $, %, &, \, *, +, -, ,/ <, :, =, >, ?, @, \, ^, _, `, |, and ~.

You can align the coils of all rungs making up diagrams to improve readability.

**To insert a coil**

You can insert coils from the Toolbox or using keyboard shortcuts.

1. From the Toolbox, drag the desired coil type into the language container and place it on the rung.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.

The coil element and its associated variable name are displayed on the rung.

**To insert a parallel coil**

1. From the Toolbox, drag the branch element into the language container, placing it on the required element.

2. From the Toolbox, drag a coil element into the language container, placing it on the branch element.

   The Variable Selector is displayed.

3. In the Variable Selector, select the required variable, then click **OK**.

The coil element and its associated variable name are displayed on the branch.

**To change the type of a coil**

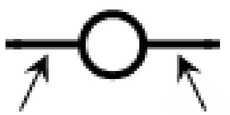
- In the language container, select the coil, then press the space bar.

**To align all coils in a diagram**

1. Right-click in the language container, and then click **Properties**.
2. In the Properties window, set the *Coil Alignment* property to True.

## Direct Coil

Direct Coils enable a Boolean output of a connection line Boolean state.

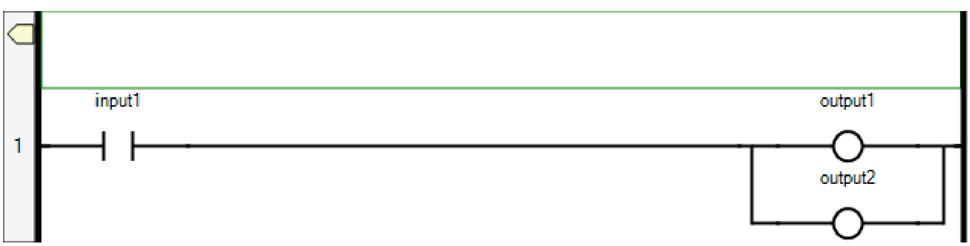


Left Connection    Right Connection

The associated variable is assigned with the Boolean state of the left connection. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

### Example




(* ST Equivalence: *)

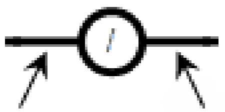
```
output1 := input1;
output2 := input1;
```

### See Also

Coils

## Reverse Coil

Reverse coils enable a Boolean output according to the Boolean negation of a connection line state.

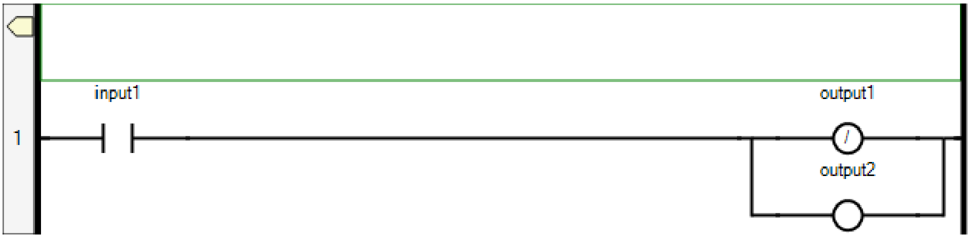


Left Connection    Right Connection

The associated variable is assigned with the Boolean negation of the state of the left connection. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

The associated name can be the name of the program (for functions only). This corresponds to the assignment of the return value of the function.

### Example




(* ST Equivalence: *)

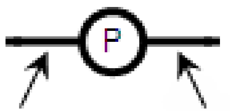
```
output1 := NOT (input1);
output2 := input1;
```

### See Also

Coils

## Pulse Rising Edge Coil

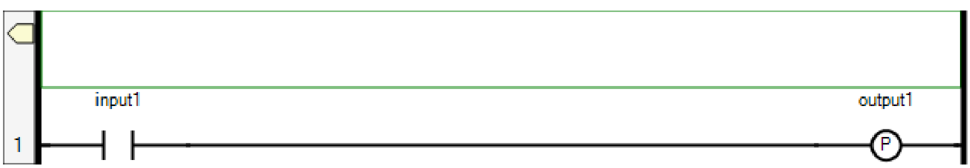
Pulse rising edge coils or "Positive" coils enable Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is set to TRUE when the Boolean state of the left connection rises from FALSE to TRUE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

```
IF (input1 and NOT(input1prev)) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
```

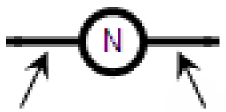
(* input1prev is the value of input1 at the previous cycle *)

### See Also

Coils

## Pulse Falling Edge Coil

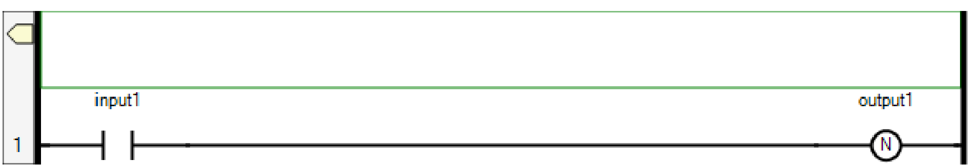
Pulse falling edge coils or "Negative" coils enable Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is set to TRUE when the Boolean state of the left connection falls from TRUE to FALSE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

```
IF (NOT(input1) and input1prev) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
```

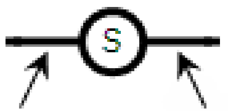
(* input1prev is the value of input1 at the previous cycle *)

### See Also

Coils

## Set Coil

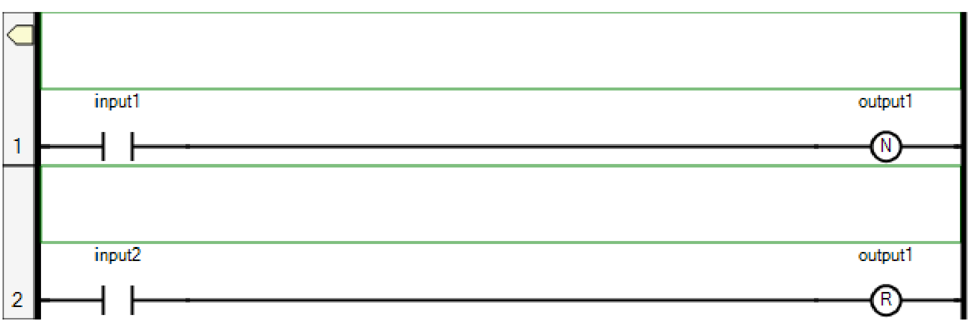
Set coils enable a Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is set to TRUE when the boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a RESET coil. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

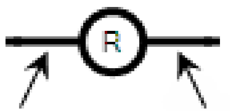
```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

**See Also**

Coils

## Reset Coil

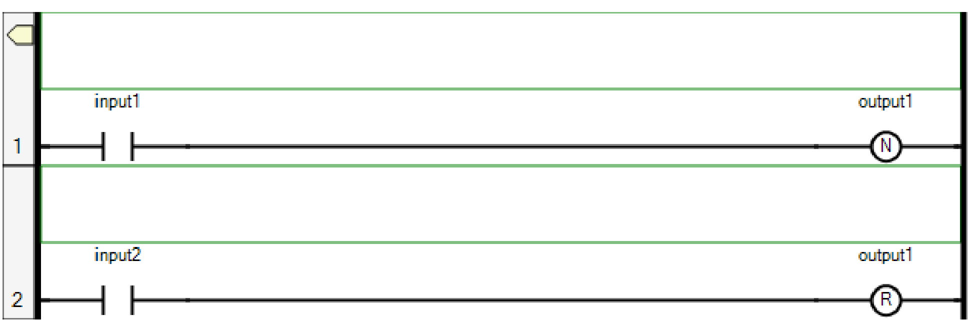
Reset coils enable Boolean output of a connection line Boolean state.




Left Connection    Right Connection

The associated variable is reset to FALSE when the Boolean state of the left connection becomes TRUE. The output variable keeps this value until an inverse order is made by a SET coil. The state of the left connection is propagated into the right connection. The right connection can be connected to the right vertical power rail.

### Example




(* ST Equivalence: *)

```
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

**See Also**

Coils

## Contacts

Contacts are graphic components of LD diagrams. Depending on the type of contact, it represents the value or function of an input or internal variable.

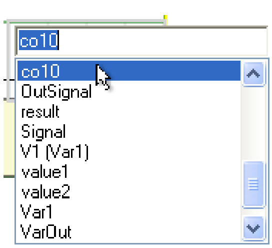
The following contact types are available from the LD toolbox:

- Direct Contact
- Reverse Contact
- Pulse Rising Edge Contact
- Pulse Falling Edge Contact

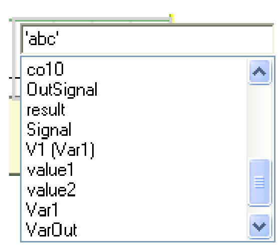
You can change the type of a contact at any time following its insertion.

When inserting contacts in POUs, you assign variables using the Variable Selector. Names of assigned variables are displayed above the contact elements within POUs. You replace existing variables by double-clicking the variable names to access the Variable Selector or by single-clicking variable names to select from drop-down combo-boxes containing the global and local variables. Also, you can single-click existing variables, then type literal values in the text boxes provided. When inserting literal values beginning with a letter or an underscore, enclose the variable name in single quotes as follows: 'abc'.

Select a variable from the drop-down combo-box:




Type a literal value in the text box:



When selecting items such as local variables, global variables, system variables, and defined words from the drop-down combo-box, typing characters in the text box focuses on the possible items. You can focus on listed items by typing letters, digits, and specific special characters: !, #, $, %, &, \, *, +, -, ,/ <, :, =, >, ?, @, \, ^, _, `, |, and ~.

**To insert a contact**

You can insert contacts from the Toolbox or using keyboard shortcuts.

1. From the Toolbox, drag the desired contact type into the language container and place it on the rung.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.

The contact and its associated variable name are displayed on the rung.

**To insert a parallel contact**

1. From the Toolbox, drag the branch element into the language container, placing it on the existing contact.

2. From the Toolbox, drag a contact element into the language container, placing it on the branch.

   The Variable Selector is displayed.

3. In the Variable Selector, select the required variable, then click **OK**.

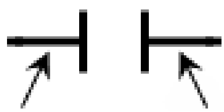
The contact and its associated variable name are displayed on the branch.

**To change the type of a contact**

- In the language container, select the contact, then press the space bar.

## Direct Contact

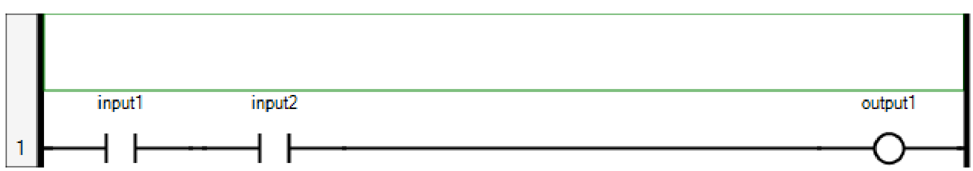
Direct contacts enable a Boolean operation between a connection line state and a Boolean variable.



Left Connection    Right Connection

The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the value of the variable associated with the contact.

### Example




(* ST Equivalence: *)

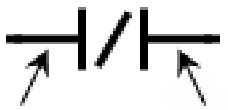
```
output1 := input1 AND input2;
```

### See Also

Contacts

## Reverse Contact

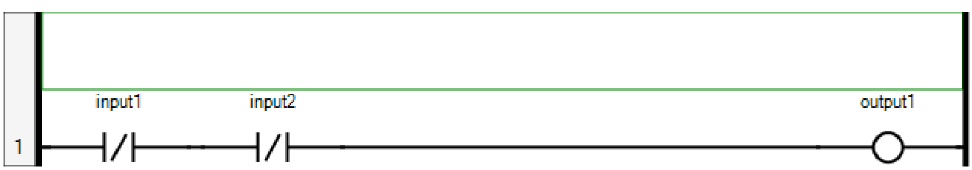
Reverse contacts enable a Boolean operation between a connection line state and the Boolean negation of a Boolean variable.



Left Connection    Right Connection

The state of the connection line on the right of the contact is the logical AND between the state of the left connection line and the Boolean negation of the value of the variable associated with the contact.

### Example




(* ST Equivalence: *)

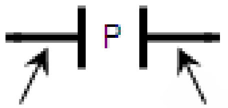
```
output1 := NOT (input1) AND NOT (input2);
```

### See Also

Contacts

## Pulse Rising Edge Contact

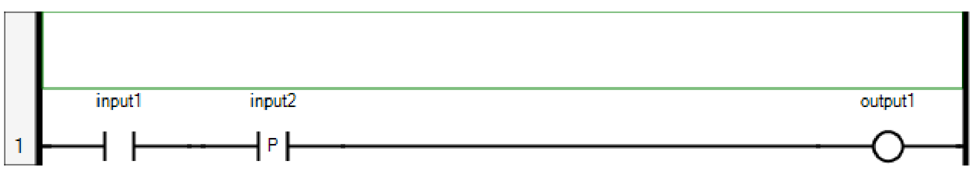
Pulse rising edge (positive) contacts enable a Boolean operation between a connection line state and the rising edge of a Boolean variable.




| | |
|---|---|
| Left Connection | Right Connection |

The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable rises from FALSE to TRUE. The state is reset to FALSE in all other cases.

### Example




(* ST Equivalence: *)

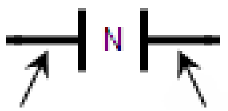
```
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)
```

### See Also

Contacts

## Pulse Falling Edge Contact

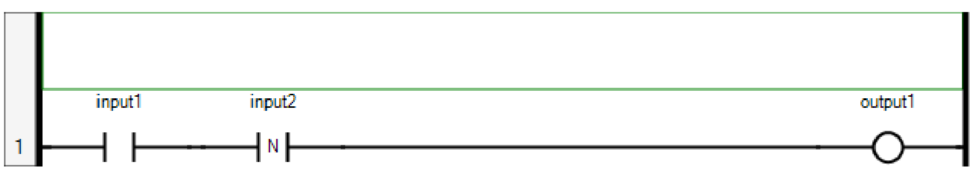
Pulse falling edge (negative) contacts enable a Boolean operation between a connection line state and the falling edge of a Boolean variable.




Left Connection
Right Connection

The state of the connection line on the right of the contact is set to TRUE when the state of the connection line on the left is TRUE, and the state of the associated variable falls from TRUE to FALSE. The state is reset to FALSE in all other cases.

### Example




(* ST Equivalence: *)

```
output1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev is the value of input2 at the previous cycle *)
```

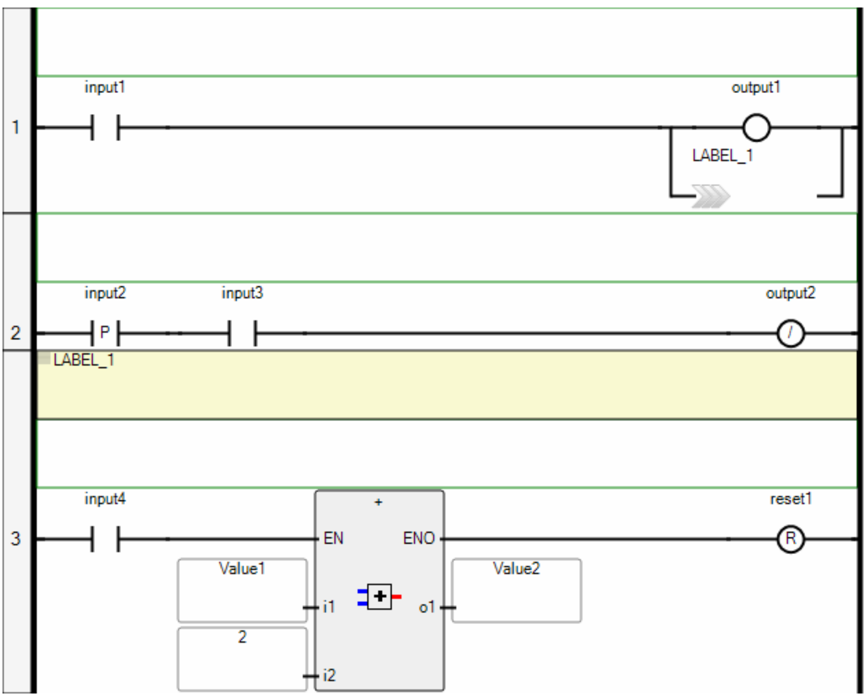
### See Also

Contacts

## Jumps

Conditional and unconditional jump elements enable controlling the execution of diagrams. You cannot place connections to the right of a jump element. When the connection on the left of the jump element has the TRUE Boolean state, the diagram execution proceeds at the label. The label and jump must have the same name.

### Example




**To insert a jump**

Before inserting jumps, define one or more labels within the program. You can insert jumps from the Toolbox or using keyboard shortcuts.

1. From the Toolbox, drag the jump element into the language container and place it on the rung.
2. In the language container, click the jump element, then select the required label name from the drop-down combo-box.

The jump is displayed on the rung with the required label name.

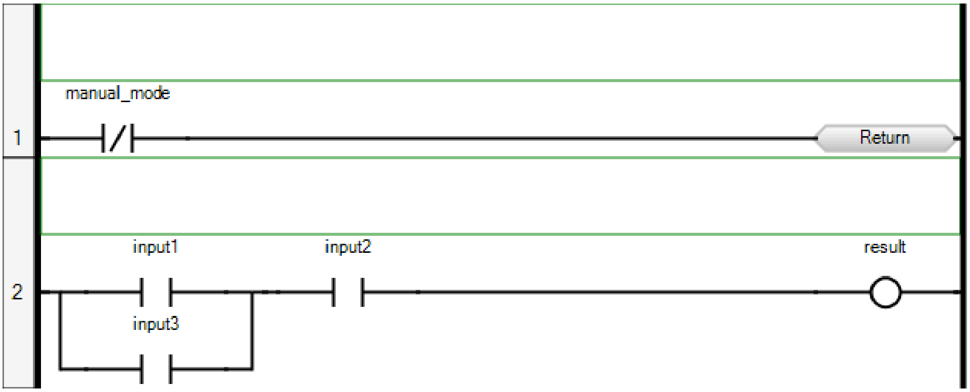
## Returns

You can use RETURN elements as outputs representing a conditional end of a diagram. You cannot place connections to the right of a RETURN element.

When the left connection line has the TRUE Boolean state, the diagram ends without executing the equations located on the next lines of the diagram.

When the LD diagram is a function, its name is associated with an output coil to set the return value (returned to the calling diagram).

### Example




(* ST Equivalence: *)

```
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;
```

**To insert a return**

You can insert returns from the Toolbox or using keyboard shortcuts.

- From the Toolbox, drag the return element into the language container, placing it on the rung.

The return element is displayed on the rung.

**See Also**

Jumps

## Branches

Branches create alternative routing for connections. You can add parallel branches to elements on a rung.

**To insert a branch**

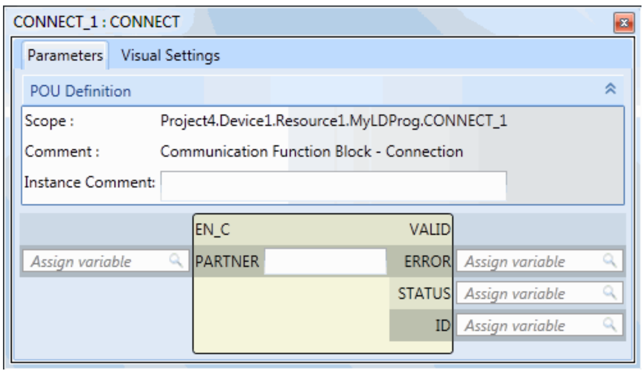
You can insert branches from the Toolbox or using keyboard shortcuts.

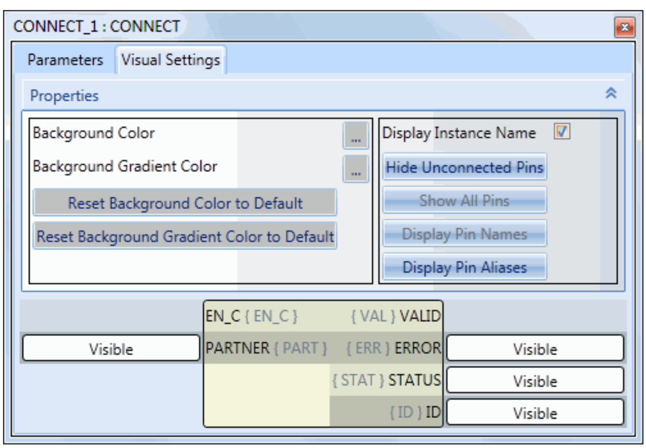
- From the Toolbox, drag the branch element into the language container and place in on the rung.

A parallel branch is displayed.

# Configuring Function Block Instances

For individual function block instances in LD, a block configurator provides an integrated environment in which to modify parameters and visual settings. You can perform the following tasks for a function block instance from a block configurator:




- Visualizing information such as the scope for the instance and comment for the block
- Specifying a comment for an instance

- Assigning variables to inputs and outputs as well as defining structure elements within arguments
- Setting background and gradient colors for an instance
- Displaying instance names
- Choosing the pins to display for the instance: hiding unconnected pins, showing all pins, or specifying individual pins
- Choosing the pin information to display for the instance: pin names or pin aliases

For function block instances having hidden pins, the Display All Pins button, enables showing all pins.

**To access information and modify parameters for a function block instance**

1. In the POU, click in the upper-right corner of the block instance.

   The block configurator window for the block instance is displayed.

2. Click the Parameters tab.
3. To visualize the scope of the instance, the comment for the block, or specify a comment for the instance, expand the POU definition by clicking .
4. To assign a variable to an input or output, click on the required item, then do one of the following:
   - Type the name of the variable in the field.
   - Click , then select the variable from the drop-down list.
5. For blocks having structure elements within arguments, click ..., then provide the required values in the fields.

**To define visual settings for a function block instance**

1. In the POU, click  in the upper-right corner of the block instance.

   The block configurator window for the block instance is displayed.

2. Click the Visual Settings tab.

3. To set the background color or background gradient color for the instance, click the color swatch for the respective item, then from the color picker, choose or specify the required color.

   You can also reset the background color or background gradient color for the instance.

4. To display the instance name in the block, select *Display Instance Name*.

5. Choose pins to display for the instance:

   - To mask unconnected pins, click **Hide Unconnected Pins**.
   - To display all connected and unconnected pins, click **Show All Pins**.
   - To specify individual pins to make visible, on the block representation, click the required pins to toggle from *Hidden* to *Visible*.

6. Choose the information to display for the instance pins:

   - To display the names of pins, click **Display Pin Names**.
   - To display the aliases for the pins, click **Display Pin Aliases**.

   You can also toggle between displaying individual pin names and aliases by clicking the item.

## LD Keyboard Shortcuts

The following keyboard shortcuts are available for use with the LD language. Some shortcuts do not apply or may differ while debugging.

| | |
|---|---|
| Ctrl+A | Selects all rungs (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while debugging) |
| Ctrl+V | Pastes elements saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl+X | Cuts the selected elements to the clipboard (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Ctrl+Alt+G | Enables/disables the grid in rungs |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+R | Toggles between Auto-Input and Manual-Input. Auto-Input automatically opens the Block Selector and Variable Selector (not available while debugging). |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctrl+I | Italicizes selected comment text (not available while debugging) |
| Ctrl+U | Underlines selected comment text (not available while debugging) |
| Enter | Calls the Variable/Block selector depending on the selected element (not available while debugging) |
| F9 | Toggles between setting or removing a breakpoint on a selected rung (available when Generate Debug Info is True). If more than one rung is selected, only sets a breakpoint on the first selected rung. |
| Space Bar | For coils or contacts, toggles between the available types (not available while debugging) |
| Ctrl+0 | Inserts a rung after a selected rung. When no rung is selected, a rung is added at the end of the rung list (not available while debugging). |
| Ctrl+Alt+0 | Inserts a rung before a selected rung. When no rung is selected, a rung is added at the end of the rung list (not available while debugging). |

| | |
|---|---|
| Ctrl+ 1 | Inserts a branch after a selected element (not available while debugging) |
| Ctrl+Alt+ 1 | Inserts a branch before a selected element (not available while debugging) |
| Ctrl+2 | Inserts a block after a selected element. When a branch is selected, a block is inserted on the branch (not available while debugging). |
| Ctrl+Alt+2 | Inserts a block before a selected element. When a branch is selected, a block is inserted on the branch (not available while debugging). |
| Ctrl+3 | Inserts a contact after a selected element. When a branch is selected, a contact is inserted on the branch (not available while debugging). |
| Ctrl+Alt+3 | Inserts a contact before a selected element. When a branch is selected, a contact is inserted on the branch (not available while debugging). |
| Ctrl+4 | When a rung or the last element on a rung is selected, inserts a coil at the end of the rung. When the last element selected on a rung is a branch, a coil is inserted on the branch (not available while debugging). |
| Ctrl+Alt+4 | When a rung or the last element on a rung is selected, inserts a coil at the end of the rung. When the last element selected on a rung is a branch, a coil is inserted on the branch (not available while debugging). |
| Ctrl+5 | When a rung or the last element on a rung is selected, inserts a jump at the end of the rung. When the last element selected on a rung is a branch, a jump is inserted on the branch (not available while debugging). |
| Ctrl+Alt+5 | When a rung or the last element on a rung is selected, inserts a jump at the end of the rung. When the last element selected on a rung is a branch, a jump is inserted on the branch (not available while debugging). |
| Ctrl+6 | When a rung or the last element on a rung is selected, inserts a return at the end of the rung. When the last element selected on a rung is a branch, a return is inserted on the branch (not available while debugging). |
| Ctrl+Alt+6 | When a rung or the last element on a rung is selected, inserts a return at the end of the rung. When the last element selected on a rung is a branch, a return is inserted on the branch (not available while debugging). |
| Ctrl+Page Up | Jumps to the top of the language container |

| | |
|---|---|
| Ctrl+Page Down | Jumps to the bottom of the language container |
| Ctrl+Up Arrow | Slowly scrolls up. |
| Ctrl+Down Arrow | Slowly scrolls down. |
| Ctrl+Left Arrow | Slowly scrolls left. |
| Ctrl+Right Arrow | Slowly scrolls right. |
| Up Arrow | Moves up the elements. |
| Down Arrow | Moves down the elements. |
| Left Arrow | Moves to the left across the elements. |
| Right Arrow | Moves to the right across the elements. |
| Alt+Up Arrow | Selects the previous rung. When no element or rung is selected, selects the last rung. |
| Alt+Down Arrow | Selects the next rung. When no element or rung is selected, selects the first rung. |
| Alt+Left Arrow | Selects the rung of the selected element. When no element is selected, selects the first rung. |
| Alt+Right Arrow | Selects the rung of the selected element. When no element is selected, selects the first rung. |
| Shift+Up Arrow | Scrolls up |
| Shift+Down Arrow | Scrolls down |
| Shift+Left Arrow | Scrolls left |
| Shift+Right Arrow | Scrolls right |
| Delete | Removes a selected rung or element (not available while debugging) |
| Ctrl+D | Only available in debug mode for the date data type. When the Write Logical Value dialog box is open, enters the current date. |

# ST Language

ST (Structured Text) is a high level structured language designed for automation processes. This language is mainly used to implement complex procedures that cannot be easily expressed with graphic languages. ST language is also used for the description of the actions within the Steps and conditions attached to the Transitions of the SFC Language.

**See Also**

ST Main Syntax

Debugging ST Programs

# ST Main Syntax

An ST program is a list of ST statements. Each statement ends with a semi-colon (";") separator. Names used in the source code (variable identifiers, constants, language keywords...) are separated with inactive separators (space character, end of line or tab stops) or by active separators, which have a well defined significance (for example, the ">" separator indicates a "greater than" comparison.

Comments enable the inclusion of non-executed information throughout code. You can insert comments anywhere in an ST program. Comments can run multiple lines and must begin with "(*" and end with "*)". You cannot use interleave comments, i.e., comments within comments.

When typing statements, a drop-down combo-box automatically lists the available items such as identifiers, operators, functions, and function blocks. The listed items are filtered by typing letters, digits, and specific special characters: !, #, $, %, &, \, *, +, -, ,/ <, :, =, >, ?, @, \, ^, _, `, |, and ~.

The following are basic types of ST statements:

- assignment statement (variable := expression;)
- function call
- function block call
- selection statements (IF, THEN, ELSE, CASE...) S
- iteration statements (FOR, WHILE, REPEAT...)
- control statements (RETURN, EXIT...)
- special statements for links with other languages

When entering ST syntax, basic coding is black while other items are displayed using customizable colors. The default colors for ST elements are the following:

- Comments are green
- The Editor background is white
- Identifiers are black

- Numbers are firebrick
- Operators are black
- POUs are blue-violet
- Punctuation marks are black
- Reserved words are fuchsia
- Strings of text are gray

Inactive separators between active separators, literals, and identifiers increase ST program legibility. ST inactive separators are the following: space (blank), tabs and end of line. You can place end of lines anywhere in a program. The following rules apply to using inactive separators:

- Write one statement on one line
- Use tabs to indent complex statements
- Insert comments to increase legibility of lines or paragraphs

### Examples

**Low Readability**

```
imax := max_ite; cond := X12;
if not(cond (* alarm *)
then return; end_if;
for i (* index *) := 1 to max_ite
do if i <> 2 then Spcall();
end_if; end_for;
(* no effect if alarm *)
```

**High Readability**

```
(* imax : number of iterations *)
(* i: FOR statement index *)
(* cond: process validity *)
imax := max_ite;
cond := X12;
if not (cond) then
      return;
end_if;
(* process loop *)
for i := 1 to max_ite do
      if i <> 2 then
      Spcall ();
      end_if;
end_for;
```

**To customize the default display settings for ST programs**

1. From the Tools menu, click **Options**.
2. From the Options dialog box, expand **IEC Languages**, and then click **Structured Text (ST)**.
3. Expand the respective category, customize the required setting, then click **OK**.

The customized settings are now the default values for ST programs.

**To customize the display settings for the current ST program**

1. From the View menu, click **Properties Window**

   The Properties Window is displayed.

2. Select the ST Container

**3.** From the Properties Window you can:

- Customize the font for the required item by clicking [...]. The Font dialog box is displayed allowing for customization of the font, text size, bold, italic, strikeout, and underline styles.
- Customize the text color for the required items. The possible colors are custom, web, and system colors.

The customized settings only affect the current ST program.

# Expressions and Parentheses

ST expressions combine ST operators and variable or constant operands. For each single expression (combining operands with one ST operator), the type of the operands must be the same. This single expression has the same data type as its operands, and can be used in a more complex expression. For example:

| | |
|---|---|
| (boo_var1 AND boo_var2) | has BOOL type |
| not (boo_var1) | has BOOL type |
| (sin (3.14) + 0.72) | has REAL type |
| (t#1s23ms + 78) | is an invalid expression |

Parentheses are used to isolate sub parts of an expression and to explicitly order the priority of operations. When no parentheses are given for a complex expression, the operation sequence is implicitly given by the default priority between ST operators.

Expressions are executed from left to right and according to the following operator precedence table:

| **Precedence** | **Operators** | **Symbols** |
|---|---|---|
| 1 (Highest) | Function evaluation | *identifier*(*arguement list*)<br>For example: `MAX(X,Y)` |
| 2 | Negation | `-` |
| | Complement | `NOT` |
| 3 | Multiplication | `*` |
| | Division | `/` |
| 4 | Addition | `+` |
| | Subtraction | `-` |
| 5 | Comparison | `<, >, <=, >=` |
| 6 | Equality | `=` |
| | Inequality | `<>` |
| 7 | Boolean AND | `&, AND` |

| | | |
|---|---|---|
| 8 | Boolean Exclusive OR | `XOR` |
| 9 (Lowest) | Boolean OR | `OR` |

**Examples:**

| | | |
|---|---|---|
| `2 + 3 * 6` | `equals 2+18=20` | because multiplication operator has a higher priority |
| `(2 + 3) * 6` | `equals 5*6=30` | priority is given by parenthesis |

## Evaluating Boolean Expressions

Boolean expressions, executed from left to right, are evaluated only to the extent necessary in determining the resultant value. The evaluation of boolean expressions applies to the operators AND and OR. Evaluating boolean expressions prevents the Virtual Machine from stopping execution, as seen when dividing by 0.

```
(* Reduce Boolean Expression Evaluation property is set to TRUE*)
A := 0;
IF A > 0 AND 4/A = 4 THEN
B := 1;
END_IF;
```

**To evaluate boolean expressions**

1. From the Solution Explorer, select the Resource.
2. In the Properties window, set the *Reduce Boolean Expression Evaluation* property to TRUE.

   Boolean expressions are now evaluated before execution.

# Calling Functions

The ST programming language enables calling functions. Function calls can be used in any expression.

**Name:** name of the called function written in IEC 61131-3 language or in "C"

**Meaning:** calls a ST, LD, or FBD functions or a "C" function and gets its return value

**Syntax:** <**variable**> **:=** <**funct**> **(**<**par1**>**,** ... <**parN**> **);**

**Operands:** The type of return value and calling parameters must follow the interface defined for the function.

**Return value:** value returned by the function

When setting the value of the return parameter in the body of a function, assign the return parameter using the same name as the function:
FunctionName := <expression>;

## Example

Example1: IEC 61131-3 function call

```
(* Main ST program *)
(* gets an integer value and converts it into a limited time value *)
ana_timeprog := SPlimit ( tprog_cmd );
appl_timer := ANY_TO_TIME (ana_timeprog * 100);

(* Called FBD function named 'SPlimit' *)
```

Example2: "C" function call – same syntax as for IEC 61131-3 function calls

```
(* Functions used in complex expressions: min, max, right, mlen and
left are standard "C" functions *)
limited_value := min (16, max (0, input_value) );
rol_msg := right (message, mlen (message) - 1) + left (message, 1);
```

# Calling Function Blocks

The ST programming language enables calling function blocks. Function block calls can be used in any expression.

**Name:** name of the function block instance

**Meaning:** calls a function block from the standard library or from the user's library and accesses its return parameters

**Syntax:** **(* call of the function block *)**
<**blockname**> **(** <**p1**>**,** <**p2**> ... **);**
**(* gets its return parameters *)**
<**result**> **:=** <**blockname**>**.** <**ret_param1**>**;**
...
<**result**> **:=** <**blockname**>**.** <**ret_paramN**>**;**

**Operands:** parameters are expressions which match the type of the parameters specified for that function block

**Return value:** See Syntax to get the return parameters.

When setting the value of the return parameter in the body of a function block, assign the return parameter using its name concatenated with the function block name:
FunctionBlockName.OutputParaName := <expression>;

### Example

```
(* ST program calling a function block *)

(* declare the instance of the block in the dictionary: *)
(* trigb1 : block R_TRIG - rising edge detection *)

(* Function block activation from ST language *)
trigb1 (b1);
(* return parameters access *)
If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;
```

# Debugging ST Programs

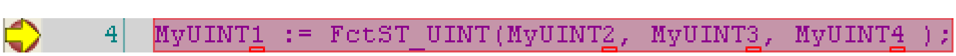
For ST programs, you can enable step-by-step execution by generating debug information for individual POUs. When debug information is generated for ST programs in a resource, the resource automatically switches to step-by-step execution when the application encounters a breakpoint. You instantiate step-by-step execution by setting breakpoints to lines of code. In the language editor, breakpoints appear as red circles to the left of the line of code and the line is highlighted in red.


When debugging, the application stops when it encounters a breakpoint. At this time, the resource is in the DEBUGGING state and you can choose to perform one of the following operations:

- Step into the highlighted line of code, executing the highlighted line of code then stepping into the subsequent line of code. When the next line of code includes a call to a function, stepping continues in the called function then returns to the next line of code in the POU.
- Step over the highlighted line of code, skips the highlighted line of code then steps to the susbsequent line of code
- Switch execution to real-time mode
- Switch execution to cycle-to-cycle mode
- Execute one cycle

**Note:** You can only set breakpoints for TIC POUs; you cannot set breakpoints for C source code POUs.

When a breakpoint is encountered, a yellow arrow is displayed beside the breakpoint and the next line of code is highlighted in pink.



When stepping passes beyond the last line of code of a POU, the arrow points downward.

**To generate debug information for an ST POU**

Generate debug information for ST POUs enables step-by-step debugging within the POU.

1. In the Solution Explorer, select the ST POU for which to generate debug information.
2. In the Properties for the POU, set *Generate Debug Info* to True.

**To set a breakpoint in an ST POU**

- Right-click in the margin to the left of the line of code on which to add a breakpoint, then click **Add Breakpoint**.

A breakpoint is displayed as a red dot to the left of the line of code.

**To remove a breakpoint**

- Right-click in the area to the left of the line of code having a breakpoint to remove, then click **Remove Breakpoint**.

The breakpoint is removed from the line of code.

**To step into the highlighted line of code**

- From the Debug menu, click **Step Into** (or press **F11**).

The POU executes the highlighted line of code then steps into the next one and stepping continues in any called function before returning to the next line of the POU.

**To step over the highlighted line of code**

- From the Debug menu, click **Step Over** (or press **F10**).

The POU skips the highlighted line of code then steps to the next one.

**To switch execution to real-time mode**

- From the Target Execution toolbar, click .

The POU executes in real-time mode.

**To switch execution to cycle-to-cycle mode**

- From the Target Execution toolbar, click .

The POU executes in cycle-to-cycle mode.

**To execute one cycle**

- From the Target Execution toolbar, click .

Executes the remaining POUs until the next cycle.

# ST Basic Elements and Statements

The basic elements and statements of the ST language are the following:

- Assignments
- CASE Statement
- EXIT Statement
- FOR Statement
- IF-THEN-ELSIF-ELSE-END_IF Statement
- REPEAT Statement
- RETURN Statement
- WHILE Statement

**See Also**
ST Main Syntax

## Assignments

**Name:** :=

**Meaning:** Assigns a variable to an expression

**Syntax:** <variable> := <any_expression> ;

**Operands:** Variable must be an internal or output variable and the expression must have the same type

The expression can be a call to a function.

### Example

```
(* ST program with assignments *)

(* variable <<= variable *)
bo23 := bo10;

(* Variable <<= expression *)

bo56 := bx34 OR alrm100 & (level >= over_value);
result := (100 * input_value) / scale;

(* assignment with function call *)
limited_value := min (16, max (0, input_value) );
```

**To insert an Assignment**

- In the language container, type **:=**.

## CASE Statement

**Name:** **CASE ... OF ... ELSE ... END_CASE**

**Meaning:** executes one of several lists of ST statements
selection is made according to an integer expression

**Syntax:** **CASE <integer_expression> OF**
<value> : <statements> ;
<value> , <value> : <statements> ;
...
**ELSE**
<statements> ;
**END_CASE;**

CASE values must be double integer (DINT) constant expressions. You can convert other data types such as reals and long integers using the ANY_TO_DINT data conversion operator.

```
CASE integer_expression OF
ANY_TO_DINT(1): xx := 1;
ANY_TO_DINT(16#FFFFFFFF): xx := 2; (* -1 *)
ANY_TO_DINT(16#10000000): xx := 3; (* 268435456 *)
ANY_TO_DINT(8#777): xx := 4; (* 511 *)
ANY_TO_DINT(2#1111_1111_1111_1111): xx := 5; (* 65535 *)
ANY_TO_DINT(10.0): xx:= 6; (* 10 *)
ELSE
xx := 99999;
END_CASE;
```

Several values, separated by commas, can lead to the same list of statements. The ELSE statement is optional.

### Example

```
(* ST program using CASE statement *)
```

```
CASE error_code OF
  255: err_msg := 'Division by zero';
fatal_error := TRUE;
  1: err_msg := 'Overflow';
  2, 3: err_msg := 'Bad sign';
ELSE
  err_msg := 'Unknown error';
END_CASE;
```

**To insert a CASE**

- From the Toolbox, drag the **CASE** element into the language container.

## EXIT Statement

**Name:** **EXIT**

**Meaning:** exit from a FOR, WHILE or REPEAT iteration statement

**Syntax:** **EXIT**;

The EXIT is commonly used within an IF statement, inside a FOR, WHILE or REPEAT block.

### Example

```
(* ST program using EXIT statement *)
(* this program searches for a character in a string *)

length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
  code := ascii (message, index);
  IF (code = searched_char) THEN
     found := YES;
     EXIT;
  END_IF;
END_FOR;
```

**To insert an EXIT**

- In the language container, type **EXIT**.

## FOR Statement

**Name:** **FOR ... TO ... BY ... DO ... END_FOR**

**Meaning:** executes a limited number of iterations, using an integer index variable

**Syntax:** **FOR <index> := <mini> TO <maxi> BY <step> DO**
<statement> ;
<statement> ;
**END_FOR;**

**Operands:** **index**: internal integer variable increased at each loop
**mini**: initial value for index (before first loop)
**maxi**: maximum allowed value for index
**step**: index increment at each loop

The [ BY step ] statement is optional. If not specified, the increment step is 1

**Warning:** Because the virtual machine is a synchronous system, input variables are not refreshed during FOR iterations.

This is the "WHILE" equivalent of a FOR statement:

```
index := mini;
while (index <= maxi) do
  <statement> ;
  <statement> ;
  index := index + step;
end_while;
```

### Example

```
(* ST program using FOR statement *)
(* this program extracts the digit characters of a string *)

length := mlen (message);
target := ''; (* empty string *)
FOR index := 1 TO length BY 1 DO
  code := ascii (message, index);
  IF (code >= 48) & (code <= 57) THEN
     target := target + char (code);
  END_IF;
END_FOR;
```

**To insert a FOR**

- From the Toolbox, drag the **FOR** element into the language container.

## IF-THEN-ELSIF-ELSE-END_IF Statement

**Name:** **IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF**

**Meaning:** executes one of several lists of ST statements
selection is made according to the value of a Boolean expression

**Syntax:**

```
IF <Boolean_expression> THEN
    <statement> ;
    <statement> ;
    ...
ELSIF <Boolean_expression> THEN
    <statement> ;
    <statement> ;
    ...
ELSE
    <statement> ;
    <statement> ;
    ...
END_IF;
```

The ELSE and ELSIF statements are optional. If the ELSE statement is not written, no instruction is executed when the condition is FALSE. You can use the ELSIF statement more than once. The ELSE statement, if used, must appear only once at the end of the 'IF, ELSIF...' sequence.

When the resource property *Reduce Boolean Expression Evaluation* is set to FALSE, **ISaGRAF** evaluates complete Boolean expressions. For instance, evaluating the following line of code, where i represents the array index having a definition of 2..10, causes a run-time error upon reaching the second part where it applies the value 1 as the array index.

```
IF i >= 2 and i <= 10 and matrix [i] > 5 THEN
```

You can avoid this type of error by using the following code:

```
IF i >= 2 and i <= 10 THEN
IF Array1[i] THEN
```

### Example

```
(* ST program using IF statement *)
```

```
IF manual AND not (alarm) THEN
  level := manual_level;
  bx126 := bi12 OR bi45;
ELSIF over_mode THEN
  level := max_level;
ELSE
level := (lv16 * 100) / scale;
END_IF;

(* IF structure without ELSE *)
If overflow THEN
  alarm_level := true;
END_IF;
```

**To insert an IF-THEN-ELSIF-ELSE-END_IF**

- From the Toolbox, drag the **IF THEN ELSE** element into the language container.

**See Also**

Expressions and Parentheses

## REPEAT Statement

**Name:** **REPEAT ... UNTIL ... END_REPEAT**

**Meaning:** iteration structure for a group of ST statements
the "continue" condition is evaluated AFTER any iteration

**Syntax:** **REPEAT**
<statement> ;
<statement> ;
...
**UNTIL <Boolean_condition>**
**END_REPEAT ;**

**Warning:** Because the virtual machine is a synchronous system, input variables are not refreshed during REPEAT iterations. The change of state of an input variable cannot be used to describe the ending condition of a REPEAT statement.

### Example

```
(* ST program using REPEAT statement *)

(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

str := ''; (* empty string *)
nbchar := 0;
IF ComIsReady ( ) THEN
  REPEAT
     str := str + ComGetChar ( );
     nbchar := nbchar + 1;
  UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )
  END_REPEAT;
END_IF;
```

**To insert a REPEAT**

- From the Toolbox, drag the **REPEAT** element into the language container.

## RETURN Statement

**Name:** **RETURN**

**Meaning:** terminates the execution of the current program

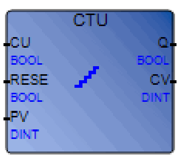
**Syntax:** **RETURN** ;

**Operands:** (none)

In an SFC action block, the RETURN statement indicates the end of the execution of that block only.

### Example

(* FBD specification of the program: programmable counter *)




```
(* ST implementation of the program, using RETURN statement *)

If NOT (CU) then
  Q := false;
  CV := 0;
  RETURN; (* terminates the program *)
end_if;

if RESET then
  CV := 0;
else
  if (CV < PV) then
     CV := CV + 1;
  end_if;
end_if;
Q := (CV >= PV);
```

**To insert a RETURN**

- In the language container, type **RETURN**.

## WHILE Statement

**Name:** **WHILE ... DO ... END_WHILE**

**Meaning:** iteration structure for a group of ST statements
the "continue" condition is evaluated BEFORE any iteration

**Syntax:** **WHILE <Boolean_expression> DO**
<statement> ;
<statement> ;
...
**END_WHILE ;**

**Warning:** Since the virtual machine is a synchronous system, input variables are not refreshed during WHILE iterations. The change of state of an input variable cannot be used to describe the condition of a WHILE statement.

### Example

```
(* ST program using WHILE statement *)

(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

str := ''; (* empty string *)
nbchar := 0;

WHILE ((nbchar < 16) & ComIsReady ( )) DO
  str := str + ComGetChar ( );
  nbchar := nbchar + 1;
END_WHILE;
```

**To insert a WHILE**

- From the Toolbox, drag the **WHILE** element into the language container.

# ST Extensions

The following statements and functions are available to control the execution of SFC child programs. You can use these within action blocks written in ST for SFC steps.

| | |
|---|---|
| GSTART | starts an SFC program or function block |
| GFREEZE | freezes an SFC program |
| GKILL | terminates an SFC program |
| GSTATUS | gets current status of an SFC program |
| GRST | restarts a frozen SFC program or function block |

**Warning:** These functions are not part of the IEC 61131-3 standard.

Simple equivalents for the GSTART and GKILL statements are available using the following syntax in an SFC step:

- child_name with the S qualifier (* equivalent to GSTART(child_name); *)
- child_name with the R qualifier (* equivalent to GKILL(child_name); *)

The following fields enable accessing the status of an SFC step or child (from its parent):

| | |
|---|---|
| StepName.x | Boolean value that represents the **activity of the Step** |
| StepName.t | time elapsed since the last activation of the step: **activity duration** ("**StepName**" represents the name of the SFC step) |
| ChildName.__S1.x | Boolean value that represents the **activity of the child** |
| ChildName.__S1.t | time elapsed since the last activation of the step: **activity duration** ("**ChildName**" represents the name of the SFC child) |

## GSTART Statement in SFC Action

**Name:** **GSTART**

**Meaning:** Starts an SFC child program or function block by placing a token into each of its initial steps. The abbreviated syntax is equivalent to an SFC Child action block having the S qualifier. The extended syntax only applies to SFC child function blocks.

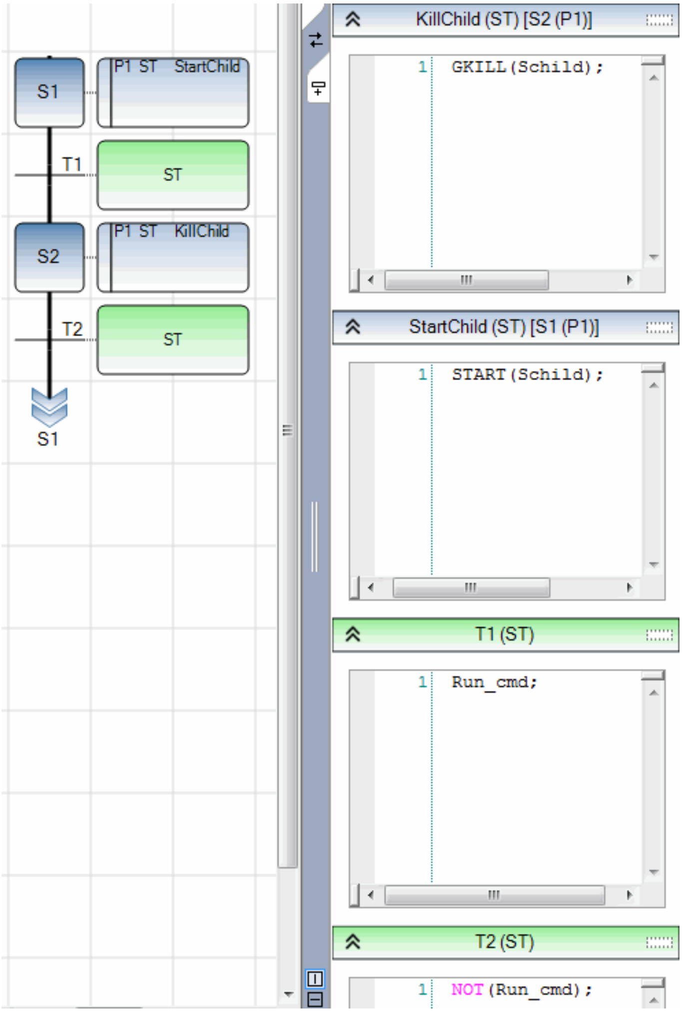
**Syntax:** **GSTART** ( <*child_name*> );
or
**GSTART** ( <*child_name,step_name,input1,input2,...inputn*> )
where
*child_name* represents the name of the SFC child POU
*step_name* represents the name of the active step. *step_name* must be preceded by two underscore characters (e.g., __S1*)*
*input1,input2,...inputn* indicate the values of the input parameters of the SFC child POU

**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** (none)

Children of the child program are not automatically started by the GSTART statement. Since GSTART is not part of the IEC 61131-3 standard, it is preferable to use the S qualifier attached to the child name.

## Example




### To insert a GSTART

- In the language container, type **GSTART**.

## GFREEZE Statement in SFC Action

**Name:** **GFREEZE**

**Meaning:** freezes a child SFC (program or function block); suspends its execution. The suspended SFC POU can then be restarted using the GRST statement.

**Syntax:** **GFREEZE (** <*child_name*> **);**
where
*child_name* represents the name of the SFC child POU

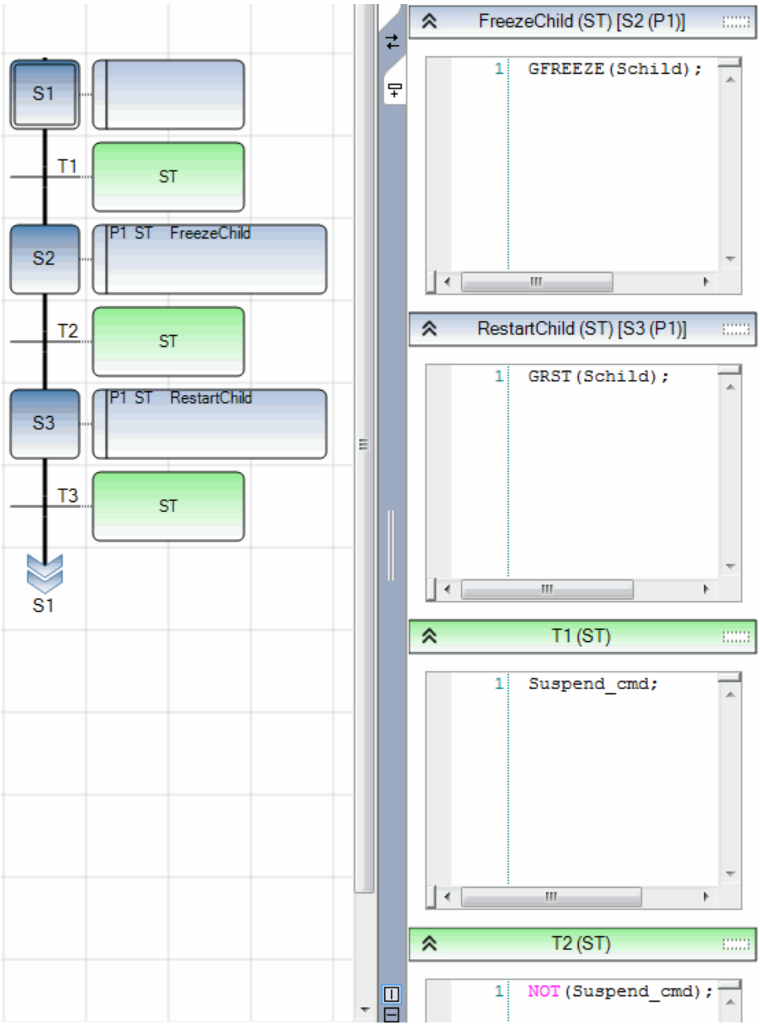
**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** (none)

Children of the child program are automatically frozen along with the specified program.

GFREEZE is not part of the IEC 61131-3 standard.

## Example




## To insert a GFREEZE

- In the language container, type **GFREEZE**.

## GKILL Statement in SFC Action

**Name:** **GKILL**

**Meaning:** Terminates a child SFC program by removing the Tokens currently existing in its Steps. The syntax is equivalent to an SFC Child action block having the R qualifier.

**Syntax:** **GKILL** ( <*child_name*> );
where
*child_name* represents the name of the SFC child POU

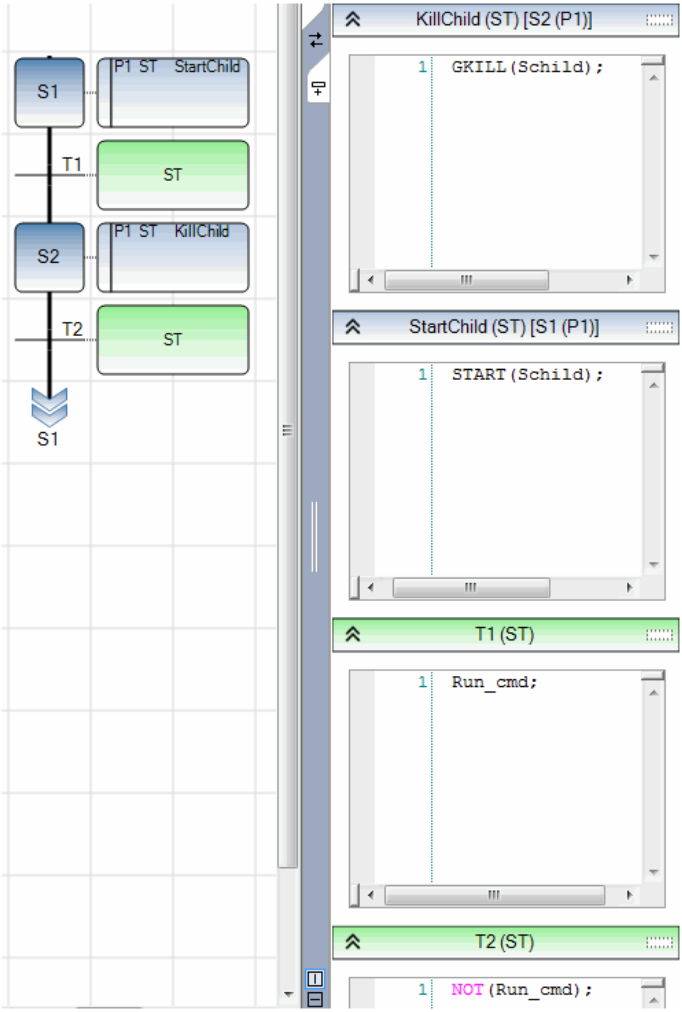
**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** (none)

Children of the child program are automatically terminated with the specified program.

Since GKILL is not part of the IEC 61131-3 standard, it is preferable to use the R qualifier attached to the child name.

### Example




**To insert a GKILL**

- In the language container, type **GKILL**.

## GSTATUS Statement in SFC Action

**Name:** **GSTATUS**

**Meaning:** returns the current status of an SFC program

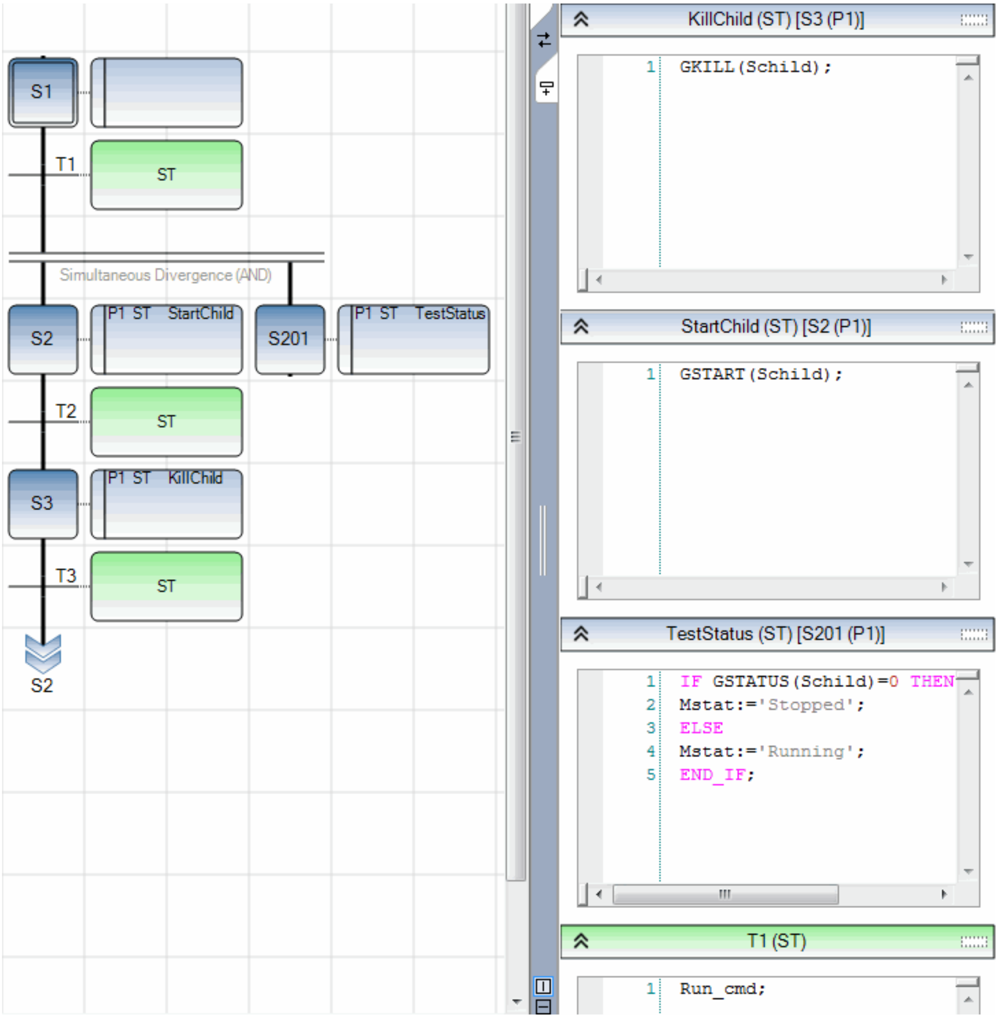
**Syntax:** **<var> := GSTATUS ( <*child_name*> );**
where
*child_name* represents the name of the SFC child POU

**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** 0 = Program is inactive (killed)
1 = Program is active (started)
2 = Program is frozen

GSTATUS is not part of the IEC 61131-3 standard.

## Example




**To insert a GSTATUS**

- In the language container, type **GSTATUS**.

## GRST Statement in SFC Action

**Name:** **GRST**

**Meaning:** restarts a child SFC program frozen by the GFREEZE statement: all the tokens removed by GFREEZE are restored. The extended syntax only applies to SFC child function blocks.

**Syntax:** **GRST** ( <*child_name*> );
or
**GRST** ( <*child_name,input1,input2,...inputn*> );
where
*child_name* represents the name of the SFC child POU
*input1,input2,...inputn* indicate the value of the input parameter of the SFC child POU

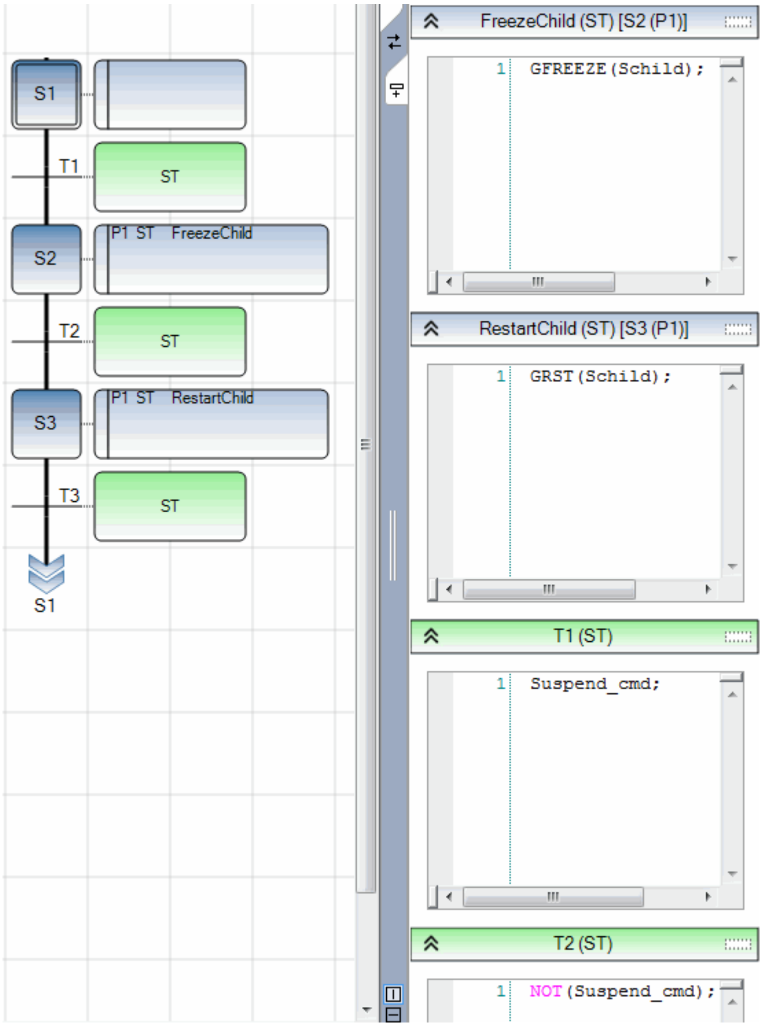
**Operands:** the specified SFC program must be a child of the one in which the statement is written

**Return value:** (none)

The GRST statement automatically restarts children of the child program.

GRST is not part of the IEC 61131-3 standard.

## Example




### To insert a GRST

- In the language container, type **GRST**.

# ST Keyboard Shortcuts

The following keyboard shortcuts are available for use with the ST language. Some shortcuts do not apply or may differ while debugging.

| | |
|---|---|
| Ctrl+A | Selects the entire document (not available while debugging) |
| Ctrl+C | Copies the selected text to the clipboard (not available while debugging) |
| Ctrl+Insert | Copies the selected text to the clipboard (not available while debugging) |
| Ctrl+V | Pastes text saved on the clipboard to the insertion point (not available while debugging) |
| Shift+Insert | Pastes text saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl+X | Cuts the selected text to the clipboard (not available while debugging) |
| Shift+Delete | Cuts the selected text to the clipboard (not available while debugging) |
| Ctrl+L | Cuts the current line to the clipboard (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Shift+Z | Redoes the previous command (not available while debugging) |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Insert | Toggles between the overwrite/insert typing mode |
| Shift+Enter | Inserts a line break. While debugging, when the insertion point is on a variable it opens the Write Logical Value dialog box. |
| Ctrl+Enter | Inserts a line above the current line. While debugging, when the insertion point is on a variable it opens the Write Logical Value dialog box. |

| | |
|---|---|
| Ctrl+Shift+Enter | Inserts a line below the current line. While debugging, when the insertion point is on a variable it opens the Write Logical Value dialog box. |
| Ctrl+Shift+T | Transposes the current and previous word (not available while debugging) |
| Ctrl+Shift+Alt+T | Transposes the current and next line (not available while debugging) |
| Ctrl+Space | Displays a drop-down combo-box listing available items such as variables, operators, functions, and function blocks. You can filter displayed items by typing letters, digits, and specific special characters: !, #, $, %, &, \, *, +, -, ,/ <, :, =, >, ?, @, \, ^, _, `, \|, and ~. (not available while debugging) |
| Ctrl+Shift+Space | Displays a drop-down combo-box listing available items such as variables, operators, functions, and function blocks. You can filter displayed items by typing letters, digits, and specific special characters: !, #, $, %, &, \, *, +, -, ,/ <, :, =, >, ?, @, \, ^, _, `, \|, and ~. (not available while debugging) |
| Ctrl+Shift+U | Changes the selected text into uppercase (not available while debugging) |
| Ctrl+U | Changes the selected text into lowercase (not available while debugging) |
| Up Arrow | Moves up lines and characters |
| Down Arrow | Moves down lines and characters |
| Left Arrow | Moves left across lines and characters |
| Right Arrow | Moves right across lines and characters |
| Ctrl+Left Arrow | Moves to the previous statement or word |
| Ctrl+Right Arrow | Moves to the next statement or word |
| Home | Jumps to the start of the line |
| End | Jumps to the end of the line |
| Ctrl+Home | Jumps to the start of the document |
| Ctrl+End | Jumps to the end of the document |
| Page Up | Jumps to the top of the visible code |

| | |
|---|---|
| Page Down | Jumps to the bottom of the visible code |
| Ctrl+Page Up | Jumps to the top of the visible code |
| Ctrl+Page Down | Jumps to the bottom of the visible code |
| Ctrl+Up Arrow | Scrolls up |
| Ctrl+Down Arrow | Scrolls down |
| Shift+Up Arrow | Selects up |
| Shift+Down Arrow | Selects down |
| Shift+Left Arrow | Selects left |
| Shift+Right Arrow | Selects right |
| Ctrl+Shift+Left Arrow | Selects to the previous statement or word |
| Ctrl+Shift+Right Arrow | Selects to the next statement or word |
| Shift+Home | Selects from the insertion point until the start of the line |
| Shift+End | Selects from the insertion point until the end of the line |
| Ctrl+Shift+Home | Selects from the insertion point until the start of the document |
| Ctrl+Shift+End | Selects from the insertion point until the end of the document |
| Ctrl+Shift+Page Up | Selects from the insertion point until the top of the visible code |
| Ctrl+Shift+Page Down | Selects from the insertion point until the end of the visible code |
| Ctrl+Shift+W | Selects the next word |
| Shift+Alt+Up Arrow | Selects the current and previous lines |
| Shift+Alt+Down Arrow | Selects the current and next lines |
| Shift+Alt+Left Arrow | Selects left on the current line |
| Shift+Alt+Right Arrow | Selects right on the current line |
| Ctrl+Shift+Alt+Left Arrow | Selects available columns in lines of code from the left to right |
| Ctrl+Shift+Alt+Right Arrow | Selects available columns in lines of code from the right to left |
| Escape | Deselects the selected text |
| Ctrl+I | Opens the Variable Selector. While debugging, opens the Variable Monitoring dialog box. |
| Ctrl+Shift+I | Opens the Variable Selector. While debugging, opens the Variable Monitoring dialog box. |

| | |
|---|---|
| Ctrl+R | Opens the Block Selector. When the insertion point is on a variable during debugging, it is selected. |
| Ctrl+Alt+R | Opens the Block Selector. When the insertion point is on a variable during debugging, it is selected. |
| Ctrl+Shift+Alt+R | Opens the Block Selector. When the insertion point is on a variable during debugging, it is selected. |
| Delete | Removes the character on the right (not available while debugging) |
| Ctrl+Shift+L | Removes the current line (not available while debugging) |
| Ctrl+Delete | Removes the next word in the current line (not available while debugging) |
| Ctrl+Backspace | Removes the previous word in the current line (not available while debugging) |
| Backspace | Removes the character on the left (not available while debugging) |
| Shift+Backspace | Removes the character on the left (not available while debugging) |

# SFC Language

The SFC language is a graphic language used to describe operations of a sequential process. This language uses a simple graphic representation for the different steps of a process, and conditions that enable the change of active steps.

SFC is the core of the IEC 61131-3 standard. The other languages (other than Flow Chart) usually describe the actions within the steps and the logical conditions for the transitions.

**See Also**

SFC Main Format
SFC Execution Behavior
SFC Program Hierarchy
Child SFC POUs
Debugging SFC Programs
SFC Elements

# SFC Main Format

An SFC program is a graphic set of steps and transitions, linked together using oriented links. Divergences and convergences represent multiple connection links from 1 to *n* and *n* to 1 respectively. The basic graphic rules of SFC are the following:

- SFC programs must have at least one initial step
- A step must follow a transition
- A transition must follow a step

SFC programs describe sequential operations, where the time variable explicitly synchronizes basic operations. These are called sequential programs. Programs before and after SFC programs describe cyclic operations and are not time-dependent. These are called cyclic programs. Main sequential programs (at the top of the hierarchy) are executed according to the SFC dynamic behavior. Cyclic programs are systematically executed at the beginning of each run time cycle. In Programs sections, sequential programs are grouped together.

Main sequential programs are described with the SFC language; Cyclic programs cannot be described with the SFC language. Any SFC program can have one or more SFC child programs.

Functions and function blocks can be called from actions or conditions of SFC programs.

SFC programs and SFC child programs have dynamic behavior limits set at the resource level. These dynamic behavior limits determine the amount of memory, allocated by a target at initialization time, designated to manage SFC dynamic behavior (i.e. token moving). The amount of allocated memory is calculated as a linear relation with the number of SFC POUs:

```
Alloc Mem (bytes) = N * NbElmt * sizeof(typVa)
```

```
NbElmt = GainFactor * NbOfSFC + OffsetFactor
```

Where:

N = 5 (constant linked to SFC engine design)
NbElmt = The maximum number of transitions that can be valid for each executed cycle, i.e., transitions with at least one of their previous steps being active.
typVa = 16 bits in the medium memory model (32 bits in the large memory model)
GainFactor and OffsetFactor = the linear parameters of the linear relation
NbOfSFC = the number of SFC POUs in the project

The following points offer a simplified and more approximate definition of the allocated memory:

- The maximum number of steps that can be active
- The maximum number of actions (N, P1 or P0 action linked to the step) that can be executed

When the available memory is insufficient at a specific moment for a target where check mode (ITGTDEF_SFCEVOCHECK defined in dsys0def.h) is generated, the target kernel generates a warning. This warning signals an SFC token moving error or an action execution error and the resource is set to ERROR mode, i.e., cycles are no longer executed or kernel behavior may become unpredictable.

For SFC function blocks and SFC child function blocks, each has a maximum number of tokens which is set in the properties of the block.

SFC function block instances, as their SFC child blocks, have a maximum number of tokens, unlike SFC programs whose dynamic behavior limits are set at the resource level.

**See Also**
SFC Execution Behavior
SFC Program Hierarchy
Child SFC POUs
Debugging SFC Programs
SFC Elements

## SFC Execution Behavior

The SFC execution behavior consists of three stages: initial situation (start), code execution, and end. Each virtual machine cycle consists of determining all clearable transitions and executing their active steps. Execution ends upon reaching unclearable transitions or the end of the control chart.

Within the execution cycle, the dynamic behaviors of the SFC language are the following:

**Initial situation**

The Initial Situation is characterized by the initial steps which are, by definition, in the active state at the beginning of the operation. At least one initial step must be present in each SFC program.

**Clearing of a transition**

A transition has three properties: enabled/disabled, active/inactive, and clearable/non-clearable. A transition is enabled when all immediately preceding steps linked to its corresponding transition symbol are active, otherwise, the transition is disabled. A transition is active if its condition is True.

A transition is clearable if it is enabled and active at the same time. When a transition is clearable, the steps immediately preceding it become inactive and those immediately following it become active. When transitions follow a divergence, multiple transitions may become clearable.

**Changing of state of active steps**

The clearing of a transition simultaneously leads to the active state of the immediately following steps and to the inactive state of the immediately preceding steps. The code within a step is only executed if the step is active.

**Simultaneous clearing of transitions**

All transitions (of all SFC programs) that can be cleared (enabled and active), are simultaneously cleared.

However, for transitions following divergences, the only transition that is cleared is the one having the highest priority among those that are enabled and active.

**End**

The End is characterized by reaching the end of clearable transitions or the end of the control chart.

### See Also

SFC Language

# SFC Program Hierarchy

The system enables the description of the vertical structure of SFC programs. SFC programs are organized in a hierarchical-tree structure. Each SFC program can control (start, terminate,...) other SFC programs. Such programs are called children of the SFC program which controls them. SFC programs are linked together into a main hierarchy tree, using a "parent - child" relationship:

**Parent** Program

**Child** Program

The basic rules implied by the hierarchy structure are:

- SFC programs having no parent are called "main" SFC programs
- Main SFC programs are activated by the system when the application starts
- A program can have several child programs
- A child of a program can only have one parent
- A child program can only be controlled by its parent
- A program cannot control the children of one of its own children

The basic actions that a parent SFC program can take to control its child program are:

| | |
|---|---|
| Start (GSTART) | Starts the child program: activates each of its initial steps. Children of this child program are not automatically started. |
| Terminate (GKILL) | Terminates the child program by deactivating each of its active steps. All the children of the child program are also terminated. |
| Freeze (GFREEZE) | Deactivates each of the active steps of the program, and memorizes them so the program can be restarted. All the children of the child program are also frozen. |

| | |
|---|---|
| Restart (GRST) | Restarts a frozen SFC program by reactivating all the suspended steps. Children of the program are not automatically restarted. |
| Get status (GSTATUS) | Gets the current status (active, inactive or frozen) of a child program. |

**See Also**

SFC Language

# Child SFC POUs

Any SFC POU may control other SFC POUs. Such low level units are called child SFC POUs. A child SFC POU is a parallel unit that can be started, terminated, frozen, or restarted by its parent. The parent POU and child POU must both be described with the SFC language. A child SFC POU can have local variables.

When a parent POU starts a child SFC, it puts an SFC token (activates) into each initial step of the child. This command is described with the GSTART statement or with the name of the child with the S qualifier. When a parent POU terminates a child SFC, it clears all the tokens existing in the steps of the child. Such a command is described with the GKILL statement or with the name of the child and the R qualifier. When a parent POU starts a child, the parent continues its execution.

When a parent POU freezes a child SFC, it clears all the tokens existing in the child, and keeps their position in memory. Such a command is described with the GFREEZE statement. When a parent POU restarts a frozen child SFC, it restores all the tokens cleared when the child was frozen. Such a command is described with the GRST statement.

For details about the usage of the GSTART, GKILL, GFREEZE, and GRST statements in SFC child POUs, consult the ST Extensions section.

Child SFC function block instances, as for their SFC function block parents, have a maximum number of tokens, unlike SFC programs whose dynamic behavior limits are set at the resource level. You specify the tokens limit for an SFC function block in its settings properties.

When using an SFC function block with an SFC child, you can access, for read-only purposes, the local values of the child from its parent by entering the child's name and the parameter in an action or transition's code. For example, to access the *Local1* parameter of an SFC child named *FB_Child*, in an action or transition defined for the SFC function block parent, you would write the following syntax:

FB_Child.Local1

# Debugging SFC Programs

When debugging SFC programs, you can visually follow the execution of the individual steps. Steps are colored red while active. You can also place SFC breakpoints on SFC steps or transitions. When a breakpoint is encountered, the resource switches to the BREAK state. This mode is equivalent to the cycle-to-cycle mode. Then to pass the breakpoint, you can choose either to execute one cycle or to switch to real-time mode. When a resource is in the BREAK state and step-by-step execution is activated for other POUs within the resource, you can also step to the first line of the first POU of the resource for which debug information is generated.

**Note:** You can only set breakpoints for TIC POUs; you cannot set breakpoints for C source code POUs. Furthermore, you cannot set or remove SFC breakpoints while a resource is in the STEPPING state.

For steps, two types of breakpoints are available:

- Breakpoint on Step Activation
- Breakpoint on Step Deactivation

Breakpoints appear as red circles with a white "X" displayed on the left part of the step or transition. For steps, breakpoints on activation are displayed at the top corner while breakpoints on deactivation are displayed at the bottom corner.

**To set a breakpoint command on a step or transition**

You set breakpoints onto steps and transitions from the contextual menu. For steps, you can apply a breakpoint on activation or a breakpoint on deactivation.

- Right-click the step or transition, and then click the required breakpoint command.

Once the breakpoint is reached, you can execute one cycle or switch to real-time mode to continue execution.

**To remove breakpoints from steps or transitions**

You remove breakpoints from steps and transitions from the contextual menu.

- Right-click the step or transition, and then click **Remove Breakpoint**.

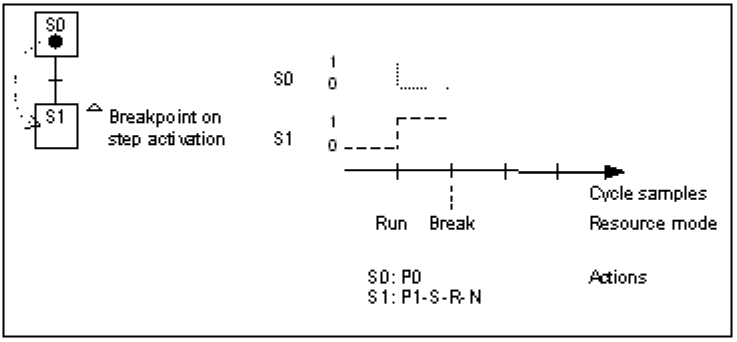
**See Also**
SFC Language

## Breakpoint on Step Activation

When the step goes from the inactive (no token) to the active (token) state, then breakpoint mode is set for the next cycle. The current cycle goes on executing normally. In particular around the step where the breakpoint is placed, before breakpoint mode is really set:

- All P0 actions, linked to all previous steps that become inactive, are executed.
- All P1 – S – R – N actions, linked to the step that becomes active, are executed.

The following illustrates cycle execution when a breakpoint on step activation is encountered.




The behavior of setting a breakpoint on step activation is the same as setting a breakpoint on step deactivation of the previous step. Whether placing a breakpoint on step activation or on deactivation of the previous step, the target executes the break at the same moment.

**To set a breakpoint on step activation**

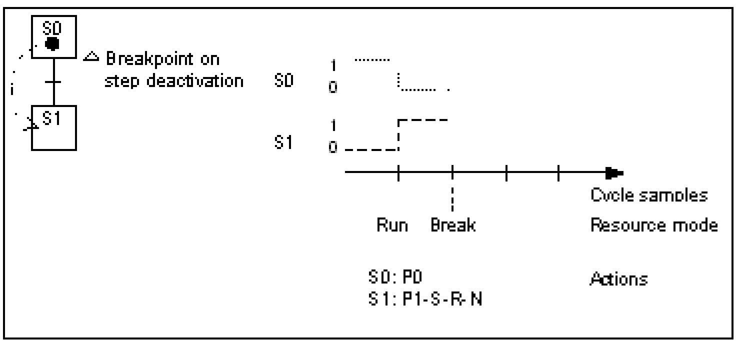
- Right-click the step, and then click **Set Breakpoint on Activation**.

## Breakpoint on Step Deactivation

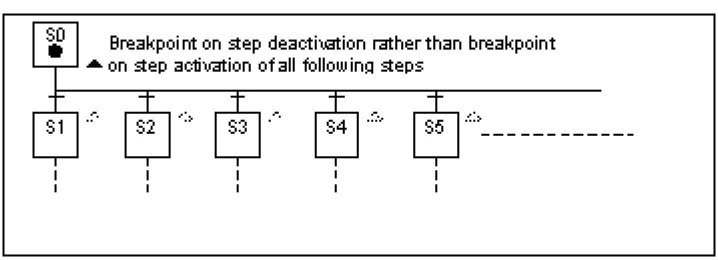
When the step goes from the active (token) to the inactive (no token) state, then breakpoint mode is set for the next cycle. Current cycle goes on executing normally. In particular around the step where the breakpoint is placed, before breakpoint mode is really set:

- All P0 actions, linked to the step that becomes inactive, are executed.
- All P1 – S – R – N actions, linked to all successor steps that become active, are executed.

The following illustrates cycle execution when a breakpoint on step deactivation is encountered.




The behavior of setting a breakpoint on step activation is the same as setting a breakpoint on step deactivation of the previous step. Whether placing a breakpoint on step activation or on deactivation of the previous step, the target executes the break at the same moment. Both breakpoint on step activation and breakpoint on step deactivation are available to avoid setting multiple breakpoints as shown below.



**Note:** On a given step, you cannot set both a breakpoint on step activation and a breakpoint on step deactivation.

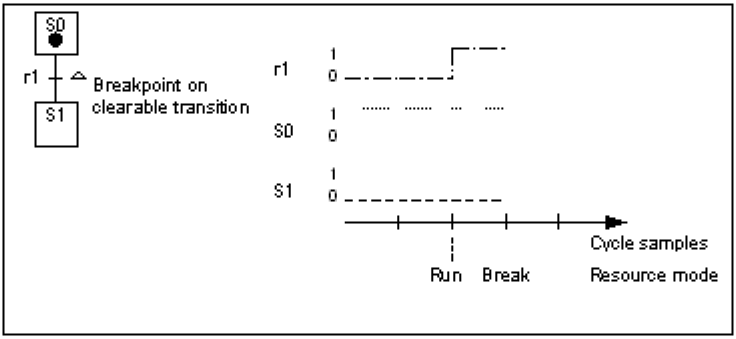
**To set a breakpoint on step deactivation**

- Right-click the step, and then click **Set Breakpoint on Deactivation**.

## Breakpoint on Transition

When a transition becomes clearable (transition is valid i.e. all previous steps are active, and its receptivity is true) then breakpoint mode is set for the next cycle. The current cycle goes on executing normally except that the transition is not cleared and therefore related tokens are not moved.

The following illustrates cycles execution when a breakpoint on transition is encountered.




**To set a breakpoint on a transition**

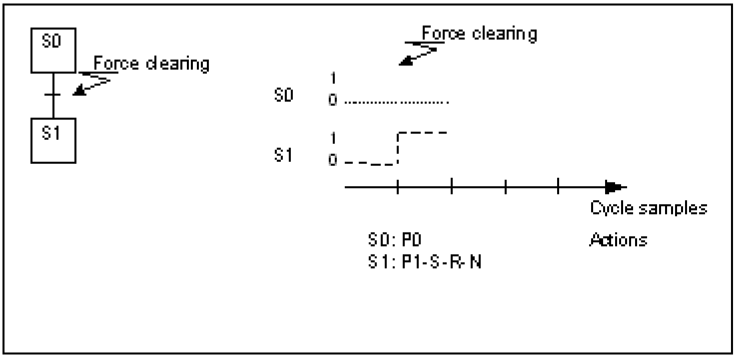
- Right-click the transition, and then click **Set Breakpoint**.

## Transition Clearing Forcing

You can force the clearing of a transition while in simulation whether all previous steps are active or not. The tokens are moved and the actions are executed the same as with usual transition clearings.

Tokens of all predecessor steps are removed and tokens of all successor steps are created. All P0 actions linked to predecessor steps and P1 - S - R - N actions linked to successor steps are executed.

The following illustrates cycle execution when forcing transition clearing:




**Warning:** Clearing a transition may cause abnormal behavior of your chart since several tokens may be created.

**To clear a transition**

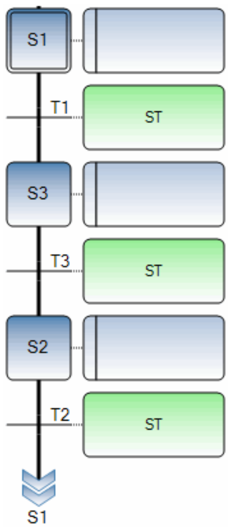
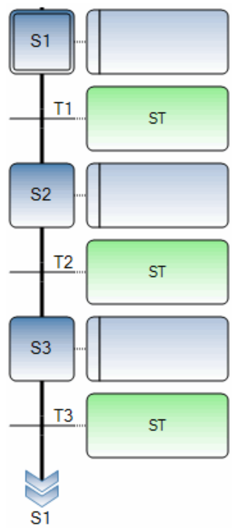
- Right-click the transition while in simulation mode, and then click **Clear Transition**.

# SFC Elements

When working in SFC programs, you can insert the following elements. A program always has an initial step.

- Steps
- Transitions
- Sequence Controls
- Jumps to Steps

When inserting steps and transitions, these are assigned a default naming convention including numbering. For steps, the default naming is S*n* where S indicates a step and *n* indicates the numbering for the step. For transitions, the default naming is T*n* where T indicates a transition and *n* indicates the numbering for the transition. You can rename steps and transitions. However, when renaming steps and transitions using the default naming convention and changing only the numbering, you can renumber these elements to a numbering scheme starting from top to bottom, then from left to right.

| Before Renumbering | After Renumbering |
| --- | --- |
|  |  |





**To renumber steps and transitions**

Renumbering ignores steps and transitions using a naming convention other than the default S*n* for steps and T*n* for transitions.

1. Open the SFC program for which to renumber the steps and transitions.
2. From the Tools menu, point to **Multi-language Editor**, and then click **Renumber Steps and Transitions**.

## See Also

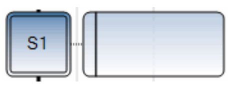
SFC Language

## Steps

SFC programs contain initial steps and steps. Initial steps express the initial situation of an SFC program. Whereas, steps are placed throughout an SFC program. An SFC program must contain at least one initial step. Initial steps and steps are referenced by a name, written in their square symbol. This information is the level 1 of the step.

An initial step has a double bordered graphic symbol.

Initial Step

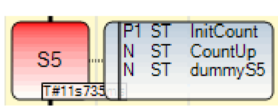



A step is represented by a single square.

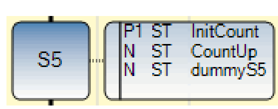
Step


At run time, a token indicates that the step is active. For initial steps, a token is automatically placed in each when the program is started.

Active Step




Inactive Step




Steps have attributes. These can be used in any of the other languages.

StepName.x activity of the Step (Boolean value)
StepName.t activation duration of the Step (time value)

(where StepName is the name of the step)

Activity of a step is an attribute of a step which is activated by an SFC token.

For SFC function blocks, when reading a child active step or duration from a father:

ChildName.__S1.x activity of the Step (Boolean value)
ChildName.__S1.t activation duration of the Step (time value)

(where ChildName is the name of the child. Note that S1 is preceded by two underscore (_)characters)

**To insert an initial step**

- From the Toolbox, drag the initial step element into the language container.

The initial step is displayed in the language container.

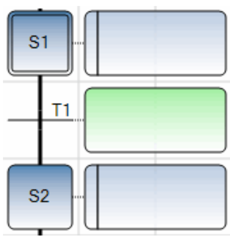
**To insert a step**

- From the Toolbox, drag the step element into the language container.

The step is displayed in the language container.

## Transitions

Transitions are represented by small horizontal bars that cross the connection link. Each transition is referenced by a name, written next to the transition symbol. This information is called the level 1 of the transition.




**To insert a transition**

- From the Toolbox, drag the transition element into the language container.

The transition is displayed in the language container.

# Sequence Controls

Sequence controls are divergences or convergences. These elements adjust automatically to the context of the SFC diagram. For instance, the editor automatically inserts the type of sequence control required according to the elements at the insertion point. Moreover, when adding a parallel element below a sequence control, the sequence control automatically branches out to the added element. Also, when a sequence control is placed erroneously within a diagram, the editor displays it as red.

- Selection Divergences, a multiple link from a step to multiple transitions
- Selection Convergences, a multiple link from multiple transitions to a single step
- Simultaneous Divergences, a multiple link from a transition to multiple steps
- Simultaneous Convergences, a multiple link from multiple steps to a single transition

Divergences are multiple links from one SFC element (step or transition) to multiple SFC symbols. Convergences are multiple connections from more than one SFC symbol to one other symbol.

When inserting a sequence control, the type is determined logically according to the number of SFC elements of a same type (whether multiple) located initially above then below the control.

**To insert a sequence control**

- From the Toolbox, drag the sequence control to the required location in the language container.

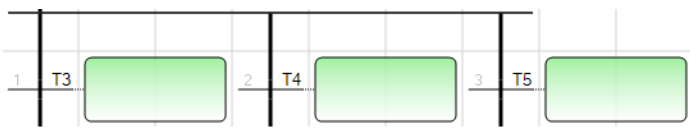
The sequence control is displayed in the language container.

## Selection Divergences

A selection divergence (OR) is a multiple link from one step to multiple transitions. The selection divergence enables an active token to pass into one of a number of branches.

Conditions attached to the different transitions at the beginning of a selection divergence are not implicitly exclusive. Exclusivity of transitions is defined by the priorities set to those transitions following the divergence.

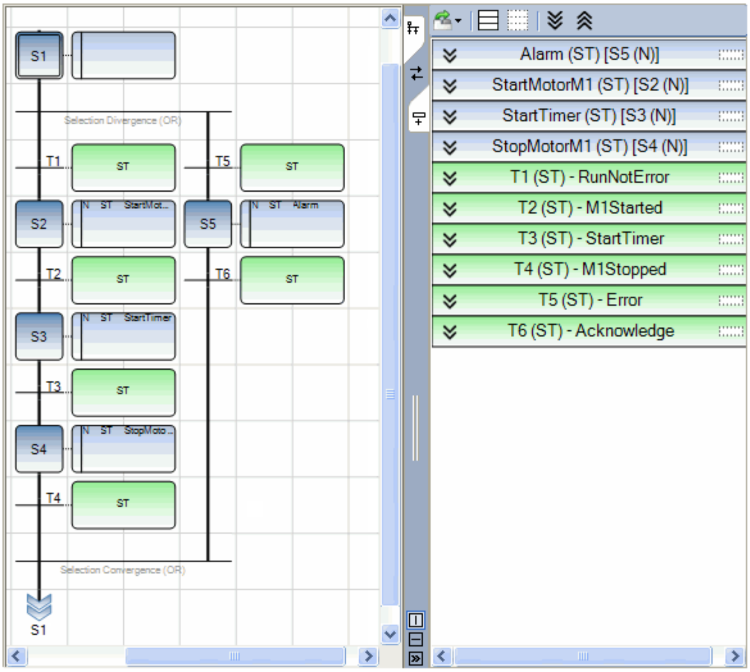
Selection divergences are represented by single horizontal lines.




The first transitions following a single divergence are set in a group to define their priority of execution. The workbench automatically assigns the priority of transitions, displayed on the left, in the order of creation of the divergence branch. You can specify a different priority for a transition in the properties. The possible priority values range from 1 to 255.

### Example

(* SFC Program with selection divergence and convergence *)



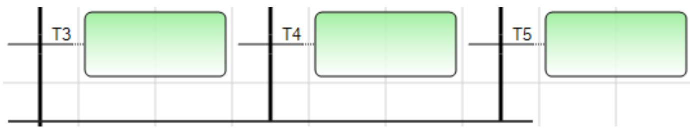

**See Also**

Selection Convergences

Simultaneous Divergences

## Selection Convergences

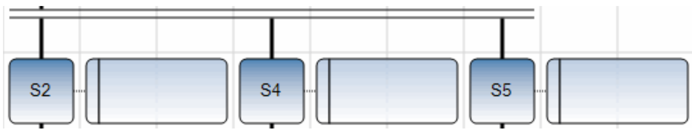
A selection convergence (OR) is a multiple link from multiple transitions to a single step. Selection convergences are generally used to group branches which were started using selection divergences. Selection convergences are represented by single horizontal lines.




**See Also**
Selection Convergences
Simultaneous Convergences

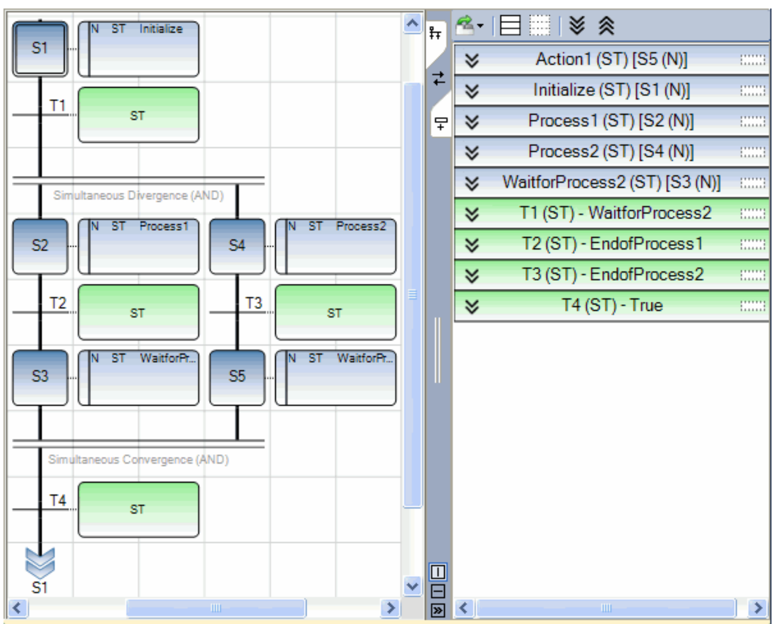
## Simultaneous Divergences

A simultaneous divergence (AND) is a multiple link from one transition to multiple steps. A simultaneous divergence corresponds to parallel operations of a process. Simultaneous divergences are represented by double horizontal lines.




### Example

(* SFC program with simultaneous divergence and convergence *)

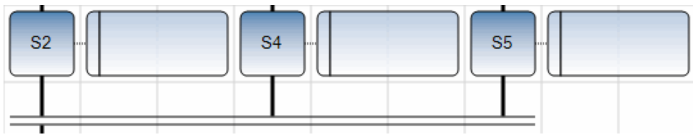



### See Also

Simultaneous Convergences
Selection Divergences

## Simultaneous Convergences

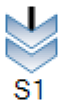
A simultaneous convergence (AND) is a multiple link from multiple steps to a single transition. Simultaneous convergences are generally used to group branches which were started using simultaneous divergences. Simultaneous convergences are represented by double horizontal lines.




**See Also**
Simultaneous Divergences
Selection Convergences

## Jumps to Steps

Jump symbols may be used to indicate a connection link from a transition to a step, without having to draw the connection line. The jump symbol must be referenced with the name of the destination step. A Jump symbol cannot represent a link from a step to a transition.




Jump to Step S1

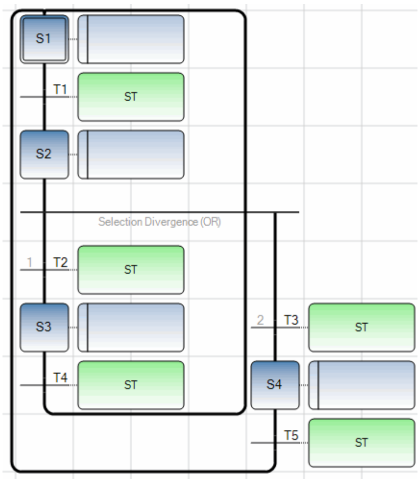
**To insert a jump to a step**

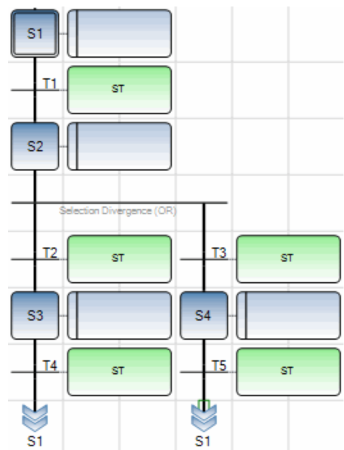
1. From the Toolbox, drag the jump element into the language container and place it directly below the existing transition.
2. In the language container, click the jump element.
3. In the drop-down combo-box, click the desired step.

The jump is displayed in the language container.

### Example

The following charts are equivalent. The chart on the left uses links to return from the bottom to the top of the chart while the chart on the right uses jumps to return to the top of the chart.



# Coding Action Blocks for Steps

Action blocks are operations executed when a step is active. Steps can contain multiple action blocks of the same or different type. You add action blocks to the level 1 of a step. Depending on the action block type, you may need to program the level 2 for the block. You program level 2 code for an action block in a level 2 window, displayed to the right of the POU. The available action block types are the following:

- Boo where the action block name is automatically associated to Boolean variable selected from the variable selector. Possible qualifiers are Action (N), Reset (R), and Set (S).
- LD where you program an LD diagram in the level 2 window. Possible qualifiers are Action (N), Pulse on Deactivation Action (P0), and Pulse On Activation Action (P1).
- SFC where the action block name is automatically associated to the SFC child. Possible qualifiers are Action (N), Reset (R), and Set (S).
- ST where you define ST code in the level 2 window. Possible qualifiers are Action (N), Pulse on Deactivation Action (P0), and Pulse On Activation Action (P1).

Individual SFC steps are executed in the following order:

1. Step activation - beginning when the previous transition is cleared. During this period, defined action blocks are executed in the order of appearance.
2. Step cycle - beginning when the step becomes active and ending when the step completes deactivation. During this period, defined action blocks are executed in the order of appearance.
3. Step deactivation - ending when the following transition becomes active. During this period, defined action blocks other than Boolean (Boo) action blocks having the N qualifier are executed in the order of appearance. Boolean (Boo) action blocks are executed after all other action blocks.

**To add action blocks to steps**

1. Select the step for which to define operations.
2. Right-click the step, point to **Add**, and then click the required action block type.

3. Specify the required properties for the action block from the Properties window by clicking the action block definition on the step.

    a) To rename the action block, type the required text in the Name field.

**Note:** The names for Boo and SFC action blocks are automatically associated to their respective assignation (Boolean variable or SFC child).

    b) To specify the qualifier for the action block, choose the required type in the Qualifier field.

    c) To include a comment, type the required text in the Comment field.

4. For a Boo action block, double-click the action block name, then from the Variable Selector, select the variable for use in the block.

5. For an ST or LD action block, access the level 2 for the block by double-clicking the action block name on the step, then program the required level 2 operations in the level 2 window displayed to the right of the POU.

**To rearrange the order of action blocks for a step**

1. On the step, select the action block to displace.

2. Right-click the action block, and then click **Move Up** or **Move Down**.

**To delete an action block**

1. On the step, select the action block to remove.

2. Right-click the action block, and then click **Delete**.

## Boolean Actions

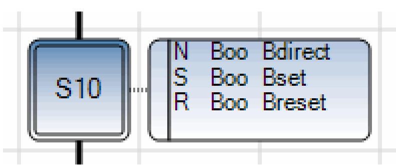
Boolean (Boo) actions assign a Boolean variable to the activity of the Step. The Boolean variable can be a VarInput or VarOutput variable. The variable is assigned each time the step activity starts or stops. The operation for Boolean actions differs for the different qualifiers:

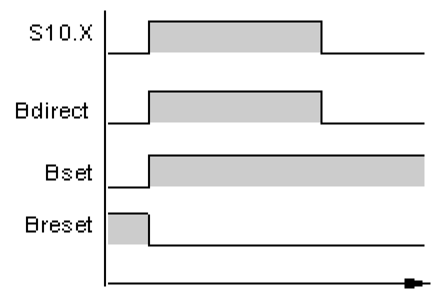
**N on a Boolean Variable** assigns the step activity signal to the variable

**S on a Boolean Variable** sets the variable to TRUE when the step activity signal becomes TRUE

**R on a Boolean Variable** resets the variable to FALSE when the step activity signal becomes TRUE

The Boolean variable must be VarInput or VarOutput. The following SFC programming leads to the indicated behavior:



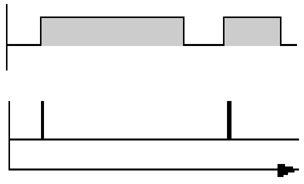



## Pulse Actions

A pulse action is a list of instructions which are executed only once at the activation of the step: P1 qualifier, or executed only once at deactivation of the step: P0 qualifier. Instructions are written using the ST or LD syntax. The following shows the results of a pulse action with the P1 qualifier:

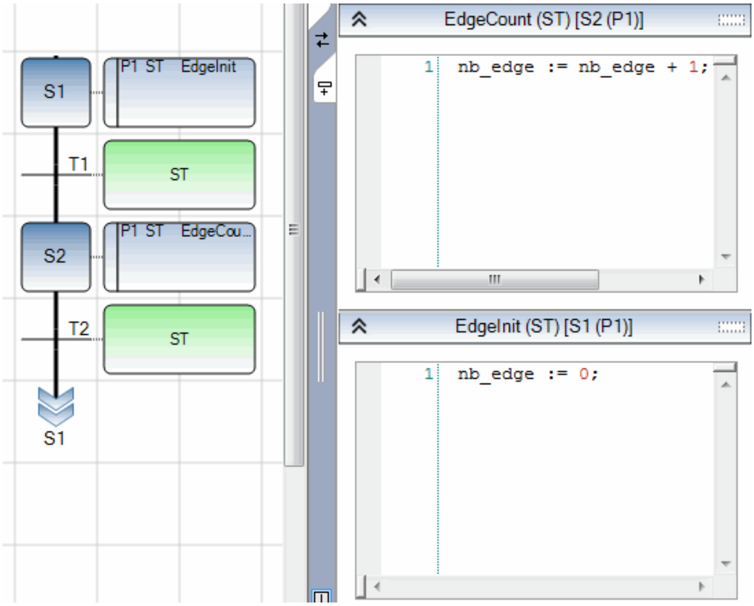
Step Activity

Execution



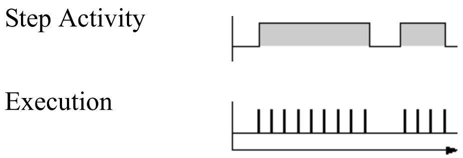
### Example

In the following SFC program, step S1 is assigned an ST action named EdgeInit having the P1 qualifier and S2 is assigned an ST action named EdgeCount having the P1 qualifier. The code for these actions is programmed in their respective level 2 window.


S1
P1 ST EdgeInit
T1
ST
S2
P1 ST EdgeCou...
T2
ST
S1
EdgeCount (ST) [S2 (P1)]
1 nb_edge := nb_edge + 1;
EdgeInit (ST) [S1 (P1)]
1 nb_edge := 0;

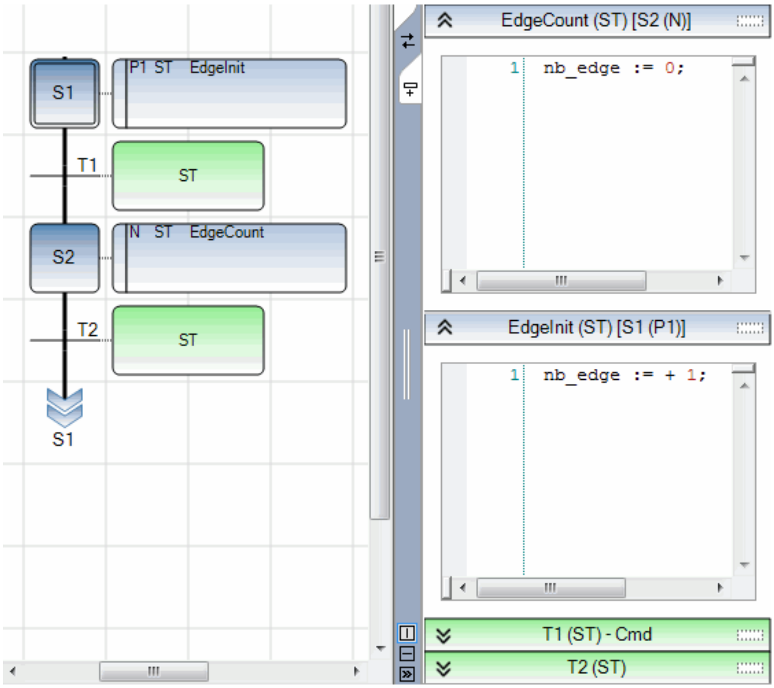
## Non-Stored Actions

A non-stored (normal) action is a list of ST or LD instructions which are executed at each cycle during the whole active period of the step. Instructions are written according to the language syntax in use. Non-stored actions have the "N" qualifier. The following is the results of a non-stored action:




### Example

In the following program, step S1 is assigned an ST action named EdgeInit having the P1 qualifier and S2 is assigned an ST action named EdgeCount having the N qualifier. The code for these actions is programmed in their respective level 2 window.



S1
P1 ST EdgeInit
T1
ST
S2
N ST EdgeCount
T2
ST
S1
EdgeCount (ST) [S2 (N)]
1 nb_edge := 0;
EdgeInit (ST) [S1 (P1)]
1 nb_edge := + 1;
T1 (ST) - Cmd
T2 (ST)

## SFC Actions

An SFC action is a child SFC sequence, started or terminated according to the change of the step activity signal. An SFC action can have the N (Non stored), S (Set), and R (Reset) qualifiers. This is the meaning of the action on an SFC child:

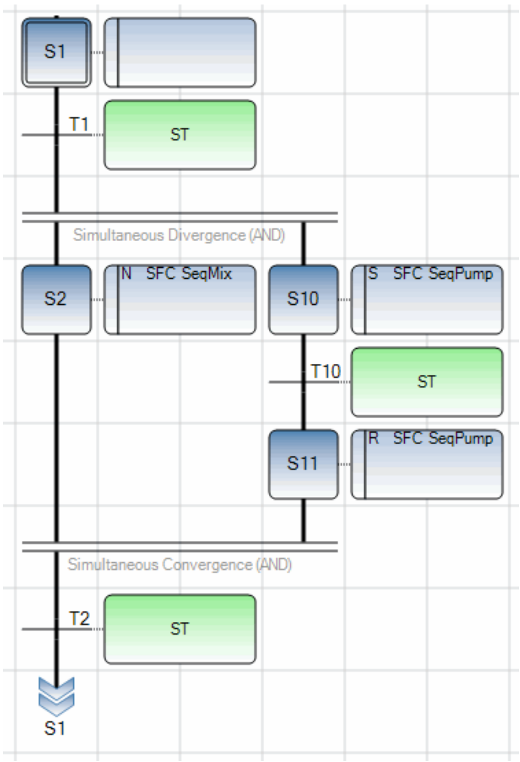
| | |
|---|---|
| **N on a child** | starts the child sequence when the step becomes active and terminates the child sequence when the step becomes inactive |
| **S on a child** | starts the child sequence when the step becomes active |
| **R on a child** | stops the child sequence when the step becomes active |

The SFC sequence specified as an action must be a child SFC program of the program currently being edited.

### Example

(* SFC Program using SFC Action *)

The main SFC program is named Parent having one SFC child, called SeqMlx. The SFC programming of the parent SFC program is the following:


S1
T1
ST
Simultaneous Divergence (AND)
S2
N SFC SeqMix
S10
S SFC SeqPump
T10
ST
S11
R SFC SeqPump
Simultaneous Convergence (AND)
T2
ST
S1

## Coding Conditions for Transitions

You code conditions for the clearing of transitions by programming these in the level 2 window. When defining the properties of conditions, you indicate a name, a comment (optional), and the programming language (type). The available programming languages for transitions are LD and ST.

When no expression is attached to the Transition, the default condition is TRUE.

**To code conditions for transitions**

**1.** Select the transition for which to code a condition.

**2.** Right-click the transition, and then click **Properties**.

**3.** Specify the required properties for the transition from the Properties window.

   **a)** To rename the transition, type the required text in the Name field.

   **b)** To specify the type (programming language) for the transition condition, choose the required type in the Type field.

   **c)** To include a comment, type the required text in the Comment field.

**4.** In the Level 2 window, program the required condition.

## Conditions Programmed in ST

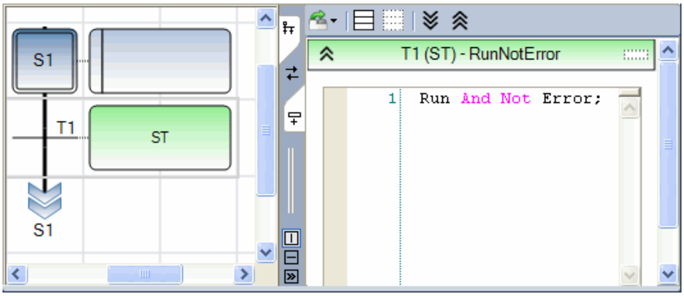
The ST language can be used to describe the condition for a transition. The complete expression must have Boolean type and may be terminated by a semi colon, according to the following syntax:

< boolean_expression > ;

The expression may be a TRUE or FALSE constant expression, a single input or an internal Boolean variable, or a combination of variables that leads to a Boolean value.

### Example

(* SFC Program with ST programming for Transitions *)

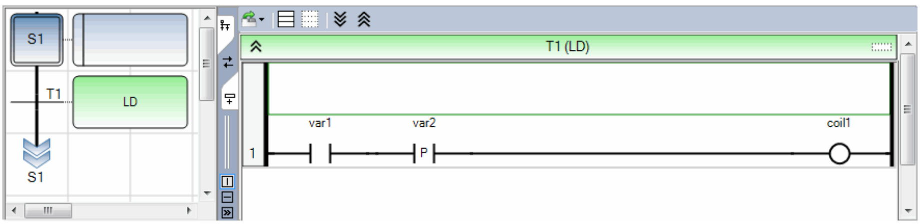


## Conditions Programmed in LD

The Ladder Diagram (LD) language can be used to describe the condition attached to a transition. The initial diagram is composed of a rung.

### Example

(* SFC Program with LD programming for transitions *)



# Calling Functions from Transitions

Any function (written in ST, LD, or FBD), or a "C" function can be called to evaluate the condition attached to a transition, according to the following syntax in ST:

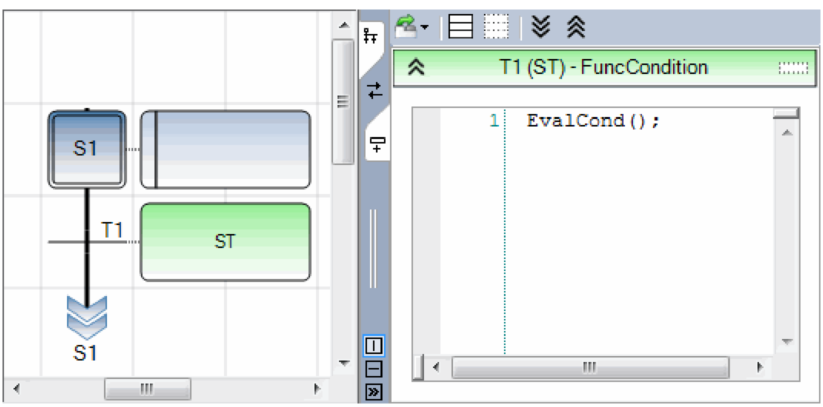
< function > ( ) ;

The value returned by the function must be Boolean and yield the resulting condition:

| | | |
|---|---|---|
| return value = **FALSE** | -> | condition is **FALSE** |
| return value = **TRUE** | -> | condition is **TRUE** |

### Example

(* SFC program with function call for transitions *)



## Calling Function Blocks from Transitions

It is not recommended to call a function block in an SFC condition for the following reasons:

- A function block should be called at each cycle, typically in a cyclic program.
- An SFC condition is evaluated only when all of its preceding steps are active (not at each cycle)

## SFC Keyboard Shortcuts

The following keyboard shortcuts are available for use with the SFC language. Some shortcuts do not apply or may differ while debugging.

| | |
|---|---|
| Ctrl+A | Selects all elements (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while debugging) |
| Ctrl+V | Pastes elements saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl+X | Cuts the selected elements to the clipboard (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Ctrl+S | Saves the selected elements (not available while debugging) |
| Ctrl+Shift+S | Saves all files making up a solution (not available while debugging) |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+0 | Inserts an initial step (not available while debugging) |
| Ctrl+1 | Inserts a step (not available while debugging) |
| Ctrl+2 | Inserts a transition (not available while debugging) |
| Ctrl+3 | Inserts a sequence control (not available while debugging) |
| Ctrl+4 | Inserts a jump (not available while debugging) |
| Ctrl+Shift+R | Renumbers the steps and transitions using the default naming convention (S*n* and T*n*) |
| Ctrl+Page Up | Jumps to the top edge of the visible language container |
| Ctrl+Page Down | Jumps to the bottom edge of the visible language container |
| Alt+Up Arrow | Scrolls up |
| Alt+Down Arrow | Scrolls down |
| Alt+Left Arrow | Scrolls left |
| Alt+Right Arrow | Scrolls right |
| Ctrl+Up Arrow | Slowly scrolls up |

| | |
|---|---|
| Ctrl+Down Arrow | Slowly scrolls down |
| Ctrl+Left Arrow | Slowly scrolls left |
| Ctrl+Right Arrow | Slowly scrolls right |
| Up Arrow | Moves up the grid or from one selected element to the next |
| Down Arrow | Moves down the grid or from one selected element to the next |
| Left Arrow | Moves to the left across the grid or from one selected element to the next |
| Right Arrow | Moves to the right across the grid or from one selected element to the next |
| Delete | Removes the selected elements (not available while debugging) |

# SAMA Language

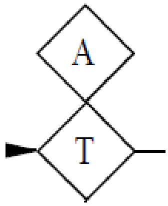
You can development Scientific Apparatus Makers Association (SAMA) diagrams using the IEC 61131-3 Function Block Diagram (FBD) language. You can build, edit, simulate and debug SAMA diagrams.

**See Also**

Debugging SAMA Programs

# SAMA Diagram Main Format

SAMA language diagrams are composed of symbols linked together using the continuously variable signal type to define complex programs. SAMA diagrams are built using the FBD editor and follow the IEC 61131-3 standard. The combination of symbols or auxiliary operations is not supported by the IEC 61131-3 standard. Therefore, the following example is not supported:




In SAMA diagrams, functions are placed from right to left as follows:

- Measuring functions are placed on the left
- Signal processing and manual functions are placed in the center
- Final control functions are placed at the right

In SAMA POUs, the main signal enters each symbol enclosure from the left (input) and exits from the right (output). For auxiliary functions, the signal enters the symbol enclosure from either the top or bottom.

You connect the logical points of a diagram using connection lines. When connecting elements, arrows indicate signal direction. From the connection properties, you can choose to modify the line style and line type. You can also choose whether arrows are displayed.

When programming a SAMA POU, the Toolbox offers the available SAMA Elements.

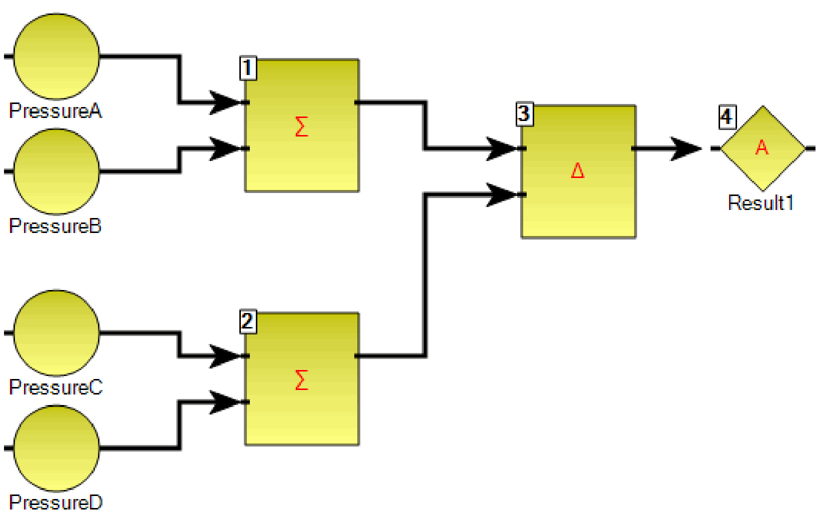
**See Also**
Execution Order of SAMA Programs

# Execution Order of SAMA Programs

SAMA programs are executed horizontally, from left to right and then from the top downward. For the execution order of a program, a block is any element in the diagram, a network is a group of blocks linked together. The position of a block is based on its top-left corner. The following rules apply to the execution order of the program:

- Networks are executed from left to right and top to bottom only.
- All inputs must be resolved before executing the block. When the inputs of two or more blocks are resolved at the same time, the decision for the execution is based on the position of the block (left to right, then top to bottom).
- The outputs of a block are executed recursively from left to right, then from top to bottom.

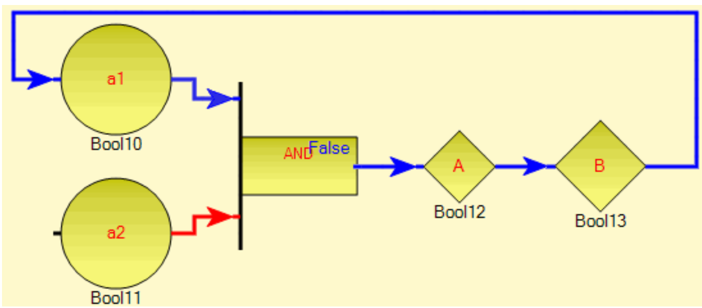
The following is an example of SAMA diagram in horizontal format:



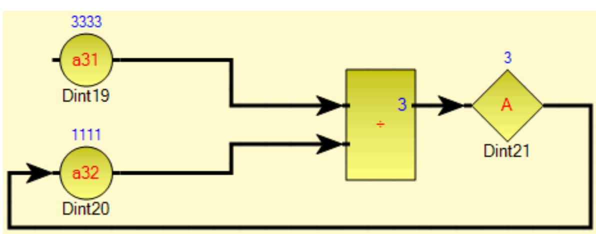
# Debugging SAMA Programs

When debugging SAMA programs, you can monitor the output values of elements. These values are displayed using color, numeric, or textual values according to their data type:

- Output values of boolean type are displayed using color. The output value color continues to the next input. When the output value is unavailable, boolean elements remain black. The colors are red when True and blue when False.




- Output values of SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, and STRING type are displayed as a numeric or textual value in the element. When the output is a structure type, the displayed value is the selected member.




When the output value for a numeric or textual value is unavailable, the *WAIT* text is displayed in the output label. Values are also displayed in the corresponding dictionary instance.

### See Also

SAMA Diagram Main Format

# SAMA Elements

When programming a SAMA POU using the FBD editor, you can drag SAMA elements and FBD elements into FBD language containers. You can also use Ladder (LD) elements in FBD containers.

- Alarm Signal
- Averaging
- Bias
- Derivative
- Difference
- Dividing
- Equal To
- Exponential
- Greater Than
- High Selecting
- Integral
- IPID
- Lesser Than
- Logical AND
- Logical OR
- Logical Signal
- Low Selecting
- Measuring or Readout
- Multiplying
- NOT
- Root Extraction
- SAMA Variable
- Server Monitored Variable
- Signal Monitor
- Summing
- Transfer
- Variable Signal Generator

**See Also**

SAMA Diagram Main Format

Execution Order of SAMA Programs

## Alarm Signal

SAMA Representation: FBD Representation:


Description:

The alarm signal is a variable signal generator representing alarms. The alarm signal is only a graphical representation and is still considered a standard ISaGRAF variable.

**To insert an Alarm Signal element**

1. From the Toolbox, drag the **Alarm Signal** element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**

The Alarm Signal variable is displayed in the language container in SAMA format.

**To insert an FBD variable**

1. From the Toolbox, drag the **variable** element into the language container.

   The Variable Selector is displayed.

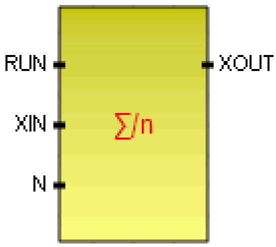
2. In the Variable Selector, select the required variable, then click **OK.**

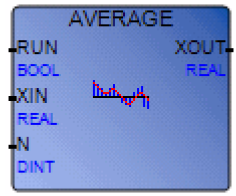
The variable is displayed in the language container in FBD format.

### See Also
FBD Variables

# Averaging

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| RUN | BOOL | TRUE=run / FALSE=reset |
| XIN | REAL | Any REAL variable |
| N | DINT | Application defined number of samples |
| XOUT | REAL | Running average of XIN value |

Description:

The output value is the algebraic sum of the input values divided by the number of inputs.

The Averaging element is mapped to the IEC 61131-3 AVERAGE function block.

### To insert an Averaging element

- From the Toolbox, drag the **Averaging** element into the language container.

The Averaging element is displayed in the language container in SAMA format.

### To insert an AVERAGE function block

**1.** From the Toolbox, drag the block element into the language container.

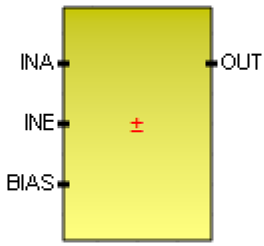
The Block Selector is displayed.

**2.** In the Block Selector, select **AVERAGE,** then click **OK**.

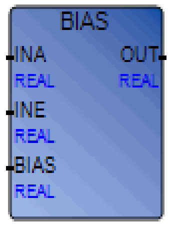
The Averaging element is displayed in the language container in SAMA format.

## Bias

SAMA Representation:




FBD Representation:




Arguments:

| | | |
|---|---|---|
| INA | REAL | Input signal A |
| INE | REAL | Input signal E |
| BIAS | REAL | Bias value |
| OUT | REAL | Output value. Output = (BIAS) + InputA + InputE |

Description:

The output value is equal to the input value plus or minus the bias value.

The Bias element is mapped to the BIAS function block.

**To insert a Bias element**

- From the Toolbox, drag the **Bias** element into the language container.

The Bias element is displayed in the language container in SAMA format.

**To insert a BIAS function block**

1. From the Toolbox, drag the block element into the language container.

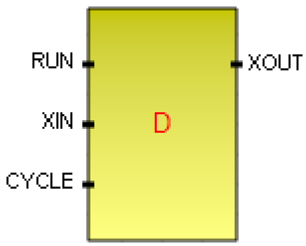
   The Block Selector is displayed.

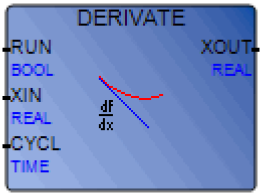
**2.** In the Block Selector, select **BIAS**, then click **OK**.

The Bias element is displayed in the language container in SAMA format.

## Derivative

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=normal / FALSE=reset |
| XIN | REAL | Input: any real value |
| CYCLE | TIME | Sampling period. Possible values range from 0ms to 23h59m59s999ms. |
| XOUT | REAL | Differentiated output |

Description:

The output value is proportional to the rate of change of the input value.

The Derivative element is mapped to the IEC 61131-3 DERIVATE function block.

**To insert a Derivative element**

- From the Toolbox, drag the **Derivative** element into the language container.

The Derivative element is displayed in the language container in SAMA format.

**To insert a DERIVATE function block**

1. From the Toolbox, drag the block element into the language container.

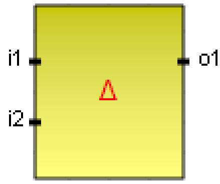
   The Block Selector is displayed.

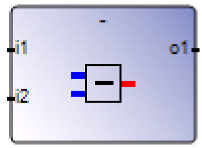
2. In the Block Selector, select **DERIVATE**, then click **OK**.

The Derivative element is displayed in the language container in SAMA format.

## Difference

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| i1 | DINT | can be any DINT |
| i2 | DINT | can be any DINT |
| o1 | DINT | subtraction (first minus second) |

Description:

The output value is the algebraic difference between the input values.

The Difference element is mapped to the IEC 61131-3 Subtraction operator.

**To insert a Difference element**

- From the Toolbox, drag the **Difference** element into the language container.

The Difference element is displayed in the language container in SAMA format.

**To insert a Subtraction operator**

1. From the Toolbox, drag the block element into the language container.

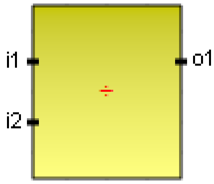
   The Block Selector is displayed.

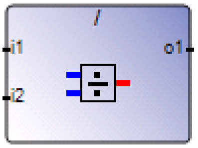
2. In the Block Selector, select **Subtraction (-)**, then click **OK**.

The Difference element is displayed in the language container in SAMA format.

## Dividing

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| i1 | DINT | can be a DINT (operand) |
| i2 | DINT | can be a DINT (divisor) |
| o1 | DINT | division of i1 by i2 |

Description:

The output value is proportional to the quotient of the input values.

The Dividing element is mapped to the IEC 61131-3 Division operator.

**To insert a Dividing element**

- From the Toolbox, drag the **Dividing** element into the language container.

The Dividing element is displayed in the language container in SAMA format.

**To insert a Division operator**

1. From the Toolbox, drag the block element into the language container.

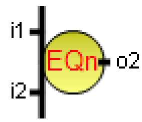
   The Block Selector is displayed.

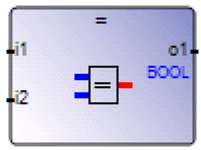
2. In the Block Selector, select **Division (/)**, then click **OK**.

The Dividing element is displayed in the language container in SAMA format

# Equal To

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| i1 | DINT | Both inputs must have the same DINT type |
| i2 | DINT | |
| o1 | BOOL | TRUE if i1 = i2 |

Description:

Compares the first input to the second to determine equality.

The Equal To element is mapped to the IEC 61131-3 Equal operator.

**To insert an Equal To element**

- From the Toolbox, drag the **Equal To** element into the language container.

The Equal To element is displayed in the language container in SAMA format.

**To insert an Equal operator**

1. From the Toolbox, drag the block element into the language container.

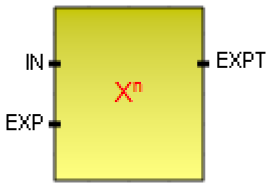
   The Block Selector is displayed.

2. In the Block Selector, select **Equal (=)**, then click **OK**.

The Equal To element is displayed in the language container in SAMA format.

# Exponential

SAMA Representation: FBD Representation:





Arguments:

| | | |
|---|---|---|
| IN | REAL | Any signed real value |
| EXP | DINT | Integer exponent |
| EXPT | REAL | $(IN^{EXP})$ |

Description:

The output value is the input value raised to a defined power.

The Exponential element is mapped to the IEC 61131-3 EXPT function.

**To insert an Exponential element**

- From the Toolbox, drag the **Exponential** element into the language container.

The Exponential element is displayed in the language container in SAMA format.

**To insert an EXPT function**

1. From the Toolbox, drag the block element into the language container.

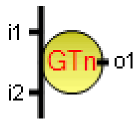
   The Block Selector is displayed.

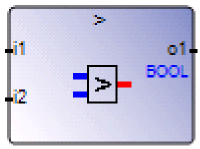
2. In the Block Selector, select **EXPT**, then click **OK**.

The Exponential element is displayed in the language container in SAMA format.

# Greater Than

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| i1 | DINT | Both inputs must have DINT type |
| i2 | DINT | |
| o1 | BOOL | TRUE if i1 > i2 |

Description:

Compares input variables to determine whether the first is greater than the second.

The Greater Than element is mapped to the IEC 61131-3 Greater Than operator.

**To insert a Greater Than element**

- From the Toolbox, drag the **Greater Than** element into the language container.

The Greater Than element is displayed in the language container in SAMA format.

**To insert a Greater Than operator**

**1.** From the Toolbox, drag the block element into the language container.

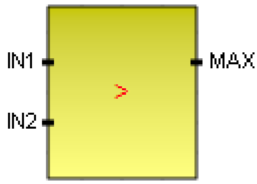
The Block Selector is displayed.

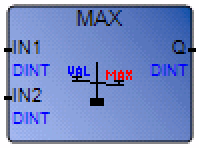
**2.** From the Block Selector, select **Greater Than (>)**, then click **OK**.

The Greater Than element is displayed in the language container in SAMA format.

# High Selecting

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| IN1 | DINT | Any signed integer value |
| IN2 | DINT | (cannot be REAL) |
| MAX | DINT | Maximum of both input values |

Description:

The output value is equal to the largest input value.

The High Selecting element is mapped to the IEC 61131-3 MAX function.

**To insert a High Selecting element**

- From the Toolbox, drag the **High Selection** element into the language container.

The High Selecting element is displayed in the language container in SAMA format.

**To insert a MAX function**

**1.** From the Toolbox, drag the block element into the language container.

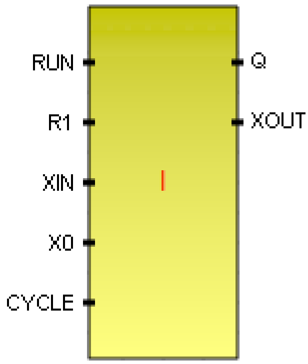
The Block Selector is displayed.

**2.** In the Block Selector, select **MAX**, then click **OK**.

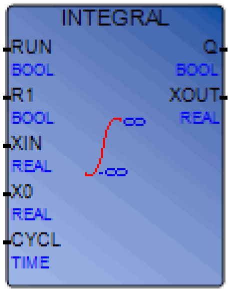
The High Selecting element is displayed in the language container in SAMA format.

# Integral

SAMA Representation:




FBD Representation:




Arguments:

| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=integrate / FALSE=hold |
| R1 | BOOL | Overriding reset |
| XIN | REAL | Input: any REAL value |
| X0 | REAL | Initial value |
| CYCLE | TIME | Sampling period. Possible values range from 0ms to 23h59m59s999ms. |
| Q | BOOL | Not R1 |
| XOUT | REAL | Integrated output |

Description:

The output value varies according to the magnitude and duration of the input value. The output value is proportional to the time integral of the input value.

The Integral element is mapped to the IEC 61131-3 INTEGRAL function block.

**To insert an Integral element**

- From the Toolbox, drag the **Integral** element into the language container.

The Integral element is displayed in the language container in SAMA format.

**To insert an INTEGRAL function block**

1. From the Toolbox, drag the block element into the language container.

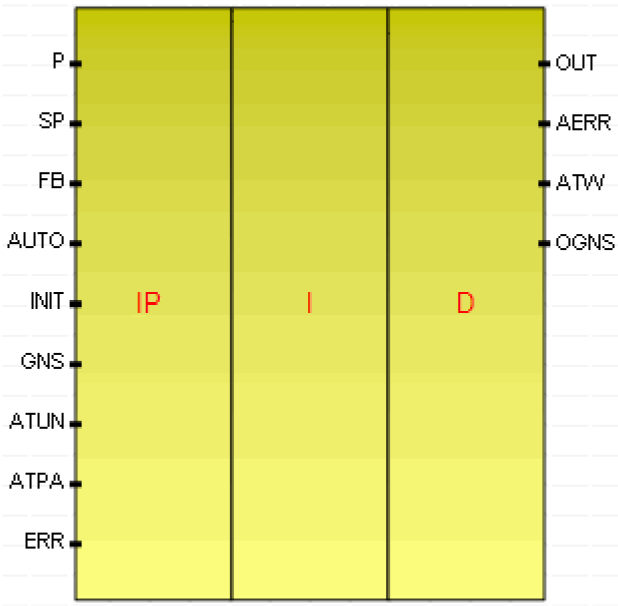
   The Block Selector is displayed.

2. In the Block Selector, select **INTEGRAL**, then click **OK**.

The Integral element is displayed in the language container in SAMA format.

# IPID

SAMA Representation:




FBD Representation


Arguments:

| | | | |
|---|---|---|---|
| Process | P | REAL | Process value |
| SetPoint | SP | REAL | Set point |
| Feedback | FB | REAL | Feed Back signal |
| Auto | AUTO | BOOL | The operation mode of the PID controller:<br>TRUE controller runs in normal mode<br>FALSE controller causes reset R to track (F-GE) |
| Initialize | INIT | BOOL | A change in value (TRUE to FALSE or FALSE to TRUE) causes the controller to eliminate any proportional gain during that cycle. Also initializes AutoTune sequences. |

| | | | |
|---|---|---|---|
| Gains | GNS | GAIN_PID | Gains PID for IPIDCONTROLLER (see GAIN_PID structure) |
| AutoTune | ATUN | BOOL | When set to TRUE and Auto and Initialize are FALSE, the AutoTune sequence is started |
| ATParameters | ATPA | GAIN_PID | Auto Tune Parameters (see AT_Param structure) |
| ErrorMode | ERR | DINT | Mode used to handle errors. Possible values are:<br>0 no error messages ErrLog file<br>1 prints error messages level 1 in ErrLog file<br>2 prints error messages level 1 and level 2 in ErrLog file |
| Output | OUT | REAL | Output value from controller |
| AbsoluteError | AERR | REAL | Absolute Error (Process – SETPOINT) from controller |
| ATWarning | ATW | DINT | Warning for Auto Tune sequence. Possible values are:<br>0 no auto tune done<br>1 in auto tune mode<br>2 auto tune done<br>-1 ERROR 1 input Auto set to TRUE, no auto tune possible<br>-2 ERROR 2 auto tune error, ATDynaSet expired |
| OutGains | OGNS | GAIN_PID | Gains calculated after AutoTune sequences (see GAIN_PID structure) |

GAIN_PID structure:

| | | |
|---|---|---|
| DirectActing | BOOL | The type of acting:<br>TRUE direct acting<br>FALSE reverse acting |
| ProportionalGain | REAL | Proportional gain for PID (>= 0.0001) |
| TimeIntegral | REAL | Time integral value for PID (>= 0.0001) |
| TimeDerivative | REAL | Time derivative value for PID (> 0.0) |
| DerivativeGain | REAL | Derivative gain for PID (> 0.0) |

AT_Param structure:

| | | |
|---|---|---|
| Load | REAL | Load parameter for auto tuning. This is the output value when starting AutoTune. |
| Deviation | REAL | Deviation for auto tuning. This is the standard deviation used to evaluate the noise band needed for AutoTune. |
| Step | REAL | Step value for AutoTune. Must be greater than noise band and less than ½ Load. |
| ATDynamSet | REAL | Waiting time before abandoning auto tune |
| ATReset | BOOL | The indication of whether the Output value is reset to zero after an AutoTune sequence:<br>TRUE resets Output to zero<br>FALSE leaves Output at Load value |

Description:

The IPID element is mapped to the IPIDCONTROLLER function block.

The IPID controller (IPIDCONTROLLER) is based on the following function block:

with A: Acting (+/- 1)

PG: Proportional Gain

DG: Derivative Gain

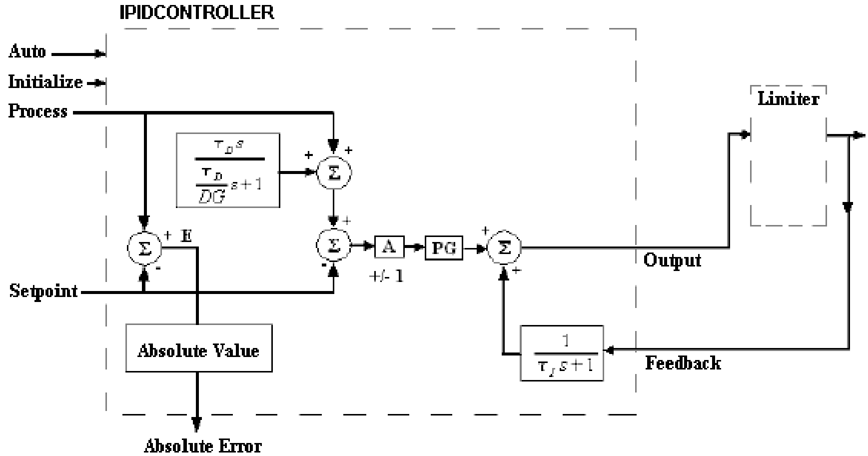
ã$_D$: Time Derivative

ã$_I$: Time Integral

In the HMI, the IPID faceplate is available for use with the IPIDCONTROLLER function block.

The IPID element enables tracking When Input Auto is Logic One (TRUE), the IPID runs in normal auto mode. When Input Auto is Logic Zero (FALSE), this causes reset R to track (F-GE). This forces the IPID Output to track the Feedback within the IPID limits and allows the controller to switch back to auto without bumping the Output.

When Input Auto is Logic One (TRUE), the IPID element runs in normal auto mode. When Input Auto is Logic Zero (FALSE), this causes reset R to track (F-GE). This forces the IPID Output to track the Feedback within the IPID element limits and allows the controller to switch back to auto without bumping the Output.




For Input Initialize, changing from Logic Zero (FALSE) to Logic One (TRUE) or Logic One (TRUE) to Logic Zero (FALSE) when AutoTune is Logic Zero (FALSE) causes the IPID element to eliminate any proportional gain action during that cycle (i.e Initialize). This can be used to prevent bumping the Output when changes are made to the SETPOINT using a switch function block.

To run an AutoTune sequence, the input ATParameters must be completed. The input Gain and DirectActing must be set according to the process and DerivativeGain set, typically, to 0.1. The AutoTune sequence is started with this sequence:

- Put input Initialize to (Logic One) TRUE
- Put input Autotune to (Logic One) TRUE
- Put back Initialize to (Logic Zero) FALSE
- Wait output ATWarning going to 2
- Transfer values for output OutGains to input Gains

To finalize the tuning, some fine tuning may be needed depending on the processes and needs. When setting TimeDerivative to 0.0, the IPID element forces DerivativeGain to 1.0 then works as a PI controller.

**To insert an IPID element**

- From the Toolbox, drag the **IPID** element into the language container.

The IPID element is displayed in the language container.

**To insert an IPIDCONTROLLER function block**

1. From the Toolbox, drag the block element into the language container.

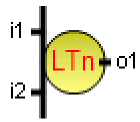
   The Block Selector is displayed.

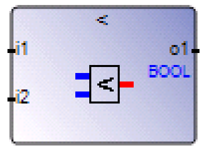
2. In the Block Selector, select **IPIDCONTROLLER**, then click **OK**.

The IPID element is displayed in the language container.

# Lesser Than

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| i1 | DINT | Both inputs must have the DINT type |
| i2 | DINT | |
| o1 | BOOL | TRUE if i1i2 < i2 |

Description:

Compares input variables to determine whether the first is less than the second.

The Lesser Than element is mapped to the IEC 61131-3 Less Than operator.

**To insert a Lesser Than element**

- From the Toolbox, drag the **Lesser Than** element into the language container.

The Lesser Than element is displayed in the language container in SAMA format.

**To insert a Less Than operator**

**1.** From the Toolbox, drag the block element into the language container.

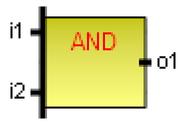
The Block Selector is displayed.

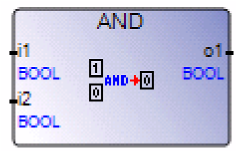
- In the Block Selector, select **Less Than (<)**, then click **OK**.

The Lesser Than element is displayed in the language container in SAMA format.

## Logical AND

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| i1 | BOOL | |
| i2 | BOOL | |
| o1 | BOOL | Boolean AND of the input terms |

Description:

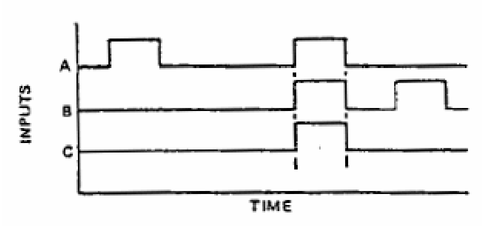
The output is a Logic One only if all of the input signals are Logic Ones.

The Logical AND element is mapped to the IEC 61131-3 AND operator.

Mathematical equation:

| A | B | C |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

Graphic representation:




**To insert a Logical AND element**

- From the Toolbox, drag the **Logical AND** element into the language container.

The Logical AND element is displayed in the language container in SAMA format.

**To insert an AND operator**

1. From the Toolbox, drag the block element into the language container.

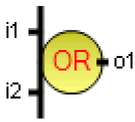
   The Block Selector is displayed.

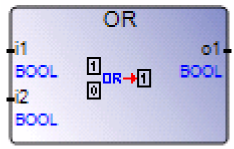
2. In the Block Selector, select **AND**, then click **OK**.

The Logical AND element is displayed in the language container in SAMA format.

## Logical OR

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| i1 | BOOL | |
| i2 | BOOL | |
| o1 | BOOL | Boolean **OR** of the input terms |

Description:

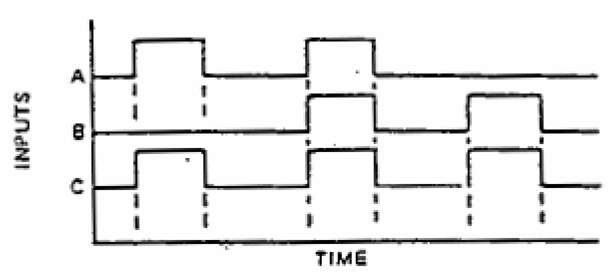
When there is one or more Logic One inputs, the output of Logical OR is Logic One.

The Logical OR element is mapped to the IEC 61131-3 OR operator.

Mathematical equation:

| A | B | C |
|---|---|---|
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |
| 0 | 0 | 0 |

Graphic representation:




**To insert a Logical OR element**

- From the Toolbox, drag the **Logical OR** element into the language container.

The Logical OR element is displayed in the language container in SAMA format.

**To insert an OR operator**

1. From the Toolbox, drag the block element into the language container.

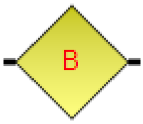
   The Block Selector is displayed.

2. In the Block Selector, select **OR**, then click **OK**.

The Logical OR element is displayed in the language container in SAMA format.

## Logical Signal

SAMA Representation: FBD Representation:




Description:

The logical signal generates logical signals for manual processing.

The Logical Signal element is mapped to the IEC 61131-3 variable element.

**To insert a Logical Signal element**

You can also insert variables using the variable element available in the FBD toolbox.

1. From the Toolbox, drag the **Logical Signal** element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK**.

The Logical Signal element is displayed in the language container in SAMA format.

**To insert an FBD variable**

1. From the Toolbox, drag the **variable** element into the language container.

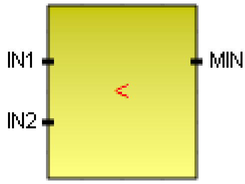
   The Variable Selector is displayed.

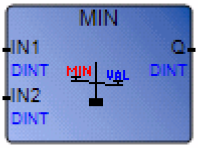
2. In the Variable Selector, select the required variable, then click **OK.**

The variable is displayed in the language container in FBD format.

## Low Selecting

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| IN1 | DINT | Any signed integer value |
| IN2 | DINT | (cannot be REAL) |
| MIN | DINT | Minimum of both input values |

Description:

The output value is equal to the smallest input value.

The Low Selecting element is mapped to the IEC 61131-3 MIN function.

**To insert a Low Selecting element**

- From the Toolbox, drag the **Low Selecting** element into the language container.

The Low Selecting element is displayed in the language container in SAMA format.

**To insert a MIN function**

1. From the Toolbox, drag the block element into the language container.

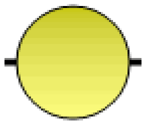
   The Block Selector is displayed.

2. In the Block Selector, select **MIN**, then click **OK**.

The Low Selecting element is displayed in the language container in SAMA format.

## Measuring or Readout

SAMA Representation: FBD Representation:



Description:

A literal value or a defined word. On the SAMA representation, you can add text on the element.

Measuring or Readout element is mapped to the IEC 61131-3 variable element.

**To insert a Measuring or Readout element**

You can also insert variables using the variable element available in the FBD toolbox.

1. From the Toolbox, drag the **Measuring or Readout** element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**
3. To add text to the element, click the element then type the required text.

The Measuring or Readout element is displayed in the language container in SAMA format.

**To insert an FBD variable**

1. From the Toolbox, drag the **variable** element into the language container.

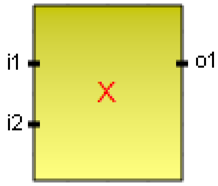
   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**

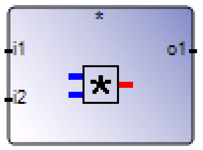
The variable is displayed in the language container in FBD format.

## Multiplying

SAMA Representation:




FBD Representation:




Arguments:

| | | |
|---|---|---|
| i1 | DINT | can be a DINT |
| i2 | DINT | can be a DINT |
| o1 | DINT | **multiplication** of the input terms |

Description:

The output value is proportional to the product of the input values.

The Multiplying element is mapped to the IEC 61131-3 Multiplication operator.

**To insert a Multiplying element**

- From the Toolbox, drag the **Multiplying** element into the language container.

The Multiplying element is displayed in the language container in SAMA format.

**To insert a Multiplication operator**

**1.** From the Toolbox, drag the block element into the language container.

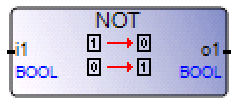
The Block Selector is displayed.

**2.** In the Block Selector, select **Multiplication (*)**, then click **OK**.

The Multiplying element is displayed in the language container in SAMA format.

# NOT

SAMA Representation: FBD Representation:


Arguments:

| | | |
|---|---|---|
| i1 | BOOL | Any Boolean variable |
| o1 | BOOL | TRUE when i1 is FALSE<br>FALSE when i1 is TRUE |

Description:

For Boolean expressions, converts variables to negated variables.

The NOT element is mapped to the IEC 61131-3 NOT operator.

### To insert a NOT element

- From the Toolbox, drag the **NOT** element into the language container.

The NOT element is displayed in the language container in SAMA format.

### To insert a NOT operator

**1.** From the Toolbox, drag the block element into the language container.

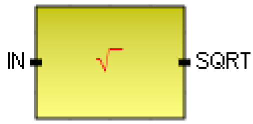
The Block Selector is displayed.

**2.** In the Block Selector, select **NOT**, then click **OK**.

The NOT element is displayed in the language container in SAMA format.

## Root Extraction

SAMA Representation: FBD Representation:





Arguments:

| | | |
|---|---|---|
| IN | REAL | Must be greater than or equal to zero |
| SQRT | REAL | Square root of the input value |

Description:

The output value is equal to the root of the input value.

The Root Extraction element is mapped to the IEC 61131-3 SQRT function.

**To insert a Root Extraction element**

- From the Toolbox, drag the **Root Extraction** element into the language container.

The Root Extraction element is displayed in the language container in SAMA format.

**To insert a SQRT function**

1. From the Toolbox, drag the block element into the language container.

   The Block Selector is displayed.

2. In the Block Selector, select **SQRT**, then click **OK**.

The Root Extraction element is displayed in the language container in SAMA format.

## SAMA Variable

SAMA Representation: FBD Representation:

Description:

The SAMA variable is a standard ISaGRAF variable representing elementary data used in SAMA programs.

**To insert a SAMA Variable element**

1. From the Toolbox, drag the **SAMA Variable** element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**

The SAMA variable is displayed in the language container in SAMA format.

**To insert an FBD variable**

1. From the Toolbox, drag the **variable** element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**

The variable is displayed in the language container in FBD format.

### See Also

FBD Variables

## Server Monitored Variable

SAMA Representation: FBD Representation:


Description:

The server monitored variable monitors logical values. The server monitored variable is only a graphical representation and is still considered a standard ISaGRAF variable.

**To insert a Server Monitored Variable element**

1. From the Toolbox, drag the **Server Monitored Variable** element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**

The Server Monitored Variable is displayed in the language container in SAMA format.

**To insert an FBD variable**

1. From the Toolbox, drag the **variable** element into the language container.

   The Variable Selector is displayed.

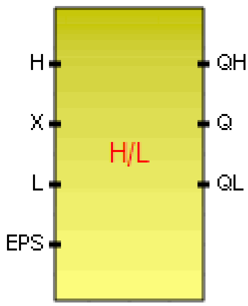
2. In the Variable Selector, select the required variable, then click **OK.**

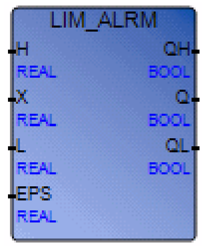
The variable is displayed in the language container in FBD format.

### See Also
FBD Variables

## Signal Monitor

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| H | REAL | High limit value |
| X | REAL | Input: any real value |
| L | REAL | Low limit value |
| EPS | REAL | Hysteresis value (must be greater than zero) |
| QH | BOOL | "high" alarm: TRUE if X above high limit H |
| Q | BOOL | Alarm output: TRUE if X out of limits |
| QL | BOOL | "low" alarm: TRUE if X below low limit L |

Description:

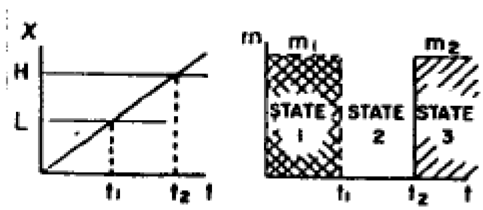
The output value has discrete states that depend on the value of the input. When the input exceeds (or becomes less than) the limit value, the output changes state. Each of these limit values may have deadband.

The Signal Monitor element is mapped to the IEC 61131-3 LIM_ALRM function block.

Mathematical equations:

| | |
|---|---|
| State 1<br>(First output $m_1$ is energized or in alarm state) | $x < L$ |
| State 2<br>(Both outputs are inactive or de-energized) | $L \leq x \leq H$ |
| State 3<br>(Second output $m_2$ is energized or in alarm state) | $x > H$ |

Graphic representation:




**To insert a Signal Monitor element**

- From the Toolbox, drag the **Signal Monitor** element into the language container.

The Signal Monitor element is displayed in the language container in SAMA format.

**To insert a LIM_ALRM function block**

**1.** From the Toolbox, drag the block element into the language container.

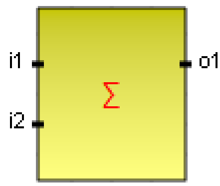
The Block Selector is displayed.

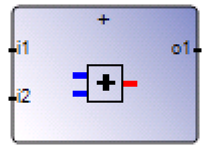
- In the Block Selector, select **LIM_ALRM**, then click **OK**.

The Signal Monitor element is displayed in the language container in SAMA format.

## Summing

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| i1 | DINT | can be of any DINT |
| i2 | DINT | can be of any DINT |
| o1 | DINT | **addition** of the input terms |

Description:

The output value is the algebraic sum of the input values.

The Summing element is mapped to the IEC 61131-3 Addition operator.

**To insert a Summing element**

- From the Toolbox, drag the **Summing** element into the language container.

The Summing element is displayed in the language container in SAMA format.

**To insert an Addition operator**

1. From the Toolbox, drag the block element into the language container.

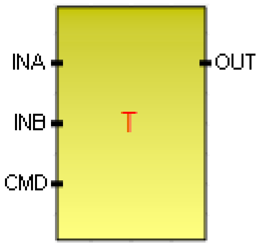
   The Block Selector is displayed.

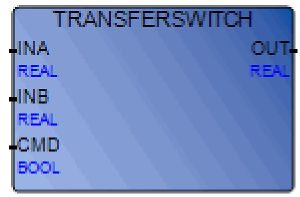
2. In the Block Selector, select **Addition (+),** then click **OK**.

The Summing element is displayed in the language container in SAMA format.

## Transfer

SAMA Representation: FBD Representation:




Arguments:

| | | | |
|---|---|---|---|
| InputA | INA | REAL | Input signal A |
| InputB | INB | REAL | Input signal B |
| Command | CMD | BOOL | (Command) Indication of which signal to select:<br>FALSE selects InputA<br>TRUE selects InputB |
| Output | OUT | REAL | Output signal |

Description:

The output value is equal to the input selected by the transfer and is either *on* or *off*. The transfer state is determined by external means.

The Transfer element is mapped to the TRANSFERSWITCH function block.

**To insert a Transfer element**

- From the Toolbox, drag the **Transfer** element into the language container.

The Transfer element is displayed in the language container in SAMA format.

**To insert a TRANSFERSWITCH function block**

1. From the Toolbox, drag the block element into the language container.

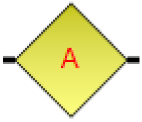
   The Block Selector is displayed.

2. In the Block Selector, select **TRANSFERSWITCH**, then click **OK**.

The Transfer element is displayed in the language container in SAMA format.

## Variable Signal Generator

SAMA Representation: FBD Representation:




Description:

The output value is an analog signal from the generator.

The Variable Signal Generator element is mapped to the IEC 61131-3 variable element.

**To insert a Variable Signal Generator element**

You can also insert variables using the variable element.

1. From the Toolbox, drag the **Variable Signal Generator** element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**

The Variable Signal Generator element is displayed in the language container in SAMA format.

**To insert an FBD variable**

1. From the Toolbox, drag the **variable** element into the language container.

   The Variable Selector is displayed.

2. In the Variable Selector, select the required variable, then click **OK.**

The variable is displayed in the language container in FBD format.

# Mapping Chart of SAMA Elements with IEC 61131-3 Elements

The following SAMA elements are mapped to IEC 61131-3 elements:

| SAMA Element | IEC 61131-3 Element |
|---|---|
| SAMA Variable | Variable |
| Server Monitored Variable | Variable |
| Alarm Signal | Variable |
| Summing | Addition |
| Averaging | AVERAGE |
| Difference | Subtraction |
| Multiplying | Multiplication |
| Dividing | Division |
| Root Extraction | SQRT |
| Exponential | EXPT |
| Measuring or Readout | Variable |
| High Selecting | MAX |
| Low Selecting | MIN |
| Variable Signal Generator | Variable |
| Signal Monitor | LIM_ALRM |
| Logical Signal | Variable |
| Logical AND | AND |
| Logical OR | OR |
| Greater Than | Greater Than |
| Lesser Than | Less Than |
| Equal To | Equal |

## SAMA Keyboard Shortcuts

The following keyboard shortcuts are available for use with the SAMA language. Some shortcuts do not apply or may differ while debugging.

| | |
|---|---|
| Ctrl+A | Selects all elements (not available while debugging) |
| Ctrl+C | Copies the selected elements to the clipboard (not available while debugging) |
| Ctrl+V | Pastes elements saved on the clipboard to the insertion point (not available while debugging) |
| Ctrl+X | Cuts the selected elements to the clipboard (not available while debugging) |
| Ctrl+Y | Redoes the previous command (not available while debugging) |
| Ctrl+Z | Undoes the previous command (not available while debugging) |
| Shift+Ctrl+Alt+G | Enables/disables the grid in the language container |
| Shift+Alt+Enter | Toggles between full-screen and windowed modes |
| Ctrl+R | Toggles between Auto-Input and Manual-Input. Auto-Input automatically opens the Block Selector and Variable Selector (not available while debugging). |
| Ctrl+B | Bolds selected comment text (not available while debugging) |
| Ctrl+I | Italicizes selected comment text (not available while debugging) |
| Ctrl+U | Underlines selected comment text (not available while debugging) |
| Ctrl+Page Up | Jumps to the top edge of the language container |
| Ctrl+Page Down | Jumps to the bottom edge of the language container |
| Up Arrow | Scrolls up |
| Down Arrow | Scrolls down |
| Left Arrow | Scrolls left |
| Right Arrow | Scrolls right |
| Alt+Up Arrow | Scrolls up |
| Alt+Down Arrow | Scrolls down |
| Alt+Left Arrow | Scrolls left |
| Alt+Right Arrow | Scrolls right |

| | |
|---|---|
| Ctrl+Up Arrow | Aligns the selected elements with the highest element. While debugging, slowly scrolls up. |
| Ctrl+Down Arrow | Aligns the selected elements with the lowest element. While debugging, slowly scrolls down. |
| Ctrl+Left Arrow | Aligns the selected elements with the leftmost element. While debugging, slowly scrolls left. |
| Ctrl+Right Arrow | Aligns the selected elements with the rightmost element. While debugging, slowly scrolls right. |
| Delete | Removes the selected elements (not available while debugging) |
| Ctrl+D | Only available in debug mode for the date data type. When the Write Logical Value dialog box is open, enters the current date. |

# Language Reference

The language reference includes information about the usage and limitations of various project elements and other aspects:

- Programs
- Functions
- Function Blocks
- Execution Rules
- Reserved Keywords
- Variables
- Directly Represented Variables
- Defined Words
- Data Types
- Literal Values

# Programs

Programs, also known as POUs, are logical programming units describing operations between variables of a process. Programs describe either sequential or cyclic operations. Cyclic programs are executed at each target system cycle. Sequential programs, representing sequential operations, are grouped together. The execution of sequential programs has a dynamic behavior.

Programs before and after sequential programs describe cyclic operations. Cyclic programs are not time-dependent. Cyclic programs are systematically executed at the beginning of each run time cycle. Main sequential programs (at the top of the hierarchy) are executed according to their respective dynamic behavior.

| | |
|---|---|
| Begin | Cyclic operations (FDB, LD, ST, SAMA, IEC 61499) |
| Sequential | Sequential operations (SFC, SFC child) |
| End | Cyclic operations (FDB, LD, ST, SAMA, IEC 61499) |

Programs located at the beginning of a cycle (before sequential programs) typically describe preliminary operations on input devices to build high level filtered variables. Sequential programs frequently use these variables. Programs located at the end of the cycle (after sequential programs) typically describe security operations on the variables operated on by sequential programs, before sending values to output devices.

Programs are described using the available graphic or literal languages. You specify the programming language when creating a program; you cannot change the programming language for an existing program.

POUs defined as programs are executed on the target system respecting the order shown in the Programs section.

Within resources, you need to respect the hierarchy of programs. Programs are linked together in a hierarchical tree. Those placed at the top of the hierarchy are activated by the system. Child-programs (lower level of the hierarchy) are activated by their parent.

POUs (programs, functions, and function blocks) within a project and dependency libraries must have unique names. These names can have up to 128 characters and must begin with a letter.

Projects can contain up to 65 536 programs.

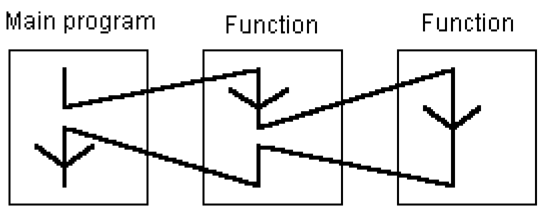
#### See Also

Execution Rules

# Functions

Functions are POUs having one or more input parameters and one output parameter. A function can be called by a program, a function or a function block. A function has no instance meaning that local data is not stored and is usually lost from one call to the other.

The execution of a function is driven by its parent program. Therefore, the execution of the parent program is suspended until the function ends:




Any POU of any section can call one or more functions. A function can have local variables.

**ISaGRAF** does not support recursivity during function calls. When a function of the *Functions* section is called by itself or one of its called functions, a build error occurs. Furthermore, functions do not store the local values of their local variables. Since functions are not instantiated, these cannot call function blocks.

The interface of a function must be explicitly defined with a type and a unique name for each of its calling (input) parameters or return (output) parameter. Functions can have up to 127 calling parameters and one return parameter. Return parameters can only have Elementary IEC 61131-3 Types.

POUs (programs, functions, and function blocks) within a project and dependency libraries must have unique names. Function names and function parameter names can have up to 128 characters. Function parameter names can begin with a letter followed by letters, digits, and single underscores.

When the *Function Internal State Enable* resource property is set to *True*, local variables having the *var* direction are initialized using their initial values only at run-time startup. When set to *False*, function calls initialize local variables, having the *var* direction, at every call.

# Function Blocks

Function blocks are POUs having multiple input and output parameters. These are instantiated meaning local variables of a function block are copied for each instance. When calling a function block in a program, you actually call the instance of the block where the same code is called but the data used is that which has been allocated to the instance. The values of the variables of an instance are stored from one cycle to the other.

Function blocks can be called by any POU in the project. Function blocks can call functions or other function blocks.

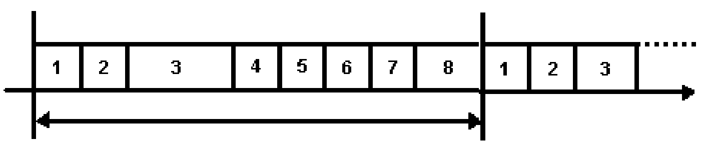
The interface of a function block must be explicitly defined with a type and a unique name for each of its calling (input) parameters or return (output) parameters. Function blocks can have more than one output parameter. The value of a return parameter for a function block differs for the various programming languages.

POUs (programs, functions, and function blocks) within a project and dependency libraries must have unique names. Function block names and function block parameter names can have up to 128 characters. Function block parameter names can begin with a letter followed by letters, digits, and single underscores.

# Execution Rules

The execution of a control application for a resource follows eight main steps within a loop. The duration of this loop is defined as the cycle timing for a resource.

1. Scan input variables
2. Consume bound variables
3. Execute POUs
4. Produce bound variables
5. Update output variables
6. Save retained values
7. Process IXL messages
8. Sleep until next cycle




In a case where bindings are defined, variables consumed by a resource are updated after the inputs are scanned and the variables produced for other resources are sent before updating outputs.

When a cycle time is specified, a resource waits until this time has elapsed before starting the execution of a new cycle. The POUs execution time varies depending on the size of the application. When a cycle exceeds the specified time, the loop continues to execute the cycle but sets an overrun flag. In such a case, the application no longer runs in real time.

When a cycle time is not specified, a resource performs all programs then restarts a new cycle without waiting.

# Reserved Keywords

Reserved keywords are unavailable for use as names of POUs or variables.

| | |
|---|---|
| **_** | _AND, _CALL, _CALL_IEC_SFC_FB, _END, _GOTO, _IF, _NOT, _PUSH_PAR, _OR, _POP_CSTK, _PUSH_CSTK, _RET, _STEP, _XOR |
| **A** | ABS, ACOS, ADD, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN, |
| **B** | BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BINDING, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE, |
| **C** | CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS, |
| **D** | DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD, |
| **E** | ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT, |
| **F** | FALSE, FIND, FOR, FUNCTION, |
| **G** | GE, GFREEZE, GKILL, GLOBALVARIABLE, GRST, GSTART, GSTATUS, GT, |
| **H** | HEADER, |
| **I** | IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME, IO, |
| **J** | JMP, JMPC, JMPCN, JMPN, JMPNC, |
| **L** | LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD, |
| **M** | MAX, MID, MIN, MOD, MOVE, MUL, MUX, |
| **N** | NE, NOT, |
| **O** | OF, ON, OR, OR_MASK, ORN, |
| **P** | PROGRAM |
| **R** | R, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL, REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REPEAT, REPLACE, RESOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR, |

**S** S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD, STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME, STRUCT, SUB, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO, SYS_SAVTMR,

**T** TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL, TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TO, TOD, TRUE, TYPE,

**U** UDINT, UINT, ULINT, UNTIL, USINT,

**V** VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, VAR_OUTPUT

**W** WHILE, WITH, WORD

**X** XOR, XOR_MASK, XORN

# Variables

The scope of variables can be local to a POU or global to a resource. Local variables are available for use within one POU only. Global variables are available for use within any POU of the resource. Variables have the following properties:

- Name, limited to 128 characters beginning with a letter or underscore character followed by letters, digits, and single underscore characters. These cannot have two consecutive underscore characters.
- Logical Value, available when online. The displayed value differs depending on the direction of the variable: varInputs are locked values, varOutputs are updated by the running TIC code, and var values are locked.
- Physical Value, available when online. The displayed value differs depending on the direction of the variable: varInputs are updated by the field value, varOutputs are locked, and var values are updated by the running TIC code.
- Lock, available when online. The indication of whether the value of the variable is locked. Locking operates differently for simple variables, array and structure elements, and function block parameters. For simple variables, individual variables are locked directly. For structure and array elements, locking an element locks all the elements of the structure or array. Possible values are Yes or No.
- Data Type, possible values are BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL, TIME, DATE, STRING, Array types, Structure types, and Function blocks
- Dimension, the size (number of elements) of an array. For example: [1..3,1..10] - represents a two-dimensional array containing a total of 30 elements.
- String Size, indicates the maximum length for string-type variables. String capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string.
- Initial Value, value held by a variable when the virtual machine starts the execution of the resource. The initial value of a variable can be the default value, a value given by the user when the variable is defined or the value of the retain variable after the virtual machine has stopped. You can set initial values for POU variables and global variables. You can

set initial values for local variables of functions and instances of function blocks. The format is comma separated values (CSV).

- Direction, for I/O wiring, function or function block, indicates whether a variable is an input (varInput), output (varOutput), or internal (var). The direction of a variable affects the logical value and physical value.
- Attribute, property of a variable indicating its read and write access rights. Possible values are read-only, write-only, and read-write.
- Retained, the indication of whether the value of the variable is saved by the virtual machine at each cycle. Possible values are Yes or No.
- Comment, user-defined free-format text
- Alias, any name (for use in POUs) limited to 128 characters beginning with a letter or underscore character followed by letters, digits, and single underscore characters. These cannot have two consecutive underscore characters.
- Wiring, (read-only cell) generated by the I/O wiring tool indicating the I/O channel to which the variable is wired. You can only wire POU variables and global variables; you cannot wire functions and function blocks. Uses the syntax of Directly Represented Variables.
- Address, user-defined address of the variable. The format is hexadecimal and the value ranges from 1 to FFFF.
- Retained Flags, available when the Retained property is selected for a variable and supported by the target type. Enables retaining specific elements of a variable whereas the Retained property applies to the entire variable. Also indicates whether to use, at the beginning of the cycle, the initial value of the variable or the value previously retained on the target. The format is comma separated values (CSV), where True indicates retaining elements of the structure, array, or function block instance.
- Groups, variable groups containing the variable listed in alphabetical order.
- Comment Fields, user-defined free-format text available for array elements. Each array element of the same type can have a different comment. The format is comma separated values (CSV).

Although function block instances are declared using variables, these variables do not follow rules applying to elementary or derived type variables. These variables can only have the var direction and the read-write attribute.

# Directly Represented Variables

The system enables the use of directly represented variables in the source of programs to represent a free channel. Free channels are those not linked to a declared I/O variable. The identifier of a directly represented variable always begins with the "%" character.

The naming conventions of a directly represented variable for a channel of a single I/O device. "***s***" is the slot number of the I/O device. "***c***" is the number of the Channel:

| | |
|---|---|
| **%IXs.c** | free Channel of a Boolean input I/O device |
| **%IBs.c** | free Channel of a Short integer, Unsigned short integer, or BYTE input I/O device |
| **%IWs.c** | free Channel of an Integer, Unsigned integer, or WORD input I/O device |
| **%IDs.c** | free Channel of a Double integer, Unsigned double integer, Double word, or DATE input I/O device |
| **%ILs.c** | free Channel of a Long integer, Unsigned long integer, Long word, or Long real input I/O device |
| **%IRs.c** | free Channel of a Real input I/O device |
| **%ITs.c** | free Channel of a Time input I/O device |
| **%ISs.c** | free Channel of a String input I/O device |
| **%QXs.c** | free Channel of a Boolean output I/O device |
| **%QBs.c** | free Channel of a Short Integer, Unsigned short integer, or BYTE output I/O device |
| **%QWs.c** | free Channel of an Integer, Unsigned integer, or WORD output I/O device |
| **%QDs.c** | free Channel of a Double integer, Unsigned double integer, Double word, or DATE output I/O device |
| **%QLs.c** | free Channel of a Long integer, Unsigned long integer, Long word, or Long real output I/O device |
| **%QRs.c** | free Channel of a Real output I/O device |
| **%QTs.c** | free Channel of a Time output I/O device |
| **%QSs.c** | free Channel of a String output I/O device |

The naming conventions of a directly represented variable for a Channel of a complex device. "s" is the slot number of the device. "b" is the index of the single I/O device within the complex device. "c" is the number of the Channel:

| | |
|---|---|
| **%IXs.b.c** | free Channel of a Boolean input I/O device |
| **%IBs.b.c** | free Channel of a Short Integer, Unsigned short integer, or BYTE input I/O device |
| **%IWs.b.c** | free Channel of an Integer, Unsigned integer, or WORD input I/O device |
| **%IDs.b.c** | free Channel of a Double integer, Unsigned double integer, Double word, or DATE input I/O device |
| **%ILs.b.c** | free Channel of a Long integer, Unsigned long integer, Long word, or Long real input I/O device |
| **%IRs.b.c** | free Channel of an Real input I/O device |
| **%ITs.b.c** | free Channel of a Time input I/O device |
| **%ISs.b.c** | free Channel of a String input I/O device |
| **%QXs.b.c** | free Channel of a Boolean output I/O device |
| **%QBs.b.c** | free Channel of a Short Integer, Unsigned short integer, or BYTE output I/O device |
| **%QWs.b.c** | free Channel of an Integer, Unsigned integer, or WORD output I/O device |
| **%QDs.b.c** | free Channel of a Double integer, Unsigned double integer, Double word, or DATE output I/O device |
| **%QLs.b.c** | free Channel of a Long integer, Unsigned long integer, Long word, or Long real output I/O device |
| **%QRs.b.c** | free Channel of a Real output I/O device |
| **%QTs.b.c** | free Channel of a Time output I/O device |
| **%QSs.b.c** | free Channel of a String output I/O device |

### Example

%QX1.6 6th channel of the I/O device #1 (boolean output)
%ID2.1.7 7th channel of the I/O device #1 in the device #2 (integer input)

# Defined Words

**ISaGRAF** supports the use of identifier names, called defined words. When building, defined words are replaced by the variables and expressions these represent. Defined words have a global scope, i.e., these are available for use in any POU of any resource of a project.

For POUs, a defined word can replace literal expressions, boolean expressions, reserved keywords, or complex ST expressions

The following are examples of defined words:

Arrays | Structures | Defined Words

| Word | Equivalent | Comment |
|---|---|---|
| OK | (auto_mode AND NOT (alarm)) | |
| PI | 3.14159 | |
| YES | TRUE | |

When such an equivalence is defined, its identifier is available anywhere in the project to replace the attached expression. The following ST programming example uses defined words:

```
If OK Then
angle := PI / 2.0;
isdone := YES;
End_if;
```

The naming of defined words must conform to the following rules:

- contain up to 128 characters
- the first character must be a letter and subsequent characters can be letters, digits, and single underscore ('_') characters. The last character can be either a letter or a digit.

The definition of a defined word cannot contain a defined word. Note the invalid definition (with strikethrough mark) in the following defined word examples:

PI is 3.14159 PI2 is 6.28318
~~PI2 is PI*2~~

# Data Types

Any literal, expression, or variable used in a POU (written in any language) must be characterized by a data type. Data type coherence must be followed in graphic operations and literal statements. Data types are one of the following types:

- Elementary IEC 61131-3 Types
- Derived Types: Arrays
- Derived Types: Structures

## Elementary IEC 61131-3 Types

You can program objects using the following elementary IEC 61131-3 types:

- ANY: user-defined type enabling overloading "C" function block inputs to support specified IEC 61131-3 data types as well as specified complex types such as arrays and structures
- ANY_ELEMENTARY: overloads "C" function block inputs to support all of the IEC 61131-3 elementary data types
- BOOL: logic (true or false) value
- SINT: short integer value (8 bit)
- USINT: unsigned short integer value (8 bit)
- BYTE: byte value (8 bit)
- INT: single integer value (16 bit)
- UINT: unsigned single integer value (16 bit)
- WORD: word value (16 bit)
- DINT: double integer value (32 bit)
- UDINT: unsigned double integer value (32 bit)
- DWORD: double word value (32 bit)
- LINT: long integer value (64 bit)
- ULINT: unsigned long integer value (64 bit)
- LWORD: long word value (64 bit)
- REAL: real (floating) value (32 bit)
- LREAL: long real (floating) value (64 bit)

- TIME: time values less than 49d17h2m47s295ms; these value types cannot store dates (32 bit)
- DATE: date values (32 bit)
- STRING: character string having a defined *size*, representing the maximum number of characters the string can contain.

Based on the above elementary IEC 61131-3 types, you can define new user types. Furthermore, you can define arrays or structures using elementary IEC 61131-3 types, arrays, or other user types.

When creating a variable, a dimension can be given to define an array. The following example shows the MyVar variable of type BOOL having a dimension defined as follows:

```
[1..10]
```

```
FOR i = 1 TO 10 DO
MyVar[i] := FALSE;
END_FOR;
```

## ANY Data Type

**Note:** For use, your target must support the ANY data type.

The ANY data type is a user-defined type enabling overloading "C" function block inputs to support specified IEC 61131-3 data types (BOOL, BYTE, DATE, DINT, DWORD, INT, LINT, LREAL, LWORD, REAL, SAFEBOOL, SINT, TIME, UDINT, UINT, ULINT, USINT, and WORD) as well as specified complex types such as arrays and structures.

**Warning:** You can only pass user-defined arrays to "C" function block inputs having defined array dimensions.

## ANY_ELEMENTARY Data Type

**Note:** For use, your target must support the ANY_ELEMENTARY data type.

The ANY_ELEMENTARY data type enables overloading "C" function block inputs to support all of the following IEC 61131-3 elementary data types: BOOL, BYTE, DATE, DINT, DWORD, INT, LINT, LREAL, LWORD, REAL, SAFEBOOL, SINT, TIME, UDINT, UINT, ULINT, USINT, and WORD.

**Warning:** You can only pass user-defined arrays to "C" function block inputs having defined array dimensions.

## Boolean Data Type

Boolean variables (BOOL) can take one of the Boolean values: **TRUE** or **FALSE**. Boolean variables are typically used in Boolean expressions.

For Boolean literal expressions, **ISaGRAF** targets evaluate all parts of such expressions. Whereas, the IEC 61131-3 standard states that Boolean expressions may be evaluated only to the extent necessary to determine the resultant value. In the following example according to the IEC 61131-3 standard, if B is zero then the first expression `(B <> 0)` is false and the second expression `(A/B > 0)` is not performed.

```
if ((B <> 0) and (A/B > 0)) then

GREATER := true;

else

GREATER := false;

end_if;
```

Boolean literal values are the following:

- TRUE is equivalent to the integer value 1
- FALSE is equivalent to the integer value 0

## Short Integer Data Type

Short Integer (SINT) variables are 8-bit signed integers from -128 to +127.

A bit of a short integer variable, array, structure, or the output of a function block instance can be accessed using the following syntax:

```
MyVar.i
```

If MyVar is a short Integer.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 7.

Short integer literal values represent signed integer (8 bit) values:

from -128 to +127

Short integer constants may be expressed with one of the following Bases. Short integer constants must begin with a Prefix that identifies the Bases used:

| **Base** | **Prefix** | **Example** |
|---|---|---|
| DECIMAL | (none) | 19 |
| HEXADECIMAL | "16#" | 16#A1 |
| OCTAL | "8#" | 8#27 |
| BINARY | "2#" | 2#0101_0101 |

The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve literal value readability.

## Unsigned Short Integer or BYTE Data Type

Unsigned Short Integer (USINT) or BYTE variables are 8-bit unsigned integers from 0 to 255.

A bit of an unsigned short integer or BYTE variable, array, structure, or the output of a function block instance can be accessed using the following syntax:

```
MyVar.i
```

If MyVar is an unsigned short integer or BYTE.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 7

Unsigned short integer and BYTE literal values represent unsigned integer (8 bit) values:

from 0 to 255

Short integer and BYTE constants may be expressed with one of the following **Bases**. These constants must begin with a **Prefix** that identifies the Bases used:

| **Base** | **Prefix** | **Example** |
|---|---|---|
| DECIMAL | (none) | 19 |
| HEXADECIMAL | "16#" | 16#A1 |
| OCTAL | "8#" | 8#27 |
| BINARY | "2#" | 2#0101_0101 |

The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve literal value readability.

## Integer Data Type

Integer (INT) variables are 16-bit signed integers from -32768 to 32767.

A bit of an integer variable, array, structure, or the output of a function block instance can be accessed using the following syntax:

```
MyVar.i
```

If MyVar is an Integer.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 15.

Integer literal values represent signed integer (16 bit) values:

from -32768 to 32767

Integer constants may be expressed with one of the following Bases. Integer constants must begin with a Prefix that identifies the Bases used:

| **Base** | **Prefix** | **Example** |
|---|---|---|
| DECIMAL | (none) | -260 |
| HEXADECIMAL | "16#" | 16#FEFC |
| OCTAL | "8#" | 8#177374 |
| BINARY | "2#" | 2#0101_0101_0101_0101 |

The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve literal value readability.

## Unsigned Integer or Word Data Type

Unsigned Integer (UINT) or WORD variables are 16-bit unsigned integers from 0 to 65535.

A bit of an unsigned integer or WORD variable, array, structure, or the output of a function block instance can be accessed using the following syntax:

```
MyVar.i
```

If MyVar is an unsigned integer or WORD.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 15.

Unsigned integer and WORD literal values represent unsigned integer (16 bit) values:

from 0 to 65535

Unsigned integer and WORD constants may be expressed with one of the following Bases. These constants must begin with a Prefix that identifies the Bases used:

| **Base** | **Prefix** | **Example** |
|---|---|---|
| DECIMAL | (none) | +33000 |
| HEXADECIMAL | "16#" | 16#80E8 |
| OCTAL | "8#" | 8#100350 |
| BINARY | "2#" | 2#0101_0101_0101_0101 |

The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve literal value readability.

## Double Integer Data Type

Double Integer (DINT) variables are 32-bit signed integers from -2147483648 to +2147483647.

A bit of a double integer variable, array, structure, or the output of a function block instance can be accessed using the following syntax:

```
MyVar.i
```

If MyVar is an Integer.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 31.

Double integer literal values represent signed double integer (32 bit) values:

from -2147483648 to +2147483647

Double integer constants may be expressed with one of the following Bases. Double integer constants must begin with a Prefix that identifies the Bases used:

| Base | Prefix | Example |
|---|---|---|
| DECIMAL | (none) | -908 |
| HEXADECIMAL | "16#" | 16#1A2B3C4D |
| OCTAL | "8#" | 8#1756402 |
| BINARY | "2#" | 2#1101_0001_0101_1101_0001_0010_1011_1001 |

The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve literal value readability.

## Unsigned Double Integer or Double Word Data Type

Unsigned Double Integer (UDINT) or Double Word (DWORD) variables are 32-bit unsigned integers from 0 to 4294967295.

A bit of an unsigned double integer or double word variable, array, structure, or the output of a function block instance can be accessed using the following syntax:

```
MyVar.i
```

If MyVar is an unsigned double integer or double word.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 31.

Unsigned double integer and Double Word literal values represent unsigned double integer (32 bit) values:

from 0 to 4294967295

Double integer and double word constants may be expressed with one of the following Bases. Double integer and double word constants must begin with a Prefix that identifies the Bases used:

| **Base** | **Prefix** | **Example** |
|---|---|---|
| DECIMAL | (none) | +908 |
| HEXADECIMAL | "16#" | 16#1A2B3C4D |
| OCTAL | "8#" | 8#1756402 |
| BINARY | "2#" | 2#1101_0001_0101_1101_0001_0010_1011_1001 |

The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve literal value readability.

## Long Integer Data Type

Long Integer (LINT) variables are 64-bit signed integers from -9223372036854775808 to 9223372036854775807.

A bit of a long integer variable, array, structure, or the output of a function block instance can be accessed using the following syntax:

```
MyVar.i
```

If MyVar is a long integer.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 63.

Long integer literal values represent signed long integer (64 bit) values:

from -9223372036854775808 to 9223372036854775807

Long integer constants may be expressed with one of the following Bases. Long integer constants must begin with a Prefix that identifies the Bases used:

| **Base** | **Prefix** | **Example** |
|---|---|---|
| DECIMAL | (none) | -908 |
| HEXADECIMAL | "16#" | 16#1A2B3C4D |
| OCTAL | "8#" | 8#1756402 |
| BINARY | "2#" | 2#1101_0001_0101_1101_0001_0010_1011_1001_<br>1101_0001_0101_1101_0001_0010_1011_1001 |

The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve literal value readability.

## Unsigned Long Integer or Long Word Data Type

Unsigned Long Integer (ULINT) or Long Word (LWORD) variables are 64-bit unsigned integers from 0 to 18446744073709551615.

A bit of an unsigned long integer or long word variable, array, structure, or the output of a function block instance can be accessed using the following syntax:

```
MyVar.i
```

If MyVar is an unsigned long integer or long word.
MyVar.i is a Boolean. "i" must be a constant value from 0 to 63.

Unsigned long integer and long word literal values represent unsigned long integer (64 bit) values:

from 0 to 18446744073709551615

Unsigned long integer and long word constants may be expressed with one of the following Bases. Long integer and long word constants must begin with a Prefix that identifies the Bases used:

| **Base** | **Prefix** | **Example** |
|---|---|---|
| DECIMAL | (none) | +908 |
| HEXADECIMAL | "16#" | 16#1A2B3C4D |
| OCTAL | "8#" | 8#1756402 |
| BINARY | "2#" | 2#1101_0001_0101_1101_0001_0010_1011_1001_<br>1101_0001_0101_1101_0001_0010_1011_1001 |

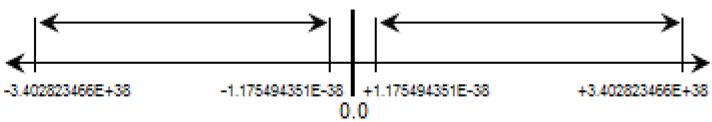
The underscore character ('_') may be used to separate groups of digits. It has no particular significance other than to improve literal value readability.

## Real Data Type

Real variables are standard IEEE 32-bit floating values (single precision).

1 sign bit + 23 mantissa bits + 8 exponent bits

A real variable has six significant digits. For larger values, the maximum possible value is ±3.402823466E+38 while for smaller values, the minimum possible value is ±1.175494351E-38. Therefore, values greater than ±3.402823466E+38 and greater than 0.0 but less than ±1.175494351E-38 are not supported. The following example shows the value ranges including 0.0 that are supported for real variables:




Real literal values can be written with either Decimal or Scientific representation. The decimal point ('.') separates the Integer and Decimal parts. The decimal point must be used to differentiate a Real literal value from an Integer one. The scientific representation uses the letter 'E' to separate the mantissa part and the exponent. The exponent part of a real scientific value must be a signed integer value from -37 to +37. A real variable has six significant digits.

### Example

| | |
|---|---|
| 3.14159 | -1.0E+12 |
| +1.0 | 1.0E-15 |
| -789.56 | +1.0E-37 |

The value "123" does not represent a Real literal value. Its correct real representation is "123.0".

## Long Real Data Type

Long Real (LREAL) variables are standard IEEE 64-bit floating values (double precision).

1 sign bit + 52 mantissa bits + 11 exponent bits

A long real variable has 15 significant digits. For larger values, the maximum possible value is ±1.7976931348623158e+308 while for smaller values, the minimum possible value is ±2.22507385850721E-308. Therefore, values greater than ±1.7976931348623158e+308 and greater than 0.0 but less than ±2.22507385850721E-308 are not supported. The following example shows the value ranges including 0.0 that are supported for long real variables:


Long real literal values can be written with either Decimal or Scientific representation. The decimal point ('.') separates the Integer and Decimal parts. The decimal point must be used to differentiate a Real literal value from an Integer one. The scientific representation uses the letter 'E' to separate the mantissa part and the exponent. The range of a real scientific expression must be a signed integer value from 1.7E -308 to 1.7E +308. A long real variable has 15 significant digits.

### Example

| | |
|---|---|
| 3.14159 | -1.0E+12 |
| +1.0 | 1.0E-15 |
| -789.56 | +1.0E-37 |

The value "123" does not represent a long real literal value. Its correct real representation is "123.0".

## Time Data Type

Time variables are typically used in Time expressions. A Time value represents values from 0 to 49d17h2m47s294ms. Time variables are stored in 32 bit words. The internal representation is a positive number of milliseconds. Time variables can be used with timer function blocks such as TOF and TON.

Time literal values represent time values from 0 to 49d17h2m47s294ms. The lowest allowed unit is a millisecond. Standard time units used in literal values are:

| | |
|---|---|
| Days | The "d" letter must follow the number of days |
| Hours | The "h" letter must follow the number of hours |
| Minutes | The "m" letter must follow the number of minutes |
| Seconds | The "s" letter must follow the number of seconds |
| Milliseconds | The "ms" letters must follow the number of milliseconds |

The time literal value must begin with "T#" or "TIME#" prefix. Prefixes and unit letters are not case sensitive. Some units may not appear.

When the TIME value is equal to -1 (as a DINT value), the value is considered as overflow and invalid. For example:

IF ANY_TO_DINT(TIME1) = -1 then
(* Handle overflow *)
END_IF;

### Example

T#1D1H450MS 1 day, 1 hour, 450 milliseconds
time#1H3M 1 hour, 3 minutes

The following ST code gets the current time for use in the clock portion of a date variable:

```
NOW_1(); (* Instance of the NOW function block *)

date1 := any_to_date(NOW_1.sec); (* Casts the seconds of NOW into a
date *)
```

```
clock := any_to_time(MOD(NOW_1.sec,86400)*1000+NOW_1.nsec/100000); (*
Gets the current time *)
```

## Date Data Type

Date variables have date values and are typically used in Date expressions. A Date value ranges from 1970-01-01 to 2038-01-18. Date variables are stored using the 32 bit ISO 'C' time_t data type. The internal representation is a positive number of seconds since 1970-01-01 at midnight GMT.

Date literal expressions represent date values in the year-month-day format, separated by hyphens. Possible date literal expressions range from DATE#1970-01-01 to DATE#2038-01-18 GMT.

The date literal expression must begin with "D#" or "DATE#" prefix. Prefixes and unit letters are not case sensitive.

### Example

D#2005-02-20
date#2005-02-20

## String Data Type

String variables contain character strings. The length of the string can change during process operations. The length of a string variable cannot exceed the capacity (maximum length) specified when the variable is declared. String capacity is limited to 252 characters excluding the terminating null character (0), a byte for the current length of the string, and a byte for the maximum length of the string. When declaring string variables, the maximum number of characters is defined in the String Size column of the Dictionary or Variable Selector.

String variables can contain any character of the standard ASCII table (ASCII code from 0 to 255). The null character (0) can exist in a character string, however, it indicates the end of the string.

String literal values represent character strings. Characters must be preceded and followed by single quote (') characters. For example:

'THIS IS A MESSAGE'

**Warning:** A string literal expression must be expressed on one line of the program source code. When placing single quote (') characters within a string literal, these characters must be preceded by the dollar ($) character. In the following string literal, note the dollar character preceding the single quote character.

'THIS IS $' A MESSAGE'

A string literal value must be expressed on one line of the program source code. Its length cannot exceed 252 characters, including spaces.

Empty string literal values are represented by two single quote (') characters, with no space or tab character between them:

'' (* this is an empty string *)

The dollar ('**$**') special character, followed by other special characters, can be used in a string literal values to represent a non-printable character:

| **Sequence** | **Meaning** | **ASCII (hex)** | **Example** |
|---|---|---|---|
| $$ | '$' character | 16#24 | 'I paid $$5 for this' |
| $' | apostrophe | 16#27 | 'Enter $'Y$' for YES' |

| | | | |
|---|---|---|---|
| $L | line feed | 16#0a | 'next $L line' |
| $R | carriage return | 16#0d | ' llo $R He' |
| $N | new line | 16#0d0a | 'This is a line$N' |
| $P | new page | 16#0c | 'lastline $P first line' |
| $T | tabulation | 16#09 | 'name$Tsize$Tdate' |
| $hh (*) | any character | 16#hh | 'ABCD = $41$42$43$44' |

(*) "hh" is the hexadecimal value of the ASCII code for the expressed character.

## Safety Type

You can program objects using a SAFE data type:

- SAFEBOOL: logic (true or false) value for binary safety signals only

## Safety Boolean Data Type

Safety Boolean (SAFEBOOL) variables behave the same way as Boolean variables and are typically used for binary safety Boolean signals. Safety Boolean variables can have one of two values: TRUE or FALSE. The false value indicates a safe value.

The safety data type recognizes that the signals are safety-relevant and must be treated with special care. You can connect safety Boolean variables to other safety Boolean variables only. You can apply safety Boolean variables to the inputs and outputs of functions, function blocks, and operators.

The workbench targets evaluate all parts of safety Boolean (SAFEBOOL) expressions in the same manner as Boolean expressions.

There are two safety Boolean constant expressions:

- TRUE is equivalent to the integer value 1
- FALSE is equivalent to the integer value 0

"True" and "False" keywords are not case-sensitive. For safety Boolean expressions, false is the default value and indicates a safe value.

### See Also

Boolean Data Type

# Derived Types: Arrays

You can define arrays of standard IEC 61131-3 types or derived types. An array has one or more dimensions. When an array is defined, a variable can be created with this type and a structure can have a field with this type. Array dimensions are positive DINT literal values and array indexes are DINT literal values or variables.

Array names can have up to 128 characters and can begin with letters or single underscores followed by letters, digits, and single underscores.

## Example

1. **One-dimensional array**:

   MyArrayType is an array of 10 BOOL. Its dimension is defined as follows: [1..10].
   MyVar is of type MyArrayType.
   Ok := MyVar[4];

2. **Two-dimensional array**:

   MyArrayType2 is an array of DINT. It has two dimensions defined as follows: [1..10,1..3]
   MyVar2 is of type MyArrayType2
   MyVar2[1,2] := 100;

3. **Array of an array**:

   MyVar3 is an array of MyArrayType; Its dimension is defined as follows [1..3]
   FOR I := 1 TO 3 DO
   FOR J := 1 TO 10 DO
   MyVar3[I][J] := FALSE;
   END_FOR;
   END_FOR;

## Derived Types: Structures

Users can define structures using elementary IEC 61131-3 types or derived types. A structure is composed of sub-entries called **Fields**. When a structure is defined, a variable can be created with this type.

### Example

MyStruct1 is composed of:

Field1 which is BOOL
Field2 which is DINT

MyStruct2 is composed of:

Field1 which is DINT
Field2 which is BOOL
Field3 which is an array of 10 DINT
Field4 which is of type MyStruct1

MyVar of type MyStruct2 can be used as follows:

```
Value1 := MyVar.Field1; (* Value1 is of type DINT *)
Ok1 := MyVar.Field2; (* Ok1 is of type BOOL *)
Tab[2] := MyVar.Field3[5]; (* Tab is an array of DINT *)
Value2 := MyVar.Field3[8]; (* Value2 is of type DINT *)
Ok2 := MyVar.Field4.Field1; (* Ok2 is of type BOOL *)
```

# Literal Values

You can type literal values in POUs written in textual and graphical languages, including ST, LD, and FBD. For literal values, the **ISaGRAF** compiler assigns appropriate data types. When the literal value exceeds the size of a specified data type, the compiler assigns a larger data type and generates an error. You can force the compiler to use a specific data type for a literal value.

For the following data types, the compiler evaluates the literal value and assigns the appropriate data type:

- For integer values, the default data type assigned is DINT. However, when the integer value exceeds the DINT data type, the compiler assigns a larger data type such as LINT.
- For floating point values, the default data type assigned is REAL. However, when the floating point value exceeds the REAL data type, the compiler assigns the larger LREAL data type.

When assigning larger data types for these literal values, the compiler may generate errors. To resolve such errors, you can specify, i.e force, the data type for compilation by using the following syntax:

- ANY_TO_*DataType(LiteralValue),* available for use with all data types.
  For example, `any_to_time(78)`
- *DataType#LiteralValue*, for use with LREAL, TIME, and DATE.
  For example, `LREAL#1.23456`

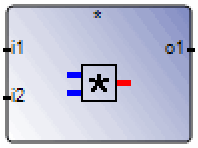
# Operators

The following are standard operators of the IEC 61131-3 languages:

| | | |
|---|---|---|
| **Arithmetic Operations** | Addition | Adds two or more variables |
| | Division | Divides two variables |
| | Multiplication | Multiplies two or more variables |
| | Subtraction | Subtracts a variable from another |
| | 1 GAIN | Assigns one variable into another |
| | NEG | Integer negation |
| **Boolean Operations** | AND | Boolean AND |
| | OR | Boolean OR |
| | XOR | Boolean exclusive OR |
| | NOT | Boolean negation |
| **Comparator Operations** | Less Than | Tests if one value is less than another |
| | Less Than or Equal | Tests if one value is less than or equal to another |
| | Greater Than | Tests if one value is greater than another |
| | Greater Than or Equal | Tests if one value is greater than or equal to another |
| | Equal | Tests if one value is equal to another |
| | Not Equal | Tests if one value is not equal to another |

| | | |
|---|---|---|
| **Data Conversion** | ANY_TO_BOOL | Converts to Boolean |
| | ANY_TO_SINT | Converts to Short integer |
| | ANY_TO_USINT | Converts to Unsigned short integer |
| | ANY_TO_BYTE | Converts to BYTE |
| | ANY_TO_INT | Converts to Integer |
| | ANY_TO_UINT | Converts to Unsigned integer |
| | ANY_TO_WORD | Converts to WORD |
| | ANY_TO_DINT | Converts to Double integer |
| | ANY_TO_UDINT | Converts to Unsigned double integer |
| | ANY_TO_DWORD | Converts to Double WORD |
| | ANY_TO_LINT | Converts to Long integer |
| | ANY_TO_ULINT | Converts to Unsigned long integer |
| | ANY_TO_LWORD | Converts to Long WORD |
| | ANY_TO_REAL | Converts to Real |
| | ANY_TO_LREAL | Converts to Long real |
| | ANY_TO_TIME | Converts to Time |
| | ANY_TO_DATE | Converts to Date |
| | ANY_TO_STRING | Converts to String |

# Multiplication




**Note:** The creation of additional inputs is supported.

Arguments:

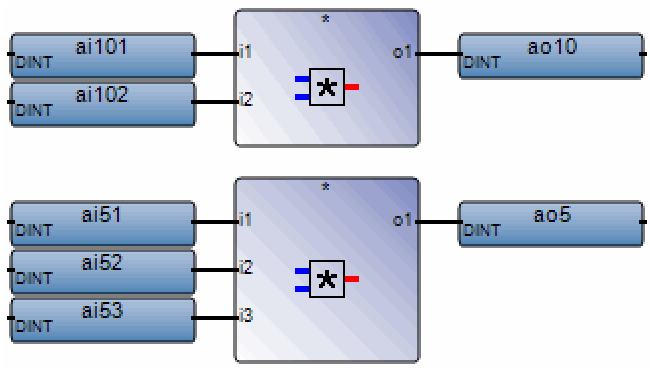
| | | |
|---|---|---|
| (inputs) | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL | can be INTEGER or REAL (all inputs must have the same format) |
| output | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL | **multiplication** of the input terms |

Description:

Multiplication of two or more integer or real variables.

### Example

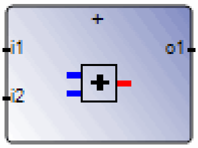
(* FBD example with Multiplication Operators *)




(* ST equivalence *)

```
ao10 := ai101 * ai102;
ao5 := (ai51 * ai52) * ai53;
```

# Addition




**Note:** The creation of additional inputs is supported.

Arguments:

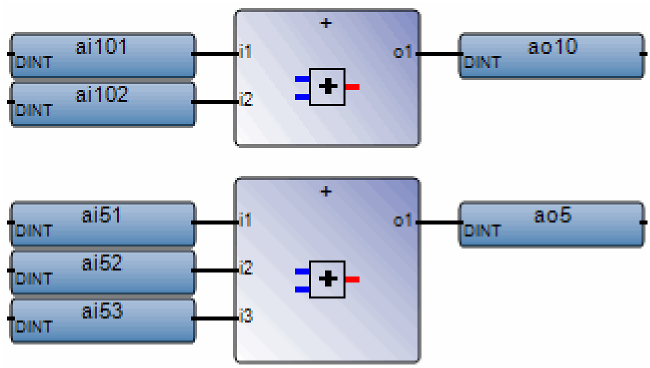
| | | |
|---|---|---|
| (inputs) | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - STRING | can be of any integer, real, TIME, or STRING format(all inputs must have the same format) |
| o1 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - STRING | **addition** of the input terms |

Description:

Addition of two or more integer, real, TIME, or STRING variables.

## Example

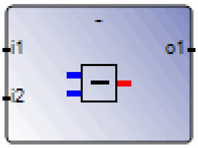
(* FBD example with Addition Operators *)




(* ST equivalence: *)

```
ao10 := ai101 + ai102;

ao5 := (ai51 + ai52) + ai53;
```

# Subtraction




Arguments:

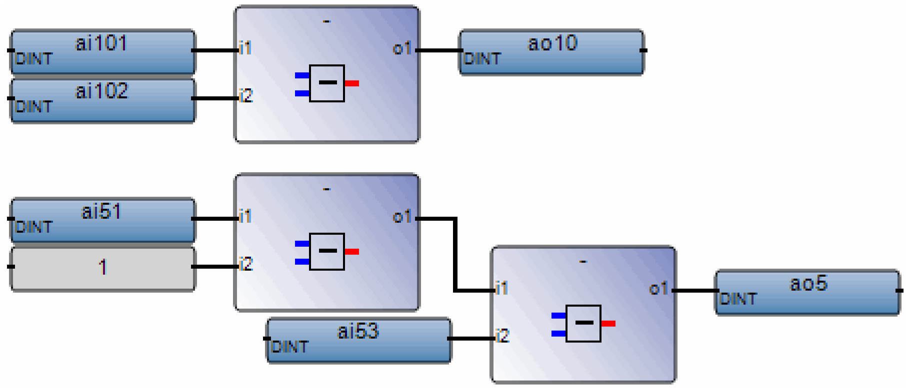
| | | |
|---|---|---|
| i1 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME | can be of any integer, real or long real, or TIME format |
| i2 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME | (i1 and i2 must have the same format) |
| o1 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME | subtraction (first minus second) |

Description:

Subtraction of two integer, real, or TIME variables.

### Example

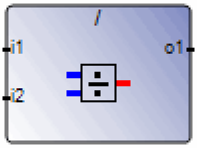
(* FBD example with Subtraction Operators *)




(* ST equivalence: *)

```
ao10 := ai101 - ai102;

ao5 := (ai51 - 1) - ai53;
```

# Division




Arguments:

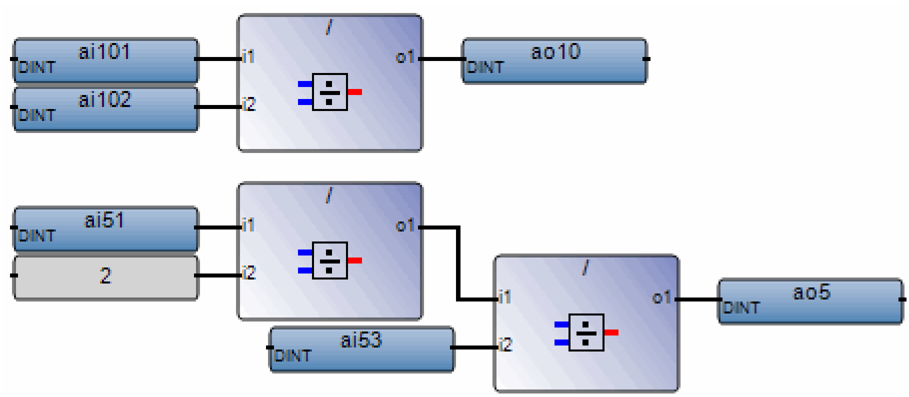
| | | |
|---|---|---|
| i1 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL | can be of any integer or real format (operand) |
| i2 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL | non-zero integer or real value (divisor)<br>(i1 and i2 must have the same format) |
| o1 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL | integer or real division of i1 by i2 |

Description:

Division of two integer or real variables (the first divided by the second).

### Example

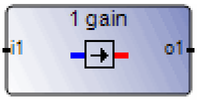
(* FBD example with Division Operators *)




(* ST Equivalence: *)

```
ao10 := ai101 / ai102;

ao5 := (ai5 / 2) / ai53;
```

# 1 GAIN




Arguments:

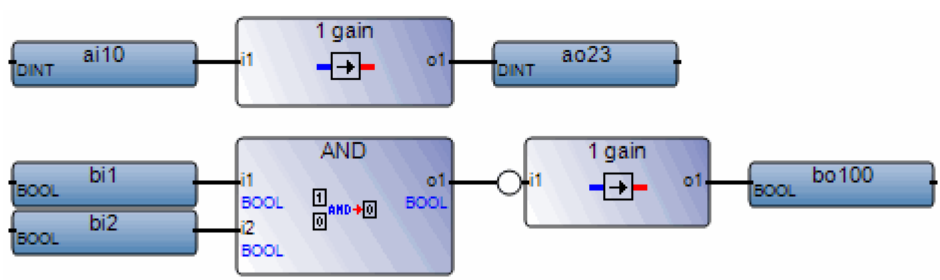
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | |
| o1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | i1 and o1 must have the same format |

Description:

Directly links the input to output. When used with a Boolean negation, inverts the state of the line connected to the output.

### Example

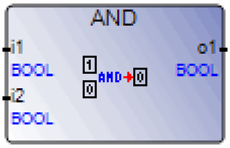
(* FBD example with assignment Operators *)




(* ST equivalence: *)

```
ao23 := ai10;

bo100 := NOT (bi1 AND bi2);
```

# AND




**Note:** The creation of additional inputs is supported.

Arguments:

| | | |
|---|---|---|
| (inputs) | BOOL | |
| o1 | BOOL | Boolean AND of the input terms |

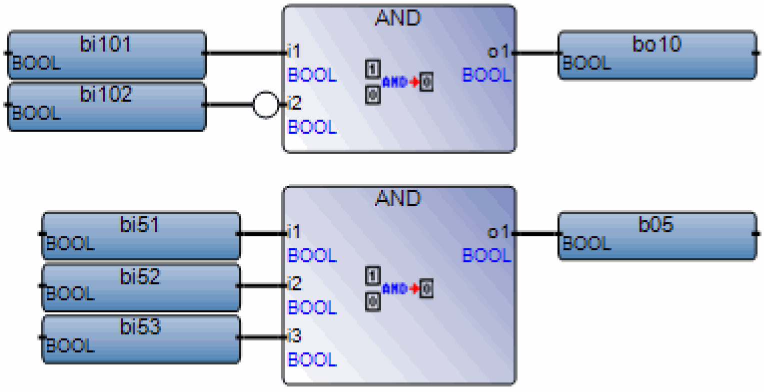
Description:

Boolean AND between two or more terms.

In the text editor, the '&' character can be used as well as typing AND.

### Example

(* FBD example with "AND" Operators *)




(* ST equivalence 1: *)

```
bo10 := bi101 AND NOT (bi102);
bo5 := (bi51 AND bi52) AND bi53;
```

(* ST equivalence 2: *)

```
bo10 := bi101 & NOT (bi102);
bo5 := (bi51 & bi52) & bi53;
```

# ANY_TO_BOOL


Arguments:

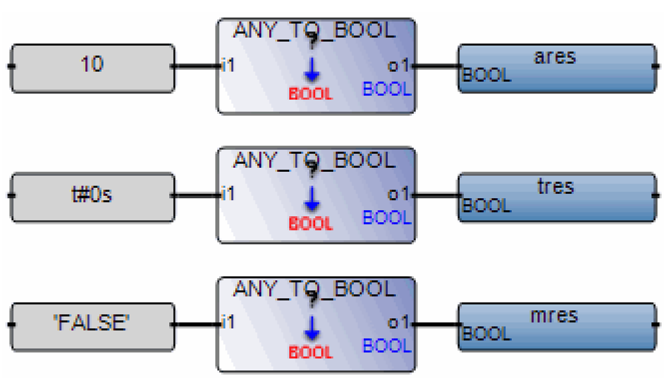
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | BOOL | TRUE for non-zero numerical value<br>FALSE for zero numerical value<br>TRUE for 'TRUE' string<br>FALSE for 'FALSE' string |

Description:

Converts variables to Boolean variables

### Example

(* FBD example with "Convert to Boolean" Operators *)




(* ST Equivalence: *)

```
ares := ANY_TO_BOOL (10);          (* ares is TRUE *)
tres := ANY_TO_BOOL (t#0s);        (* tres is FALSE *)
mres := ANY_TO_BOOL ('FALSE');     (* mres is FALSE *)
```

# ANY_TO_SINT


Arguments:

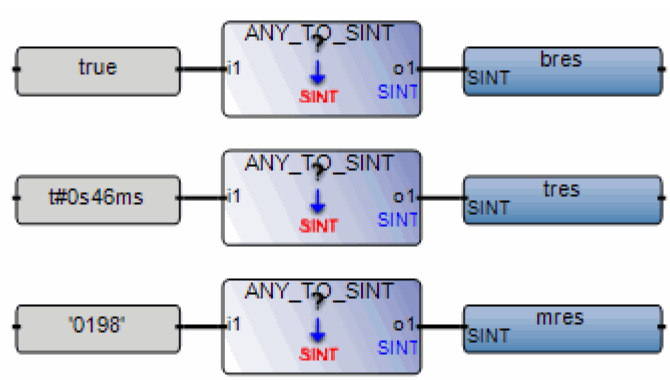
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | SINT | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

Description:

Converts variables to 8-bit short integer variables

### Example

(* FBD example with "Convert to Short Integer" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_SINT (true);         (* bres is 1 *)
tres := ANY_TO_SINT (t#0s46ms);     (* tres is 46 *)
mres := ANY_TO_SINT ('0198');       (* mres is 198 *)
```

# ANY_TO_USINT


Arguments:

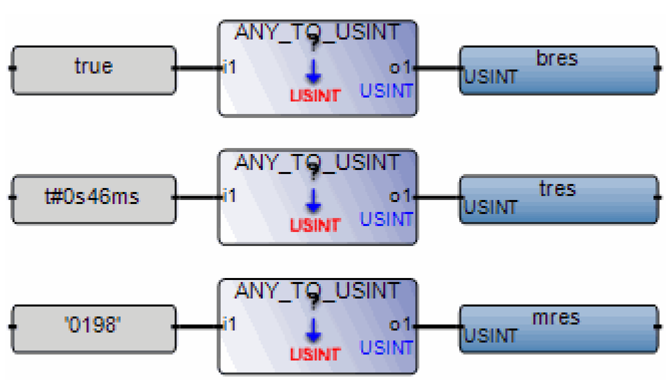
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | USINT | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

Description:

Converts variables to 8-bit unsigned short integer variables

### Example

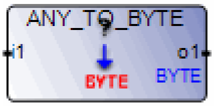
(* FBD example with "Convert to Unsigned Short Integer" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_USINT (true);          (* bres is 1 *)
tres := ANY_TO_USINT (t#0s46ms);      (* tres is 46 *)
mres := ANY_TO_USINT ('0198');        (* mres is 198 *)
```

# ANY_TO_BYTE




Arguments:

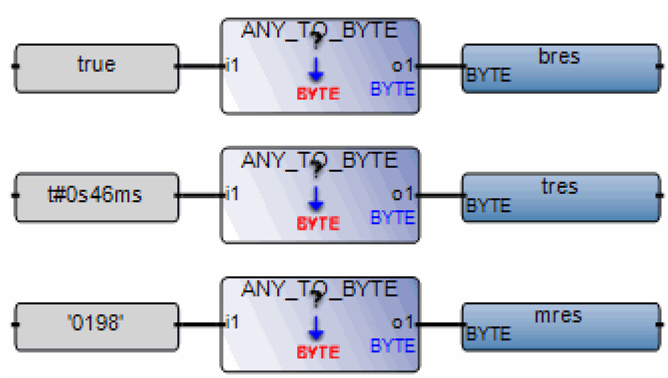
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | BYTE | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

Description:

Converts variables to 8-bit BYTE variables

## Example

(* FBD example with "Convert to BYTE" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_BYTE (true);           (* bres is 1 *)
tres := ANY_TO_BYTE (t#0s46ms);       (* tres is 46 *)
mres := ANY_TO_BYTE ('0198');         (* mres is 198 *)
```

# ANY_TO_INT


Arguments:

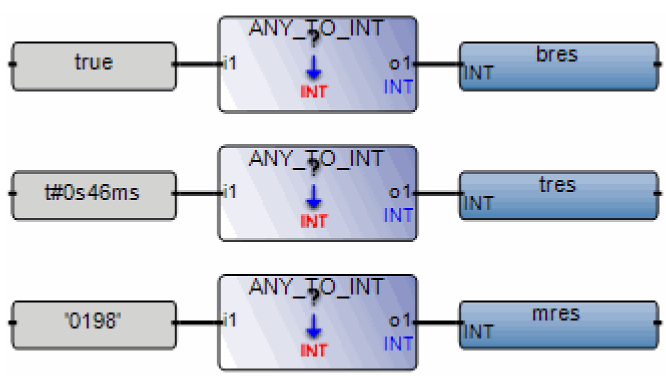
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | INT | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

Description:

Converts variables to 16-bit integer variables

## Example

(* FBD example with "Convert to Integer" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_INT (true);            (* bres is 1 *)
tres := ANY_TO_INT (t#0s46ms);        (* tres is 46 *)
mres := ANY_TO_INT ('0198');          (* mres is 198 *)
```

# ANY_TO_UINT


Arguments:

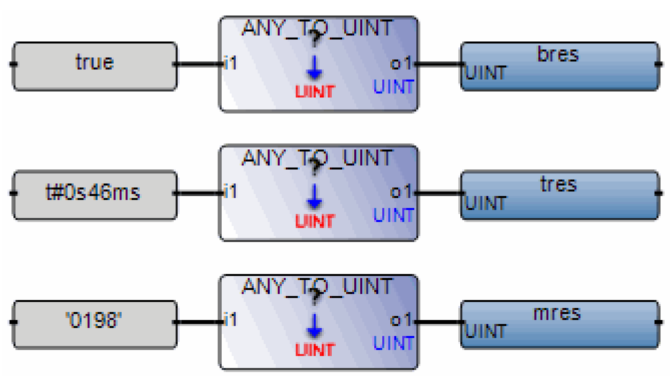
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | UINT | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

Description:

Converts variables to 16-bit unsigned integer variables

### Example

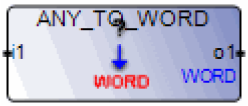
(* FBD example with "Convert to Unsigned Integer" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_UINT (true);          (* bres is 1 *)
tres := ANY_TO_UINT (t#0s46ms);      (* tres is 46 *)
mres := ANY_TO_UINT ('0198');        (* mres is 198 *)
```

# ANY_TO_WORD




Arguments:

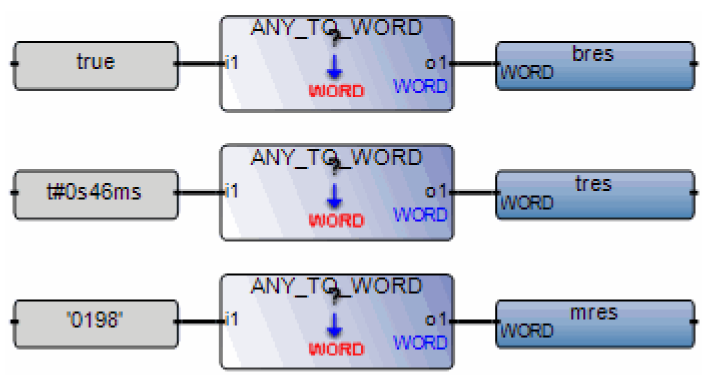
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value<br>0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |
| o1 | WORD | |

Description:

Converts variables to 16-bit WORD variables

### Example

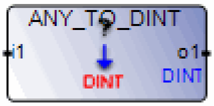
(* FBD example with "Convert to WORD" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_WORD (true);            (* bres is 1 *)
tres := ANY_TO_WORD (t#0s46ms);        (* tres is 46 *)
mres := ANY_TO_WORD ('0198');          (* mres is 198 *)
```

# ANY_TO_DINT




Arguments:

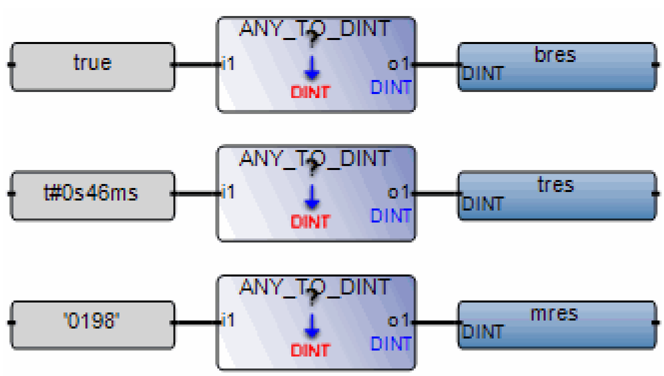
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | DINT | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

Description:

Converts variables to 32-bit double integer variables

### Example

(* FBD example with "Convert to Double Integer" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_DINT (true);            (* bres is 1 *)
tres := ANY_TO_DINT (t#1s46ms);        (* tres is 1046 *)
mres := ANY_TO_DINT ('0198');          (* mres is 198 *)
```

# ANY_TO_UDINT


Arguments:

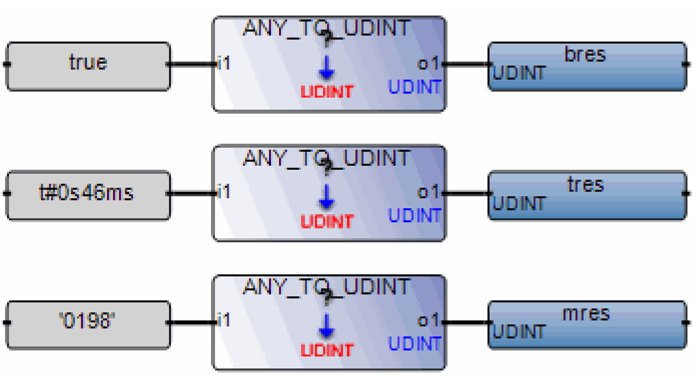
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | UDINT | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

Description:

Converts variables to 32-bit unsigned double integer variables

### Example

(* FBD example with "Convert to Unsigned Double Integer" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_UDINT (true);          (* bres is 1 *)
tres := ANY_TO_UDINT (t#1s46ms);      (* tres is 1046 *)
mres := ANY_TO_UDINT ('0198');        (* mres is 198 *)
```

# ANY_TO_DWORD


Arguments:

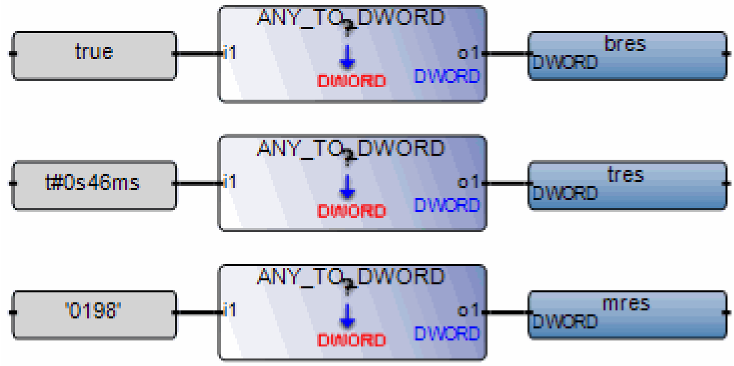
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | DWORD | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

Description:

Convert variables to 32-bit double WORD variables

### Example

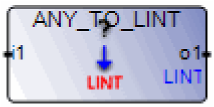
(* FBD example with "Convert to Double WORD" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_DWORD (true);            (* bres is 1 *)
tres := ANY_TO_DWORD (t#1s46ms);        (* tres is 1046 *)
mres := ANY_TO_DWORD ('0198');          (* mres is 198 *)
```

# ANY_TO_LINT




Arguments:

| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | LINT | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

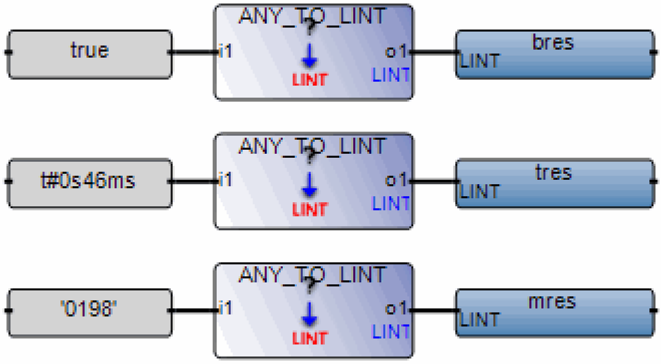
Description:

Converts variables to 64-bit long integer variables.

**Note:** The maximum value for a REAL or LREAL input must be less than 9.2233720e+18. For input values greater than this maximum, the output value is determined by the target type. For Windows and Linux targets, the output value will be reset to zero when the input value is greater than 9.2233720e+18. While for QNX targets, the output value will go into overflow.

## Example

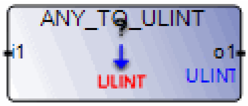
(* FBD example with "Convert to Long Integer" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_LINT (true);             (* bres is 1 *)
tres := ANY_TO_LINT (t#0s46ms);         (* tres is 46 *)
mres := ANY_TO_LINT ('0198');           (* mres is 198 *)
```

# ANY_TO_ULINT




Arguments:

| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | ULINT | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for real<br>decimal number represented by a string |

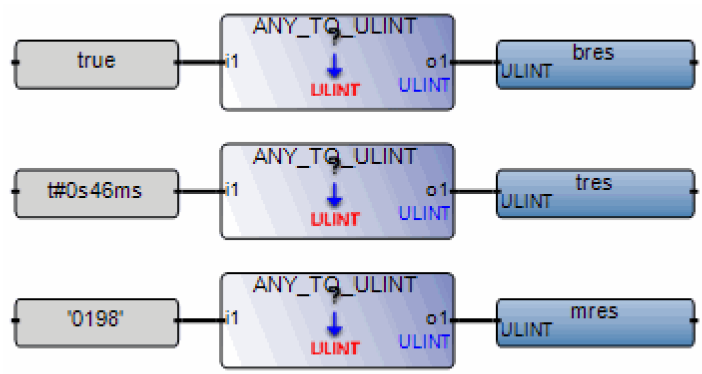
Description:

Converts variables to 64-bit unsigned long integer variable.

**Note:** The maximum value for a REAL or LREAL input must be less than 1.8446744e+19. For input values greater than this maximum, the output value is determined by the target type. For Windows and Linux targets, the output value will be reset to zero when the input value is greater than 1.8446744e+19. While for QNX targets, the output value will go into overflow.

### Example

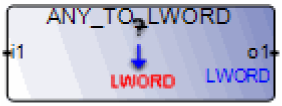
(* FBD example with "Convert to Unsigned Long Integer" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_ULINT (true);          (* bres is 1 *)
tres := ANY_TO_ULINT (t#0s46ms);      (* tres is 46 *)
mres := ANY_TO_ULINT ('0198');        (* mres is 198 *)
```

# ANY_TO_LWORD




Arguments:

| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | LWORD | 0 if i1 is FALSE / 1 if i1 is TRUE<br>number of milliseconds for a timer<br>integer part for a real<br>decimal number represented by a string |

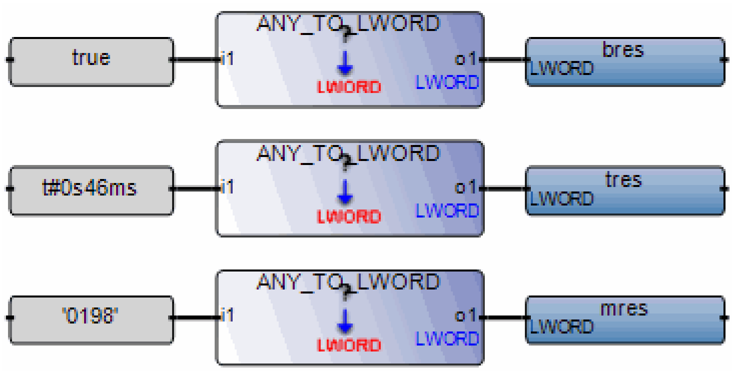
Description:

Converts variables to 64-bit long WORD variables.

**Note:** The maximum value for a REAL or LREAL input must be less than 1.8446744e+19. For input values greater than this maximum, the output value is determined by the target type. For Windows and Linux targets, the output value will be reset to zero when the input value is greater than 1.8446744e+19. While for QNX targets, the output value will go into overflow.

## Example

(* FBD example with "Convert to Long Word" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_LWORD (true);          (* bres is 1 *)
tres := ANY_TO_LWORD (t#0s46ms);      (* tres is 46 *)
mres := ANY_TO_LWORD ('0198');        (* mres is 198 *)
```

# ANY_TO_REAL


Arguments:

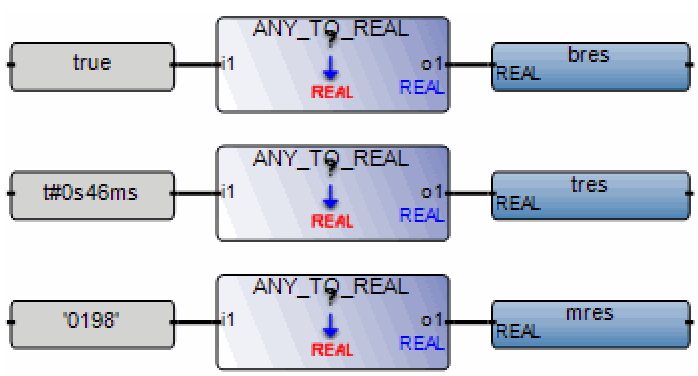
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | REAL | 0.0 if i1 is FALSE / 1.0 if i1 is TRUE<br>number of milliseconds for a timer<br>equivalent number for integer |

Description:

Converts variables to REAL variables

## Example

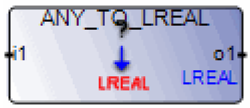
(* FBD example with "Convert to Real" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_REAL (true);             (* bres is 1.0 *)
tres := ANY_TO_REAL (t#1s46ms);         (* tres is 1046.0 *)
ares := ANY_TO_REAL (198);              (* ares is 198.0 *)
```

# ANY_TO_LREAL




Arguments:

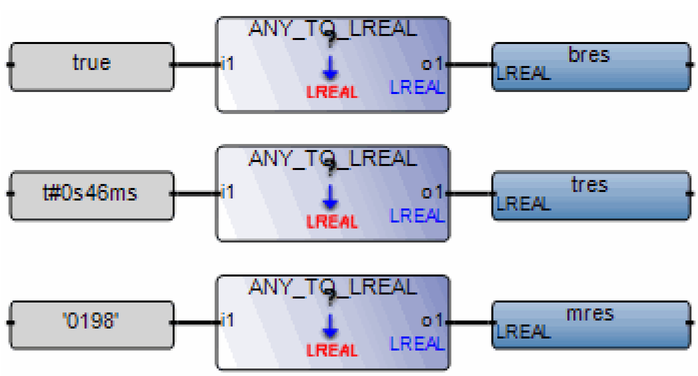
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | LREAL | 0.0 if i1 is FALSE / 1.0 if i1 is TRUE<br>number of milliseconds for a timer<br>equivalent number for integer |

Description:

Converts any variable to a long REAL variable

### Example

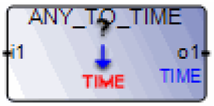
(* FBD example with "Convert to Long REAL" Operators *)




(* ST Equivalence: *)

```
bres := ANY_TO_LREAL (true);     (* bres is 1.0 *)
tres := ANY_TO_LREAL (t#1s46ms); (* tres is 1046.0 *)
ares := ANY_TO_LREAL (198);      (* ares is 198.0 *)
```

# ANY_TO_TIME




Arguments:

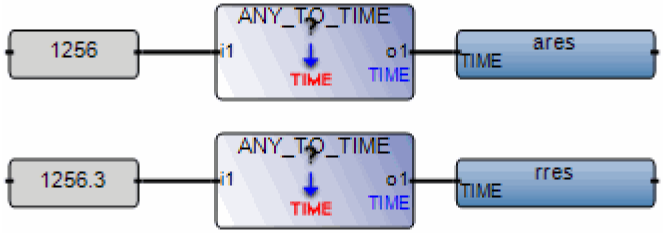
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value<br>i1 (or integer part of i1 if it is real) is the number of milliseconds<br>STRING (number of milliseconds, for example, a value of 300032 represents 5 minutes and 32 milliseconds) |
| o1 | TIME | time value represented by i1. A value of 1193h2m47s295ms indicates an invalid time. |

Description:

Converts variables to TIME variables, except for TIME and DATE variables. The SUB_DATE_DATE function enables the conversion of a DATE to TIME format.

### Example

(* FBD example with "Convert to Timer" Operators *)




(* ST Equivalence: *)

```
ares := ANY_TO_TIME (1256);          (* ares := t#1s256ms *)
rres := ANY_TO_TIME (1256.3);        (* rres := t#1s256ms *)
```

# ANY_TO_DATE


Arguments:

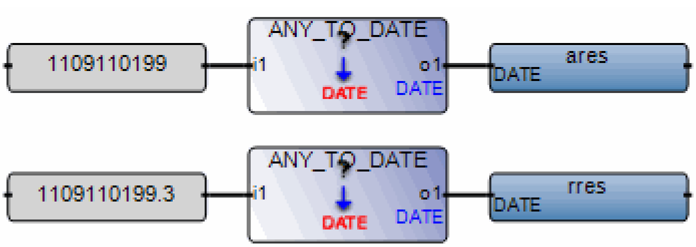
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | DATE | date represented by i1. A value of -1 indicates an invalid date. |

Description:

Converts variables to DATE variables. A 32-bit variable, providing the number of seconds since Jan 1, 1970, based on the time_t data type.

### Example

(* FBD example with "Convert to DATE" Operators *)




(* ST Equivalence: *)

```
ares := ANY_TO_DATE (1109110199);     (* ares := d#2005-02-22 *)
rres := ANY_TO_DATE (1109110199.3);   (*rres := d#2005-02-22 *)
```

# ANY_TO_STRING


Arguments:

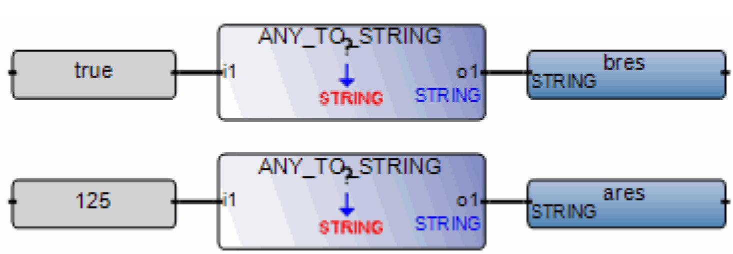
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Any value |
| o1 | STRING | If i1 is a Boolean, 'FALSE' or 'TRUE'<br>If i1 is an integer or a real, decimal representation<br>If i1 is a TIME:<br>TIME time1<br>STRING s1<br>time1 :=13 ms;<br>s1 :=ANY_TO_STRING(time1);<br>(* s1 = '0s13' *) |

Description:

Converts variables to STRING variables

### Example

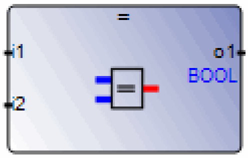
(* FBD example with "Convert to STRING" Operators *)



(* ST Equivalence: *)

```
bres := ANY_TO_STRING (TRUE);    (* bres is 'TRUE' *)
ares := ANY_TO_STRING (125);     (* ares is '125' *)
```

# Equal




Arguments:

| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Both inputs must have the same format. |
| i2 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | |
| o1 | BOOL | TRUE if i1 = i2 |

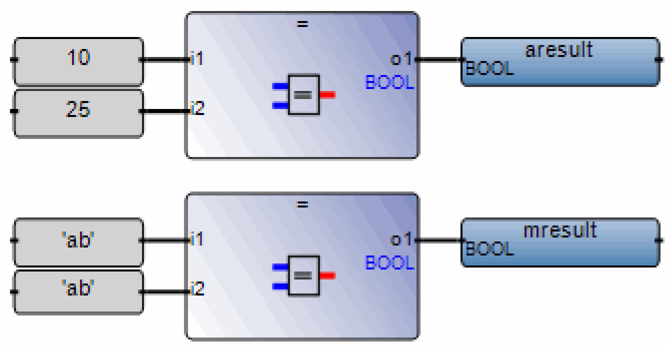
Description

For integer, REAL, TIME, DATE, and STRING variables, compares the first input to the second to determine equality.

For TON, TP, TOF, BLINK, and StepName.t in SFC chart, equality testing of TIME variables is not recommended.

### Example

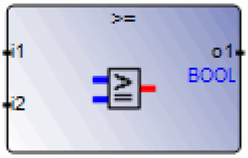
(* FBD example with "Is Equal to" Operators *)




(* ST Equivalence: *)

```
aresult := (10 = 25); (* aresult is FALSE *)
mresult := ('ab' = 'ab'); (* mresult is TRUE *)
```

# Greater Than or Equal




Arguments:

| | | |
|---|---|---|
| i1 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Both inputs must have the same type. |
| i2 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | |
| o1 | BOOL | TRUE if i1 >= i2 |

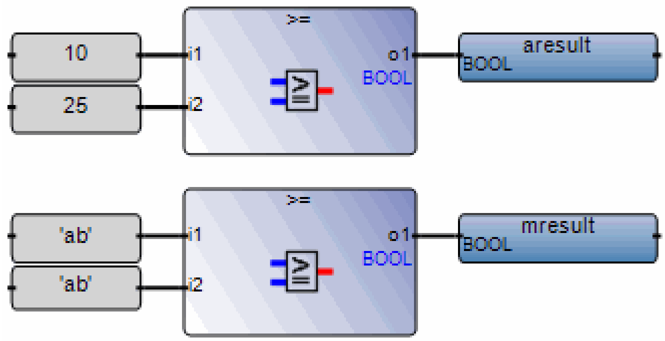
Description:

For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is greater than or equal to the second.

For TON, TP, TOF, BLINK, and StepName.t in SFC chart, equality testing of TIME variables is not recommended.

### Example

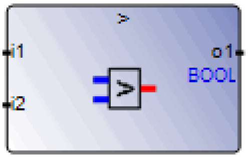
(* FBD example with "Greater or Equal to" Operators *)




(* ST Equivalence: *)

```
aresult := (10 >= 25); (* aresult is FALSE *)

mresult := ('ab' >= 'ab'); (* mresult is TRUE *)
```

# Greater Than




Arguments:

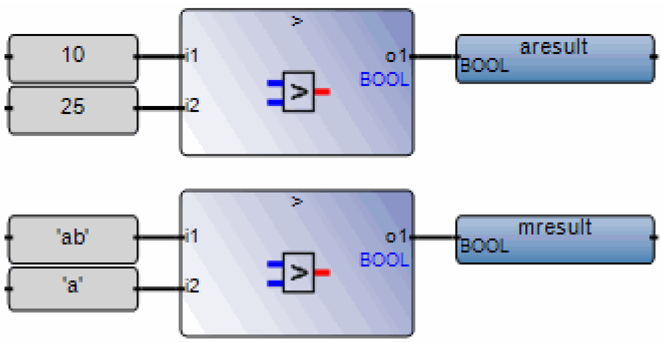
| | | |
|---|---|---|
| i1 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Both inputs must have the same type |
| i2 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | |
| o1 | BOOL | TRUE if i1 > i2 |

Description:

For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is greater than the second.

### Example

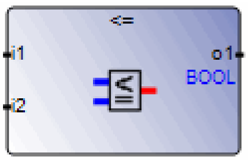
(* FBD example with "Greater than" Operators *)




(* ST Equivalence: *)

```
aresult := (10 > 25); (* aresult is FALSE *)
mresult := ('ab' > 'a'); (* mresult is TRUE *)
```

# Less Than or Equal




Arguments:

| | | |
|---|---|---|
| i1 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Both inputs must have the same type. |
| i2 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | |
| o1 | BOOL | TRUE if i1 <= i2 |

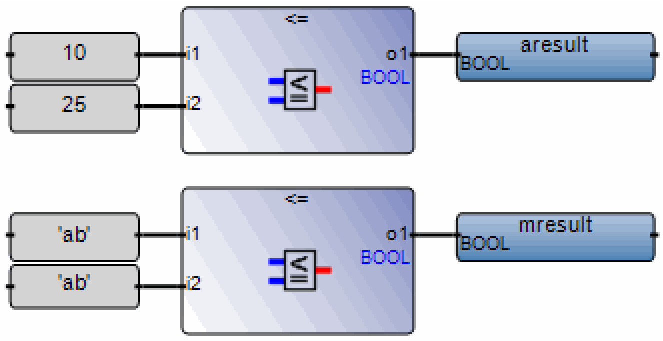
Description:

For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is less than or equal to the second.

For TON, TP, TOF, BLINK, and StepName.t in SFC chart, equality testing of TIME variables is not recommended.

### Example

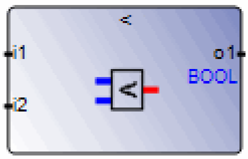
(* FBD example with "Less or Equal to" Operators *)




(* ST Equivalence: *)

```
aresult := (10 <= 25); (* aresult is TRUE *)

mresult := ('ab' <= 'ab'); (* mresult is TRUE *)
```

# Less Than




Arguments:

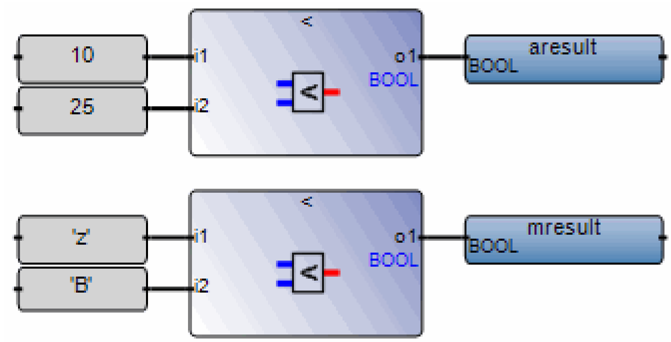
| | | |
|---|---|---|
| i1 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | Both inputs must have the same type |
| i2 | SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | |
| o1 | BOOL | TRUE if i1i2 < i2 |

Description:

For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is less than the second.

### Example

(* FBD example with "Less than" Operators *)




(* ST Equivalence: *)

```
aresult := (10 < 25); (* aresult is TRUE *)

mresult := ('z' < 'B'); (* mresult is FALSE *)
```

# NEG


Arguments:

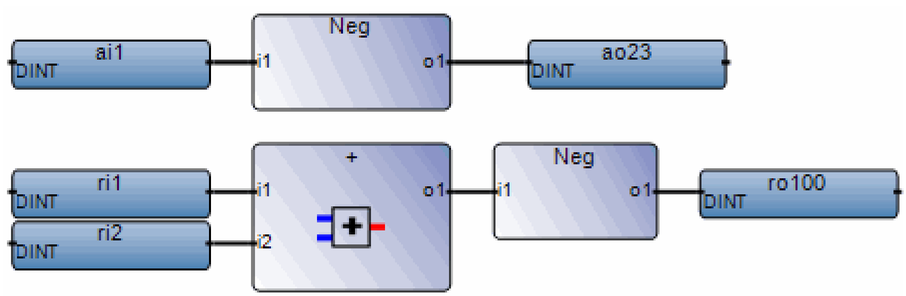
| | | |
|---|---|---|
| i1 | SINT - INT - DINT - LINT - REAL - LREAL | Input and output must have the same format |
| o1 | SINT - INT - DINT - LINT - REAL - LREAL | |

Description:

Converts variables to negated variables

### Example

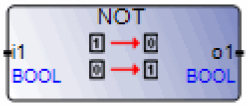
(* FBD example with Negation Operators *)




(* ST equivalence: *)

```
ao23 := - (ai10);
ro100 := - (ri1 + ri2);
```

# NOT




Arguments:

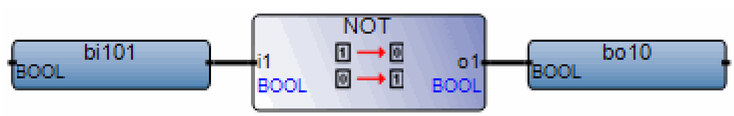
| | |
|---|---|
| i1 | Any Boolean variable or complex expression |
| o1 | TRUE when i1 is FALSE<br>FALSE when i1 is TRUE |

Description:

For Boolean expressions, converts variables to negated variables.

## Example

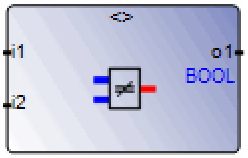
(* FBD example with "NOT" Operator *)




(* ST equivalence: *)

```
bo10 := NOT bi101;
```

# Not Equal




Arguments:

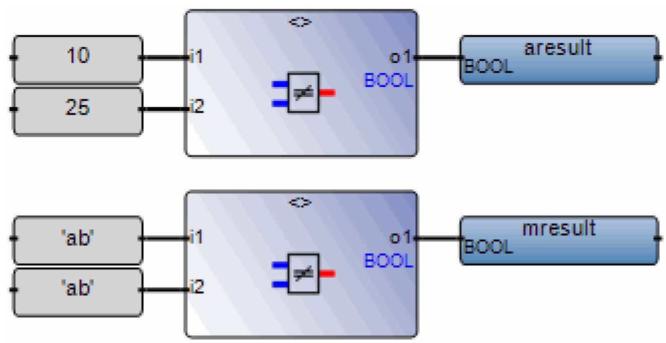
| | | |
|---|---|---|
| i1 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | both inputs must have the same type |
| i2 | BOOL - SINT - USINT - BYTE - INT - UINT - WORD - DINT - UDINT - DWORD - LINT - ULINT - LWORD - REAL - LREAL - TIME - DATE - STRING | |
| o1 | BOOL | TRUE if first <> second |

Description:

For integer, REAL, TIME, DATE, and STRING variables, compares input variables to determine whether the first is not equal to the second.

### Example

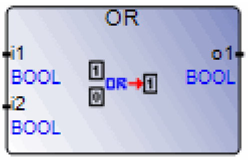
(* FBD example with "Is Not Equal to" Operators *)




(* ST Equivalence: *)

```
aresult := (10 <> 25); (* aresult is TRUE *)

mresult := ('ab' <> 'ab'); (* mresult is FALSE *)
```

# OR




**Note:** The creation of additional inputs is supported.

Arguments:

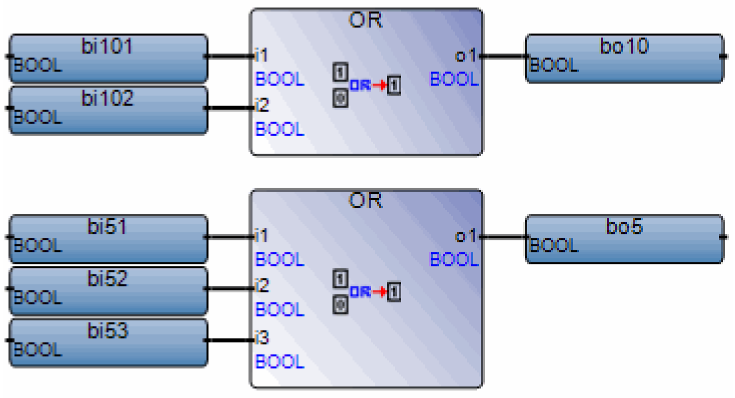
| | | |
|---|---|---|
| (inputs) | BOOL | |
| output | BOOL | Boolean **OR** of the input terms |

Description:

Boolean OR of two or more variables

## Example

(* FBD example with "OR" Operators *)

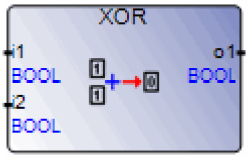



(* ST equivalence: *)

```
bo10 := bi101 OR NOT (bi102);
```

```
bo5 := (bi51 OR bi52) OR bi53;
```

# XOR




Arguments:

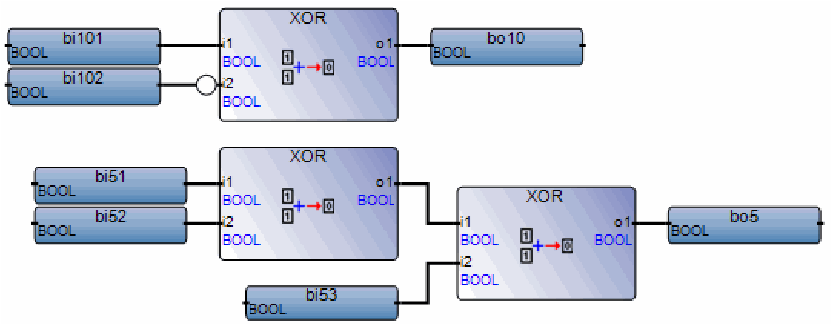
| | | |
|---|---|---|
| i1 | BOOL | |
| i2 | BOOL | |
| o1 | BOOL | Boolean **exclusive OR** of the two input terms |

Description:

Boolean exclusive OR of two variables

## Example

(* FBD example with "XOR" operators *)




(* ST equivalence: *)

```
bo10 := bi101 XOR NOT (bi102);

bo5 := (bi51 XOR bi52) XOR bi53;
```

# Functions

The following are the functions supported by the system:

| | | |
|---|---|---|
| **Arithmetic Operations** | ABS | Absolute value of a REAL value |
| | EXPT, POW | Exponent, power calculation of REAL values |
| | LOG | Logarithm of a REAL value |
| | MOD | Modulo |
| | SQRT | Square root of a REAL value |
| | RAND | Random value |
| | TRUNC | Truncate decimal part of a REAL value |
| | ACOS, ASIN, ATAN | Arc cosine, Arc sine, Arc tangent of a REAL value |
| | COS, SIN, TAN | Cosine, Sine, Tangent of a REAL value |
| **Binary Operations** | AND_MASK | Integer bit-to-bit AND mask |
| | OR_MASK | Integer bit-to-bit OR mask |
| | XOR_MASK | Integer bit-to-bit Exclusive OR mask |
| | NOT_MASK | Integer bit-to-bit negation |
| | ROL, ROR | Rotate Left, Rotate Right an integer value |
| | SHL, SHR | Shift Left, Shift Right an integer value |
| **Boolean Operations** | ODD | Odd parity |
| **Process Control** | MIN, MAX, LIMIT | Minimum, Maximum, Limit |
| | MUX4, MUX8 | Multiplexer (4 or 8 entries) |
| | SEL | Binary selector |

| | | |
|---|---|---|
| **String Manipulation** | ASCII | Character -> ASCII code |
| | CHAR | ASCII code -> Character |
| | MLEN | Get string length |
| | DELETE, INSERT | Delete sub-string, Insert string |
| | FIND, REPLACE | Find sub-string, Replace sub-string |
| | LEFT, MID, RIGHT | Extract left, middle or right of a string |
| **System Operations** | LOCK_CPU, UNLOCK_CPU | Lock data space, unlock data space |
| **Time Operations** | CURRENT_ISA_DATE | Gets the current date |
| | SUB_DATE_DATE | Compares two dates and provides the difference in TIME format |

# ABS


Arguments:

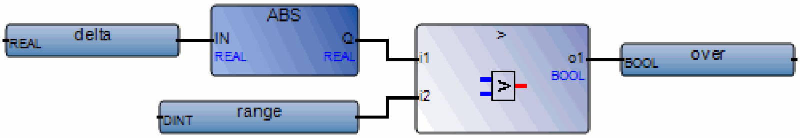
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any signed real value |
| ABS | Q | REAL | Absolute value (always positive) |

Description:

Yields the absolute (positive) value of a REAL value.

## Example

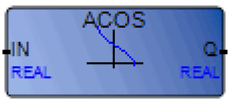
(* FBD Program using "ABS" Function *)




(* ST Equivalence: *)

```
over := (ABS (delta) > range);
```

# ACOS




Arguments:

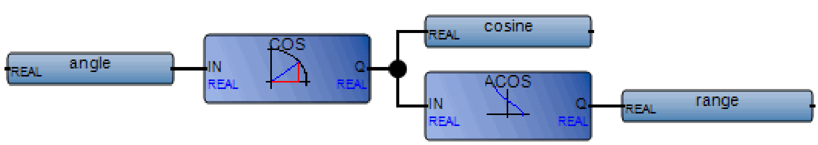
| | | | |
|---|---|---|---|
| IN | IN | REAL | Must be in set [-1.0 .. +1.0] |
| ACOS | Q | REAL | Arc-cosine of the input value (in set [0.0 .. PI])<br>= 0.0 for invalid input |

Description:

Yields the Arc Cosine of a REAL value. Input and output values are in radians.

## Example

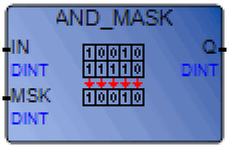
(* FBD Program using "COS" and "ACOS" Functions *)




(* ST Equivalence: *)

```
cosine := COS (angle);
result := ACOS (cosine); (* result is equal to angle *)
```

# AND_MASK




Arguments:

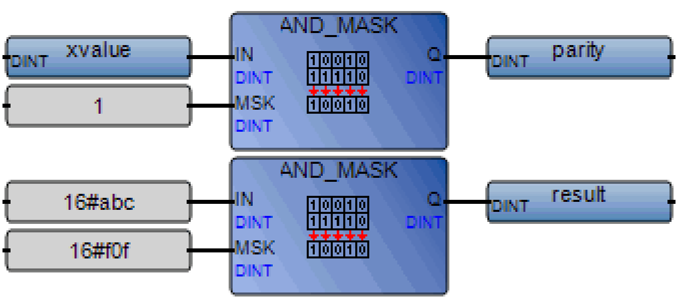
| | | | |
|---|---|---|---|
| IN | IN | DINT | Must have integer format |
| MSK | MSK | DINT | Must have integer format |
| AND_MASK | Q | DINT | Bit-to-bit logical AND between IN and MSK |

Description:

Integer AND bit-to-bit mask.

## Example

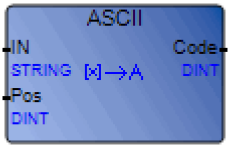
(* FBD example with AND_MASK Operators *)




(* ST Equivalence: *)

```
parity := AND_MASK (xvalue, 1);  (* 1 if xvalue is odd *)
result := AND_MASK (16#abc, 16#f0f); (* equals 16#a0c *)
```

# ASCII




Arguments:

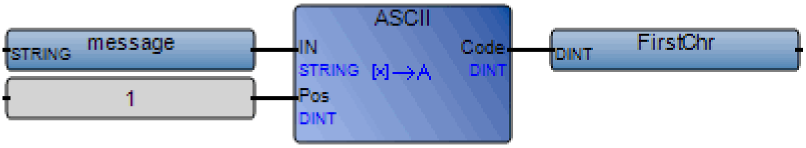
| | | | |
|---|---|---|---|
| IN | IN | STRING | Any non-empty string |
| Pos | Pos | DINT | Position of the selected character in set [1.. len] (len is the length of the IN string) |
| ASCII | Code | DINT | Code of the selected character (in set [0 .. 255]) yields 0 is Pos is out of the string |

Description:

Yields the ASCII code for characters in strings.

## Example

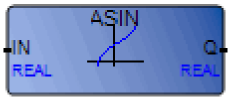
(* FBD Program using "ASCII" Function *)




(* ST Equivalence: *)

```
FirstChr := ASCII (message, 1);
(* FirstChr is the ASCII code of the first character of the string *)
```

## ASIN




Arguments:

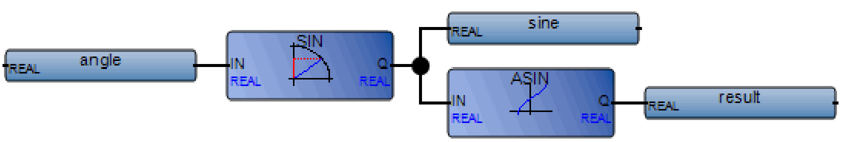
| | | | |
|---|---|---|---|
| IN | IN | REAL | Must be in set [-1.0 .. +1.0] |
| ASIN | Q | REAL | Arc-sine of the input value (in set [-PI/2 .. +PI/2])<br>= 0.0 for invalid input |

Description:

Yields the Arc Sine of a REAL value.

### Example

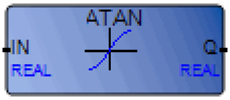
(* FBD Program using "SIN" and "ASIN" Functions *)




(* ST Equivalence: *)

```
sine := SIN (angle);
result := ASIN (sine); (* result is equal to angle *)
```

# ATAN




Arguments:

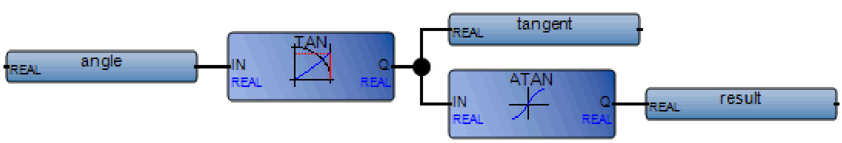
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any real value |
| ATAN | Q | REAL | Arc-tangent of the input value (in set [-PI/2 .. +PI/2])<br>= 0.0 for invalid input |

Description:

Yields the Arc Tangent of a REAL value.

## Example

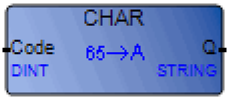
(* FBD Program using "TAN" and "ATAN" Function *)




(* ST Equivalence: *)

```
tangent := TAN (angle);
result := ATAN (tangent); (* result is equal to angle*)
```

# CHAR




Arguments:

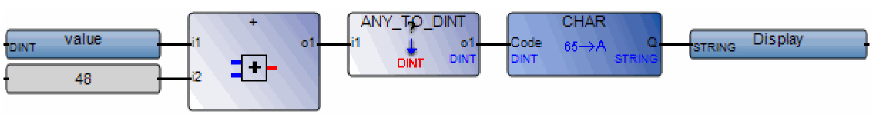
| | | | |
|---|---|---|---|
| Code | Code | DINT | Code in set [0 .. 255] |
| CHAR | Q | STRING | One character string<br>the character has the ASCII code given in input Code<br>(ASCII code is used modulo 256) |

Description:

For a given ASCII code, provides a string containing one character.

## Example

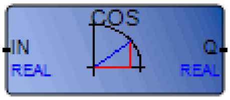
(* FBD Program using "CHAR" Function *)




(* ST Equivalence: *)

```
Display := CHAR ( value + 48 );
(* value is in set [0..9] *)
(* 48 is the ascii code of '0' *)
(* result is one character string from '0' to '9' *)
```

# COS




Arguments:

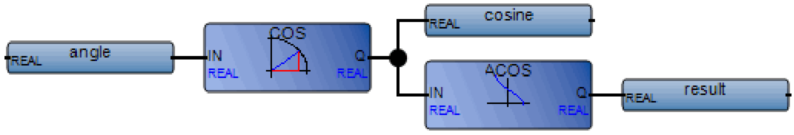
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any REAL value |
| COS | Q | REAL | Cosine of the input value (in set [-1.0 .. +1.0]) |

Description:

Yields the Cosine of a REAL value.

## Example

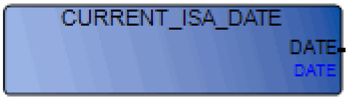
(* FBD Program using "COS" and "ACOS" Functions *)




(* ST Equivalence: *)

```
cosine := COS (angle);
result := ACOS (cosine); (* result is equal to angle *)
```

# CURRENT_ISA_DATE




Arguments:

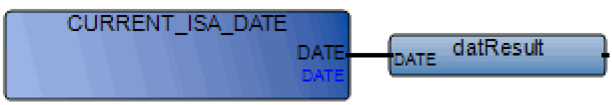
| | | | |
|---|---|---|---|
| CURRENT_ISA_DATE | DATE | DATE | The current date |

Description:

Retrieves the current date.

## Example

(* FBD Program using "CURRENT_ISA_DATE" Function *)




(* ST Equivalence: *)

```
datResult := CURRENT_ISA_DATE();
```

# DELETE


Arguments:

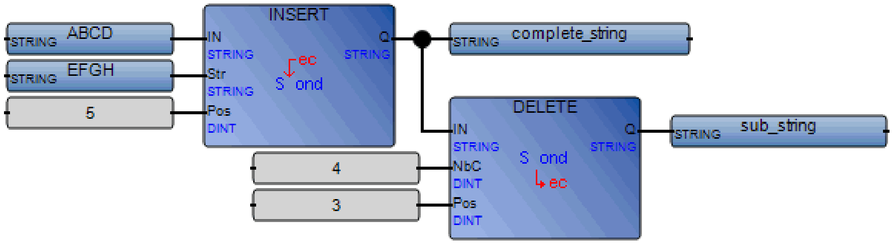
| | | | |
|---|---|---|---|
| IN | IN | STRING | Any non-empty string |
| NbC | NbC | DINT | Number of characters to be deleted |
| Pos | Pos | DINT | Position of the first deleted character<br>(first character of the string has position 1) |
| DELETE | Q | STRING | modified string<br>empty string if Pos < 1<br>initial string if Pos > IN string length<br>initial string if NbC <= 0 |

Description:

Deletes part of a string.

### Example

(* FBD Program using "DELETE" Function *)




(* ST Equivalence: *)

```
complete_string := INSERT ('ABCD ', 'EFGH', 5); (* complete_string is
'ABCDEFGH ' *)
```

```
sub_string := DELETE (complete_string, 4, 3); (* sub_string is
'ABGH '*)
```

# EXPT


Arguments:

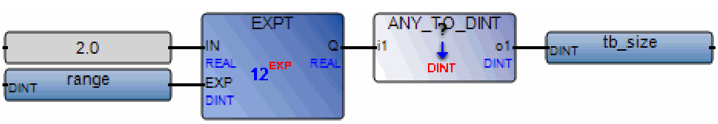
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any signed real value |
| EXP | EXP | DINT | Integer exponent |
| EXPT | Q | REAL | ($IN^{EXP}$) |

Description:

Where 'base' is the first argument and 'exponent' is the second argument, yields the REAL result of the following operation: ($base^{exponent}$).

## Example

(* FBD Program using "EXPT" Function *)




(* ST Equivalence: *)

```
tb_size := ANY_TO_DINT (EXPT (2.0, range) );
```

# FIND


Arguments:

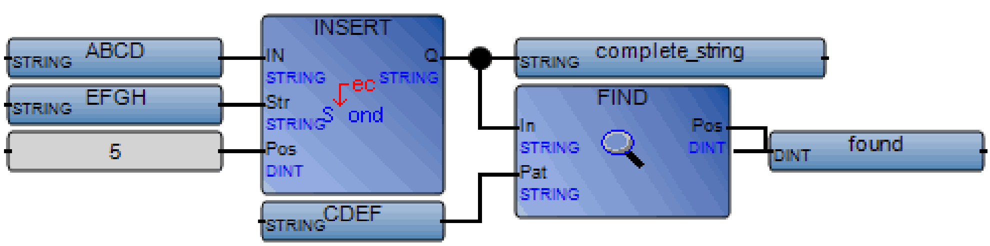
| | | | |
|---|---|---|---|
| In | In | STRING | Any string |
| Pat | Pat | STRING | Any non-empty string (Pattern) |
| FIND | Pos | DINT | = 0 if sub string Pat not found<br>= position of the first character of the first occurrence of the sub-string Pat<br>(first position is 1)<br>this function is case sensitive |

Description:

Locates and provides the position of sub-strings within strings.

## Example

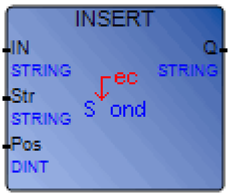
(* FBD Program using "FIND" Function *)




(* ST Equivalence: *)

```
complete_string := 'ABCD' + 'EFGH'; (* complete_string is 'ABCDEFGH ' *)
found := FIND (complete_string, 'CDEF'); (* found is 3 *)
```

# INSERT




Arguments:

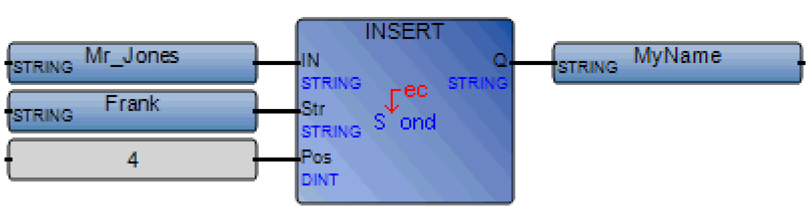
| | | | |
|---|---|---|---|
| IN | IN | STRING | Initial string |
| Str | Str | STRING | String to be inserted |
| Pos | Pos | DINT | Position of the insertion<br>the insertion is done before the position<br>(first valid position is 1) |
| INSERT | Q | STRING | Modified string<br>empty string if Pos <= 0<br>concatenation of both strings if Pos is greater than the length of the IN string |

Description:

Inserts sub-strings at user-defined positions within strings.

### Example

(* FBD Program using "INSERT" Function*)



(* ST Equivalence: *)

```
MyName := INSERT ('Mr JONES', 'Frank ', 4);

(* MyName is 'Mr Frank JONES' *)
```

# LEFT


Arguments:

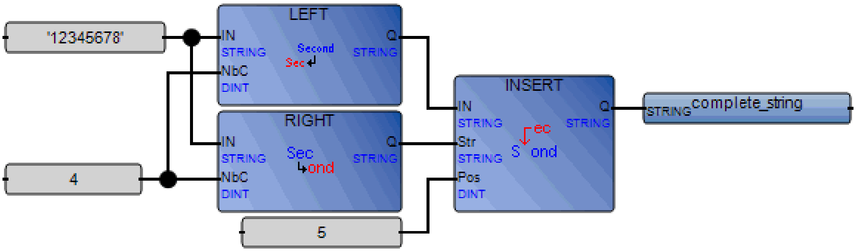
| | | | |
|---|---|---|---|
| IN | IN | STRING | Any non-empty string |
| NbC | NbC | DINT | Number of characters to be extracted. This number cannot be greater than the length of the IN string. |
| LEFT | Q | STRING | Left part of the IN string (its length = NbC)<br>empty string if NbC <= 0<br>complete IN string if NbC >= IN string length |

Description:

From the left end of strings, yields the number of characters defined.

### Example

(* FBD Program using "LEFT" and "RIGHT" Functions *)




(* ST Equivalence: *)

```
complete_string := INSERT (RIGHT ('12345678', 4), LEFT ('12345678', 4),
5);
```

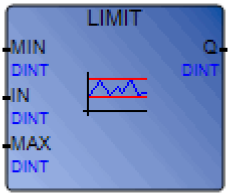
(* complete_string is '56781234'

the value issued from RIGHT call is '5678'

the value issued from LEFT call is '1234'

*)

## LIMIT




Arguments:

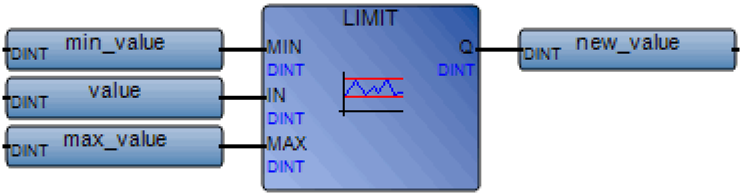
| | | | |
|---|---|---|---|
| MIN | MIN | DINT | Minimum value allowed |
| IN | IN | DINT | Any signed integer value |
| MAX | MAX | DINT | Maximum value allowed |
| LIMIT | Q | DINT | Input value restricted to the allowed range |

Description:

Restricts integer values to a given interval. Integer values between the minimum and maximum are unchanged. Integer values greater than the maximum are replaced with the maximum value. Integer values less than the minimum are replaced with the minimum value.

### Example

(* FBD Program using "LIMIT" Function *)




(* ST Equivalence: *)

```
new_value := LIMIT (min_value, value, max_value);
```

(* bounds the value to the [min_value..max_value] set *)

# LOCK_CPU


Arguments:

| | | | |
|---|---|---|---|
| TMOT | TMOT | TIME | Maximum time delay to lock the data space, in milliseconds. If the data space is unavailable, i.e. locked, during this time period, the lock operation fails. |
| OK | OK | BOOL | Status of the thread lock operation<br>TRUE=Successful lock operation<br>FALSE=Unsuccessful lock operation |

Description:

When using run-time supporting interrupts, each interrupt and main loop should use the LOCK_CPU to access variables shared between different execution paths. LOCK_CPU grants a POU exclusive access to the variables accessed after a call. The exclusive access is disabled once UNLOCK_CPU is called. If multiple POUs on distinct interrupts call LOCK_CPU at the same time, only one POU will run until UNLOCK_CPU is called. Afterwards, the next POU in the execution stack will run LOCK_CPU.

**Note:** Ensure LOCK_CPU is called once in a given thread before calling UNLOCK_CPU.

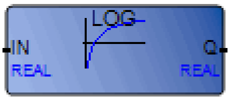
### Example

(* FBD Program using "LOCK_CPU" Function *)


(* ST Equivalence: *)

```
status1:=LOCK_CPU(T#8ms);
```

# LOG




Arguments:

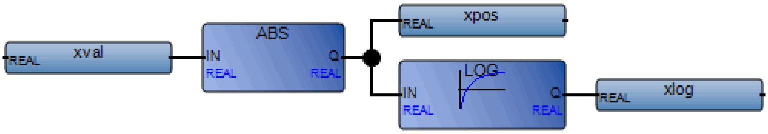
| | | | |
|---|---|---|---|
| IN | IN | REAL | Must be greater than zero |
| LOG | Q | REAL | Logarithm (base 10) of the input value |

Description:

Yields the logarithm (base 10) of a REAL value.

## Example

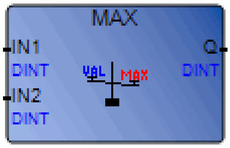
(* FBD Program using "LOG" Function *)




(* ST Equivalence: *)

```
xpos := ABS (xval);
xlog := LOG (xpos);
```

# MAX




Arguments:

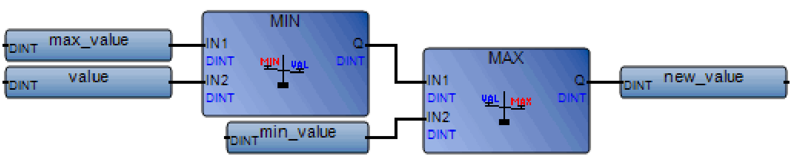
| | | | |
|---|---|---|---|
| IN1 | IN1 | DINT | Any signed integer value |
| IN2 | IN2 | DINT | (cannot be REAL) |
| MAX | Q | DINT | Maximum of both input values |

Description:

Yields the maximum of two integer values.

## Example

(* FBD Program using "MIN" and "MAX" Function *)




(* ST Equivalence: *)

```
new_value := MAX (MIN (max_value, value), min_value);
```

(* bounds the value to the [min_value..max_value] set *)

# MID


Arguments:

| | | | |
|---|---|---|---|
| IN | IN | STRING | Any non-empty string |
| NbC | NbC | DINT | Number of characters to extract (must be less than or equal to the length of the IN string) |
| Pos | Pos | DINT | Position of the sub-string<br>the sub-string first character is the one pointed to by Pos<br>(the first valid position is 1) |
| MID | Q | STRING | Middle part of the string (its length = NbC).<br>When the number of characters to extract exceeds the length of the IN string, NbC is automatically recalculated to get the remainder of the string only. When NbC or Pos are zero or negative numbers, an empty string is returned. |

Description:

Using the position and number of characters provided, yields required parts of strings.

### Example

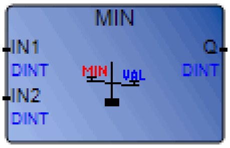
(* FBD Program using "MID" Function *)

(* ST Equivalence: *)

```
sub_string := MID ('abcdefgh', 2, 4);
```

(* sub_string is 'de' *)

# MIN




Arguments:

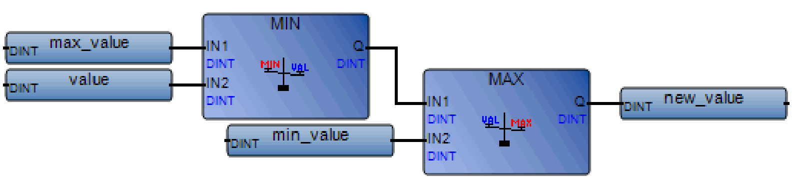
| | | | |
|---|---|---|---|
| IN1 | IN1 | DINT | Any signed integer value |
| IN2 | IN2 | DINT | (cannot be REAL) |
| MIN | Q | DINT | Minimum of both input values |

Description:

Yields the minimum of two integer values.

## Example

(* FBD Program using "MIN" and "MAX" Function *)




(* ST Equivalence: *)

```
new_value := MAX (MIN (max_value, value), min_value);
```

(* bounds the value to the [min_value..max_value] set *)

# MLEN


Arguments:

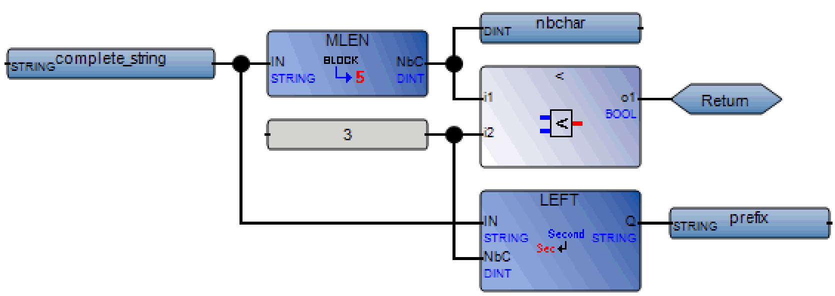
| | | | |
|---|---|---|---|
| IN | IN | STRING | Any string |
| MLEN | NbC | DINT | Number of characters in the IN string |

Description:

Yields the length of strings.

## Example

(* FBD Program using "MLEN" Function *)

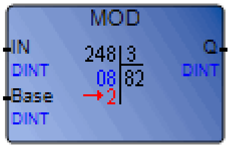



(* ST Equivalence: *)

```
nbchar := MLEN (complete_string);
If (nbchar < 3) Then Return; End_if;
prefix := LEFT (complete_string, 3);
```

(* this program extracts the 3 characters on the left of the string and put the result in the prefix string variable nothing is done if the string length is less than three characters *)

# MOD




Arguments:

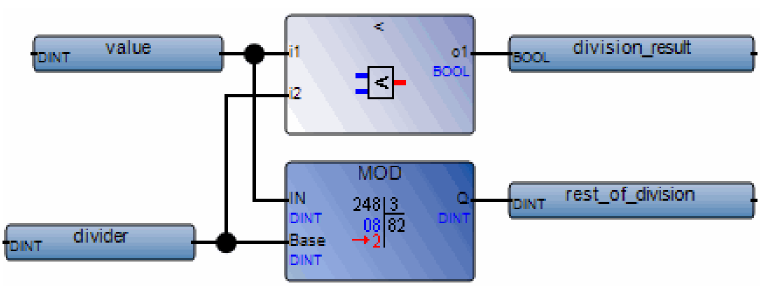
| | | | |
|---|---|---|---|
| IN | IN | DINT | Any signed INTEGER value |
| Base | Base | DINT | Must be greater than zero |
| MOD | Q | DINT | Modulo calculation (input MOD base) yields -1 if Base <= 0 |

Description:

Yields the modulo of an integer value.

## Example

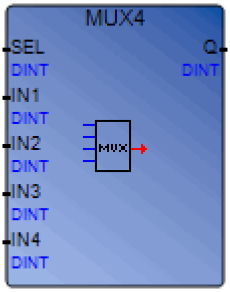
(* FBD Program using "MOD" Function *)




(* ST Equivalence: *)

```
division_result := (value / divider); (* integer division *)
rest_of_division := MOD (value, divider); (* rest of the division *)
```

# MUX4




Arguments:

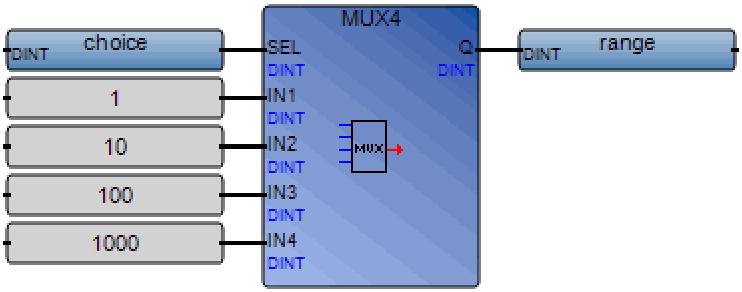
| | | | |
|---|---|---|---|
| SEL | SEL | DINT | Selector integer value (must be in set [0..3]) |
| IN1..IN4 | IN1..IN4 | DINT | Any integer values |
| MUX4 | Q | DINT | = value1 if SEL = 0<br>= value2 if SEL = 1<br>= value3 if SEL = 2<br>= value4 if SEL = 3<br>= 0 for all other values of the selector |

Description:

Yields a value between four integer values.

### Example

(* FBD Program using "MUX4" Function *)

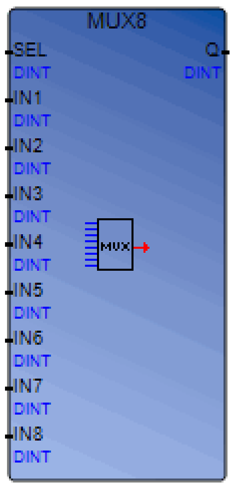



(* ST Equivalence: *)

```
range := MUX4 (choice, 1, 10, 100, 1000);
```

(* select from 4 predefined ranges, for example, if choice is 1, range will be 10 *)

# MUX8




Arguments:

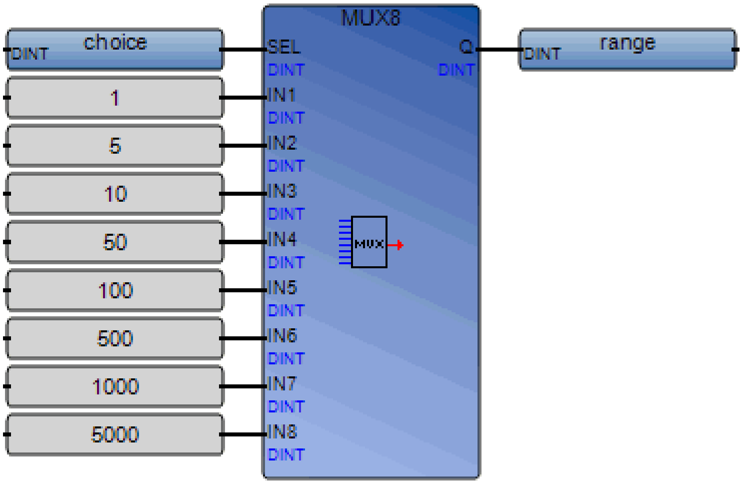
| | | | |
|---|---|---|---|
| SEL | SEL | DINT | Selector integer value (must be in set [0..7]) |
| IN1..IN8 | IN1..IN8 | DINT | Any integer values |
| MUX8 | Q | DINT | = value1 if selector = 0<br>= value2 if selector = 1<br>...<br>= value8 if selector = 7<br>= 0 for all other values of the selector |

Description:

Yields a value between eight integer values.

## Example

(* FBD Program using "MUX8" Function *)

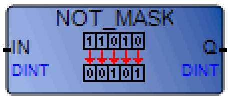



(* ST Equivalence: *)

```
range := MUX8 (choice, 1, 5, 10, 50, 100, 500, 1000, 5000);
```

(* select from 8 predefined ranges, for example, if choice is 3, range will be 50 *)

## NOT_MASK




Arguments:

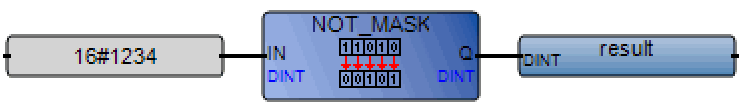
| | | | |
|---|---|---|---|
| IN | IN | DINT | Must have integer format |
| NOT_MASK | Q | DINT | Bit-to-bit negation on 32 bits of IN |

Description:

Integer bit-to-bit negation mask.

### Example

(* FBD example with NOT_MASK Operators *)




(*ST equivalence: *)

```
result := NOT_MASK (16#1234);
```

(* result is 16#FFFF_EDCB *)

# ODD


Arguments:

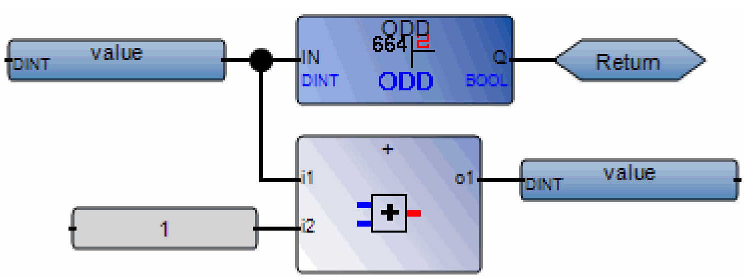
| | | | |
|---|---|---|---|
| IN | IN | DINT | Any signed integer value |
| Odd | Q | BOOL | TRUE if input value is odd<br>FALSE if input value is even |

Description:

Determines the parity of an integer, yielding an odd or even result.

## Example

(* FBD Program using "ODD" Function *)




(* ST Equivalence: *)

```
If Not (ODD (value)) Then Return; End_if;
value := value + 1;
```

(* makes value always even *)

# OR_MASK


Arguments:

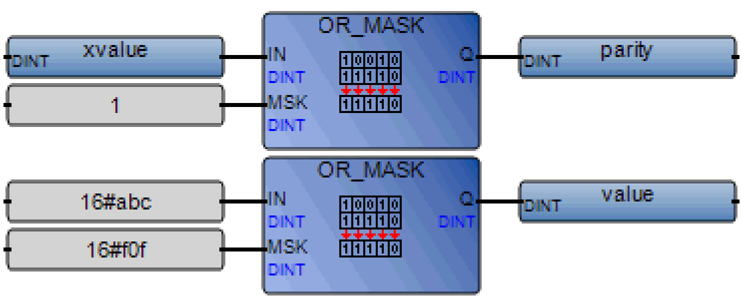
| | | | |
|---|---|---|---|
| IN | IN | DINT | Must have integer format |
| MSK | MSK | DINT | Must have integer format |
| OR_MASK | Q | DINT | Bit-to-bit logical **OR** between IN and MSK |

Description:

Integer OR bit-to-bit mask.

## Example

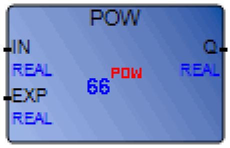
(* FBD example with OR_MASK Operators *)




(* ST Equivalence: *)

```
parity := OR_MASK (xvalue, 1); (* makes value always odd *)
result := OR_MASK (16#abc, 16#f0f); (* equals 16#fbf *)
```

# POW




Arguments:

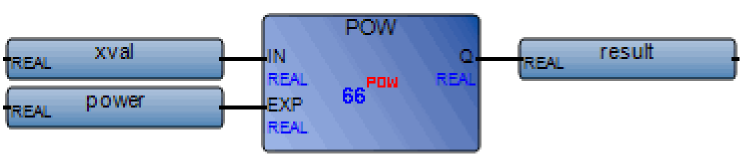
| | | | |
|---|---|---|---|
| IN | IN | REAL | REAL number to be raised |
| EXP | EXP | REAL | Power (exponent) |
| POW | Q | REAL | ($IN^{EXP}$)<br>1.0 if IN is not 0.0 and EXP is 0.0<br>0.0 if IN is 0.0 and EXP is negative<br>0.0 if both IN and EXP are 0.0<br>0.0 if IN is negative and EXP does not correspond to an integer |

Description:

When the first argument is 'base' and the second argument is 'exponent', yields the REAL result of the following: ($base^{exponent}$). 'Exponent' is a REAL value.

## Example

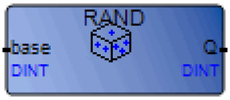
(* FBD Program using "POW" Function *)




(* ST Equivalence: *)

```
result := POW (xval, power);
```

# RAND




Arguments:

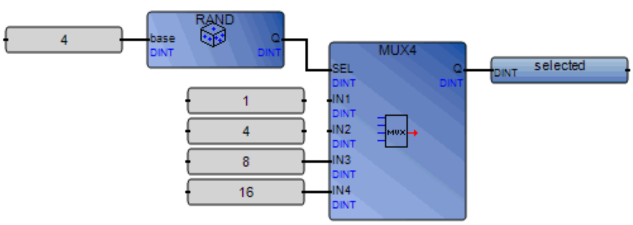
| | | | |
|---|---|---|---|
| base | base | DINT | Defines the allowed set of number |
| RAND | Q | DINT | Random value in set [0..base-1] |

Description:

From a defined range, yields random integer values.

## Example

(* FBD Program using "RAND" function *)

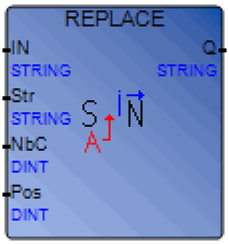



(* ST Equivalence: *)

```
selected := MUX4 ( RAND (4), 1, 4, 8, 16 );
```

```
(* random selection of 1 of 4 pre-defined values. The value issued of
RAND call is in set [0..3], so 'selected' issued from MUX4, will get
'randomly' the value 1 if 0 is issued from RAND,or 4 if 1 is issued
from RAND,or 8 if 2 is issued from RAND,or 16 if 3 is issued from RAND,

*)
```

# REPLACE




Arguments:

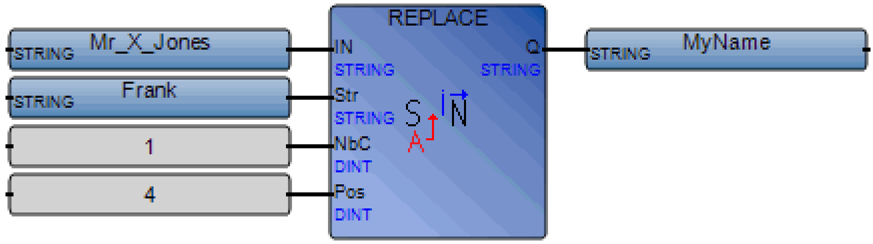
| | | | |
|---|---|---|---|
| IN | IN | STRING | Any string |
| Str | Str | STRING | String to be inserted (to replace NbC chars) |
| NbC | NbC | DINT | Number of characters to be deleted |
| Pos | Pos | DINT | Position of the first modified character<br>(first valid position is 1) |
| REPLACE | Q | STRING | Modified string:<br>- NbC characters are deleted at position Pos<br>- then substring Str is inserted at this position<br>yields empty string if Pos <= 0<br>yields strings concatenation (IN+Str) if Pos is greater than the length of the IN string<br>yields initial string IN if NbC <= 0 |

Description:

Replaces parts of a strings with new sets of characters.

## Example

(* FBD Program using "REPLACE" function *)




(* ST Equivalence: *)

```
MyName := REPLACE ('Mr X JONES, 'Frank', 1, 4);
```

(* MyName is 'Mr Frank JONES' *)

# RIGHT


Arguments:

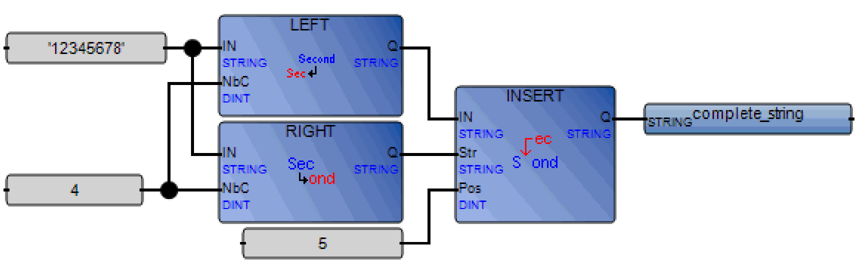
| | | | |
|---|---|---|---|
| IN | IN | STRING | Any non-empty string |
| NbC | NbC | DINT | Number of characters to be extracted. This number cannot be greater than the length of the IN string. |
| RIGHT | Q | STRING | Right part of the string (length = NbC)<br>empty string if NbC <= 0<br>complete string if NbC >= string length |

Description:

From the right ends of strings, yields the number of characters defined.

### Example

(* FBD Program using "LEFT" and "RIGHT" Functions *)




(* ST Equivalence: *)

```
complete_string := INSERT (RIGHT ('12345678', 4), LEFT ('12345678',
4),5);
```

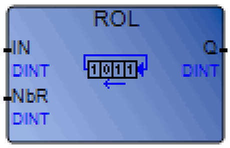
(* complete_string is '56781234'

the value issued from RIGHT call is '5678'

the value issued from LEFT call is '1234'

*)

# ROL

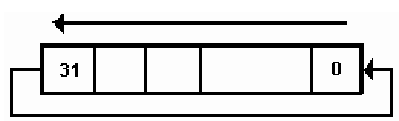



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DINT | Integer value |
| NbR | NbR | DINT | Number of 1-bit rotations (in set [1..31]) |
| ROL | Q | DINT | Left rotated value<br>When NbR <= 0, no change occurs. |

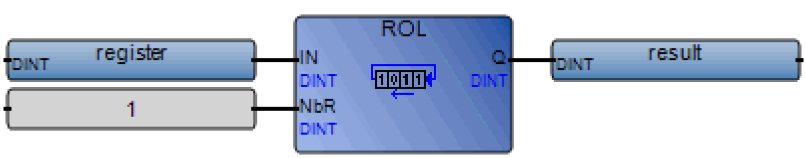
Description:

For 32-bit integers, rotates integer bits to the left.




## Example

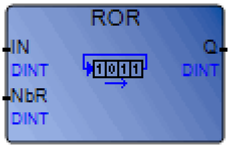
(* FBD Program using "ROL" Function *)




(* ST Equivalence: *)

```
result := ROL (register, 1);
(* register = 2#0100_1101_0011_0101*)
(* result = 2#1001_1010_0110_1010*)
```

# ROR

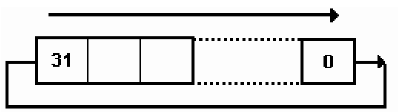



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DINT | Any integer value |
| NbR | NbR | DINT | Number of 1 bit rotations (in set [1..31]) |
| ROR | Q | DINT | Right rotated value<br>no effect if NbR <= 0 |

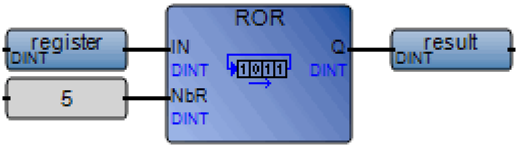
Description:

For 32-bit integers, rotates integer bits to the right.



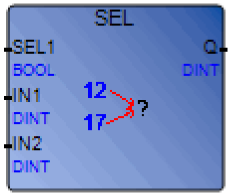

## Example

(* FBD Program using "ROR" Function *)




```
(* ST Equivalence: *)

result := ROR (register, 2);

(* register = 2#0011_0011_0010_1011_0011_0010_1001_1001 *)

(* result = 2#0100_1100_1100_1010_1100_1100_1010_0110 *)
```

# SEL




Arguments:

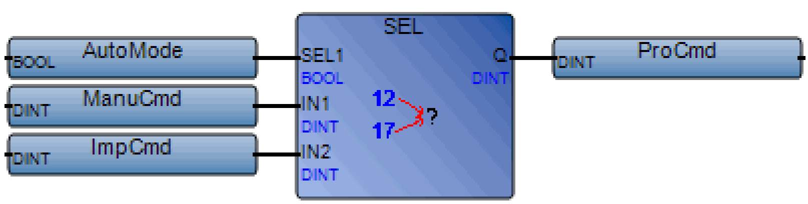
| | | | |
|---|---|---|---|
| SEL1 | SEL1 | BOOL | Indicates the chosen value |
| IN1, IN2 | IN1, IN2 | DINT | Any integer values |
| SEL | Q | DINT | = IN1 if SEL is FALSE<br>= IN2 if SEL is TRUE |

Description:

Specifies the input to use between two integer values.

### Example

(* FBD Program using "SEL" Function *)

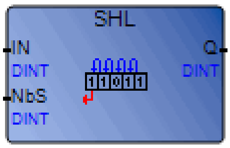



(* ST Equivalence: *)

```
ProCmd := SEL (AutoMode, ManuCmd, InpCmd);
```

(* process command selection *)

# SHL

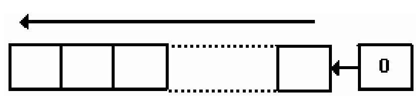



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DINT | Any integer value |
| NbS | NbS | DINT | Number of 1 bit shifts (in set [1..31]) |
| SHL | Q | DINT | Left shifted value<br>no effect if NbS <= 0<br>0 replaces the least significant bit |

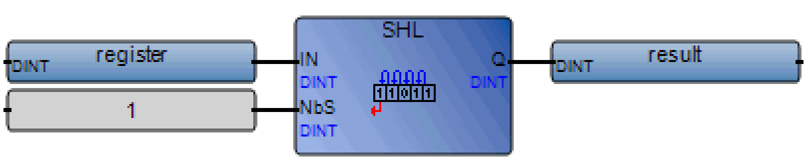
Description:

For 32-bit integers, moves integers to the left and places 0 in the least significant bit.




## Example

(* FBD Program using "SHL" Function *)




(* ST Equivalence: *)

```
result := SHL (register,1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1001_1010_0110_1010 *)
```

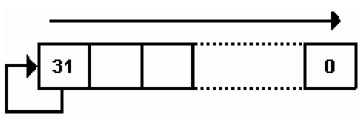
# SHR


Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DINT | Any integer value |
| NbS | NbS | DINT | Number of 1 bit shifts (in set [1..31]) |
| SHR | Q | DINT | Right shifted value<br>no effect if NbS <= 0<br>the leftmost bit is replicated if NbS >=1 |

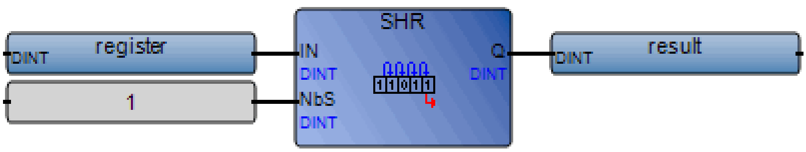
Description:

Shifts the 32 bits of an integer to the right and replicates the leftmost bit (significant bit) to fill the vacant bits.




## Example

(* FBD Program using "SHR" Function *)

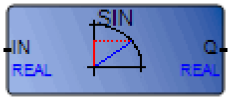



(* ST Equivalence: *)

```
result := SHR (register,1);
(* register = 2#1100_1101_0011_0101 *)
```

```
(* result = 2#1110_0110_1001_1010 *)
```

## SIN




Arguments:

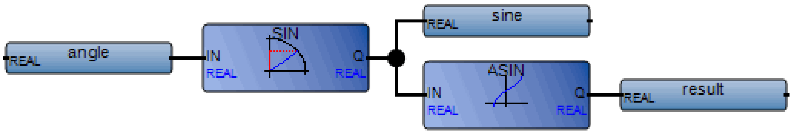
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any REAL value |
| SIN | Q | REAL | Sine of the input value (in set [-1.0 .. +1.0]) |

Description:

Yields the Sine of a REAL value.

### Example

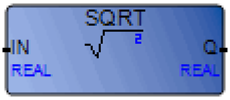
(* FBD Program using "SIN" and "ASIN" Functions *)




(* ST Equivalence: *)

```
sine := SIN (angle);
result := ASIN (sine); (* result is equal to angle *)
```

# SQRT




Arguments:

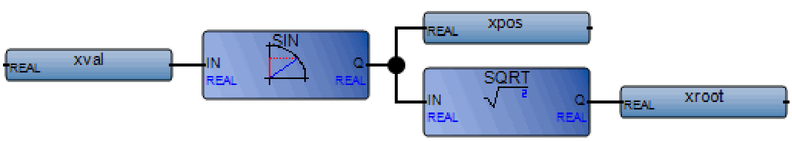
| | | | |
|---|---|---|---|
| IN | IN | REAL | Must be greater than or equal to zero |
| SQRT | Q | REAL | Square root of the input value |

Description:

Yields the square root of a REAL value.

## Example

(* FBD Program using "SQRT" Function *)




(* ST Equivalence: *)

```
xpos := ABS (xval);
xroot := SQRT (xpos);
```

# SUB_DATE_DATE


Arguments:

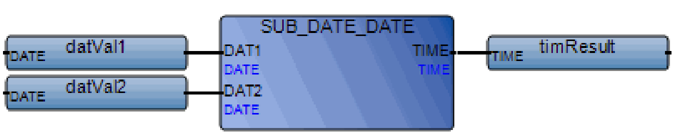
| | | | |
|---|---|---|---|
| DAT1 | DAT1 | DATE | First date in a comparison |
| DAT2 | DAT2 | DATE | Second date in a comparison |
| SUB_DATE_DATE | TIME | TIME | Difference in TIME format between DAT1 and DAT2. The possible date difference values range from t#0h to t#1193h2m47s294ms inclusively. A value of 1193h2m47s295ms indicates an error for either of the following conditions:<br>- DAT1 is less than DAT2<br>- The difference between DAT1 and DAT2 is greater than 1193h2m47s294ms |

Description:

Compares two dates and yields the difference in TIME format.

### Example

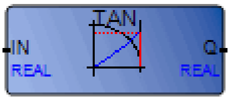
(* FBD Program using "`SUB_DATE_DATE`" Function *)




(* ST Equivalence: *)

```
timResult := SUB_DATE_DATE (datVal1, datVal2);
```

# TAN




Arguments:

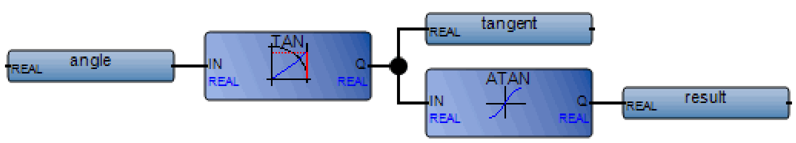
| | | | |
|---|---|---|---|
| IN | IN | REAL | Cannot be equal to PI/2 modulo PI |
| TAN | Q | REAL | Tangent of the input value<br>= 1E+38 for invalid input |

Description:

Yields the Tangent of a REAL value.

## Example

(* FBD Program using "TAN" and "ATAN" Functions *)




(* ST Equivalence: *)

```
tangent := TAN (angle);
result := ATAN (tangent); (* result is equal to angle*)
```

# TRUNC


Arguments:

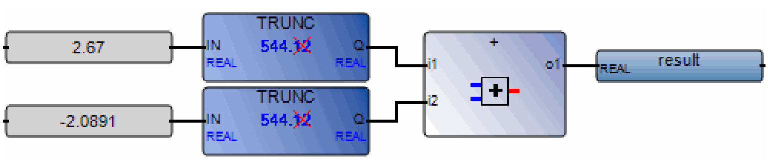
| | | | |
|---|---|---|---|
| IN | IN | REAL | Any REAL value |
| TRUNC | Q | REAL | If IN>0, biggest integer less or equal to the input<br>If IN<0, least integer greater or equal to the input |

Description:

Truncates REAL values, leaving just the integer.

### Example

(* FBD Program using "TRUNC" Function *)




(* ST Equivalence: *)

```
result := TRUNC (+2.67) + TRUNC (-2.0891);
(* means: result := 2.0 + (-2.0) := 0.0; *)
```

# UNLOCK_CPU


Arguments:

| | | | |
|---|---|---|---|
| OK | OK | BOOL | Status of the thread unlock operation<br>TRUE=Successful unlock operation<br>FALSE=Unsuccessful unlock operation |

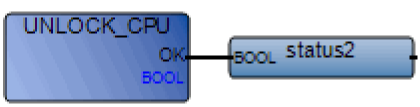
Description:

When using interrupts, releases the lock on the data space, granting other interrupt POUs access.

**Note:** Ensure UNLOCK_CPU is called once in a given thread before calling LOCK_CPU.

## Example

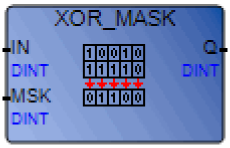
(* FBD Program using "UNLOCK_CPU" Function *)




(* ST Equivalence: *)

```
status2:=UNLOCK_CPU();
```

# XOR_MASK




Arguments:

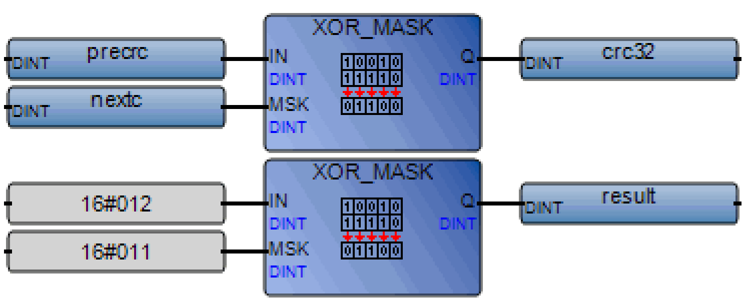
| | | | |
|---|---|---|---|
| IN | IN | DINT | Must have integer format |
| MSK | MSK | DINT | Must have integer format |
| XOR_MASK | Q | DINT | Bit-to-bit logical **Exclusive OR** between IN and MSK |

Description:

Integer exclusive OR bit-to-bit mask

### Example

(* FBD example with **XOR_MASK** Operators *)




(* ST Equivalence: *)

```
crc32 := XOR_MASK (prevcrc, nextc);
result := XOR_MASK (16#012, 16#011); (* equals 16#003 *)
```

# Function Blocks

The workbench supports the following function blocks:

| | | |
|---|---|---|
| **Alarms Management** | LIM_ALRM | High/low limit alarm with hysteresis |
| **Boolean Operations** | SR | Set dominant bistable |
| | RS | Reset dominant bistable |
| | R_TRIG | Rising edge detection |
| | F_TRIG | Falling edge detection |
| **Comparator Operations** | CMP | Full comparison function block |
| **Counters** | CTU | Up counter |
| | CTD | Down counter |
| | CTUD | Up-down counter |
| **Process Control** | AVERAGE | Running average over N samples |
| | BLINK | Blinking Boolean signal |
| | DERIVATE | Differentiation of a real value according to time |
| | HYSTER | Boolean hysteresis on difference of reals |
| | INTEGRAL | Integration over time |
| | SIG_GEN | Signal generator |
| | STACKINT | Stack of integer |
| **Remote Device Communications** | CONNECT | Connection to a resource |
| | USEND_S | Sending of a message to a resource |
| | URCV_S | Reception of a message from a resource |
| **Time Operations** | TON | On-delay timing |
| | TOF | Off-delay timing |
| | TP | Pulse timing |

**Note:** When new function blocks are created, these can be called from any language.

# AVERAGE


Arguments:

| | | |
|---|---|---|
| RUN | BOOL | TRUE=run / FALSE=reset |
| XIN | REAL | Any real Variable |
| N | DINT | Application defined number of samples |
| XOUT | REAL | Running average of XIN value |

**Note:** When setting or changing the value for N, you need to set RUN to FALSE, then set it back to TRUE.

Description:

Stores a value at each cycle and calculates the average value of all stored values. Only the latest N values are stored.

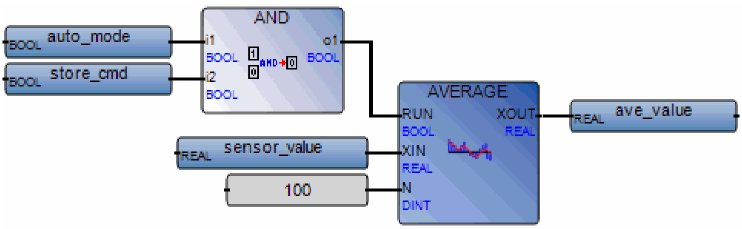
The maximum number of samples N is 128. When N exceeds 128, the number of samples is truncated to 128.

When the "RUN" command is FALSE (reset mode), the output value is equal to the input value.

Upon reaching the maximum N of stored values, the first stored value is overwritten with the latest value.

### Example

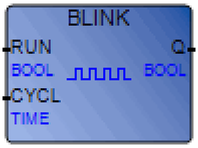
(* FBD program using the AVERAGE block: *)




(* ST Equivalence: AVERAGE1 instance of AVERAGE block *)

```
AVERAGE1((auto_mode & store_cmd), sensor_value, 100);
ave_value := AVERAGE1.XOUT;
```

# BLINK



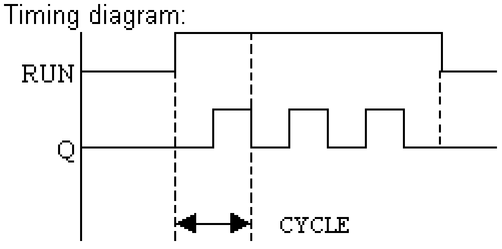

Arguments:

| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=blinking / FALSE=reset the output to false |
| CYCL | TIME | Blinking period. Possible values range from 0ms to 1193h2m47s294ms. |
| Q | BOOL | Output blinking signal |

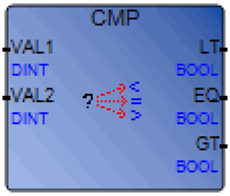
Description:

Generates a blinking signal.

Timing diagram:



# CMP




Arguments:

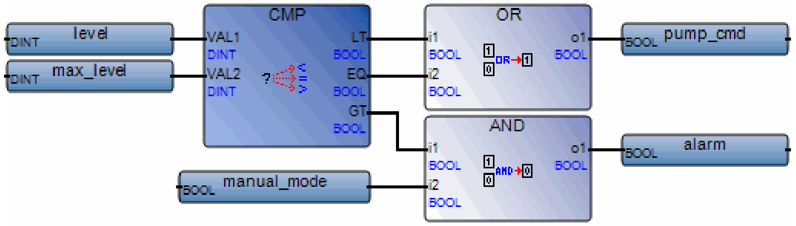
| | | |
|---|---|---|
| VAL1 | DINT | Any signed integer value |
| VAL2 | DINT | Any signed integer value |
| LT | BOOL | TRUE if val1 is Less Than val2 |
| EQ | BOOL | TRUE if val1 is Equal to val2 |
| GT | BOOL | TRUE if val1 is Greater Than val2 |

Description:

Compare two values: tell if they are equal, or if the first is less or greater than the second one.

## Example

(* FBD program using the CMP block *)

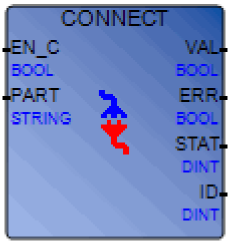



(* ST Equivalence: We suppose CMP1 is an instance of CMP block *)

```
CMP1(level, max_level);
pump_cmd := CMP1.LT OR CMP1.EQ;
```

```
alarm := CMP1.GT AND NOT(manual_mode);
```

# CONNECT




**Note:** This function is only available for use with multi-task runtimes.

Arguments:

| | | |
|---|---|---|
| EN_C | BOOL | Enable connection. |
| PART | STRING | Name of the remote communication partner. |
| VAL | BOOL | If TRUE, connection ID is valid. |
| ERR | BOOL | If TRUE, new non-zero status received. |
| STAT | DINT | Last detected status. |
| ID | DINT | Identification of the communication Channel. |

Description:

Creates a connection with a remote or local virtual machine (of current Project or another Project) and manages the exchanges (for blocks USEND_S and URCV_S).

It creates a communication channel identifier (ID).

This identifier is required in all others communication function blocks (URCV_S or USEND_S).

PARTNER parameter is a string with the following format:

```
'ResourceNumber@Address'
```

### Example

```
'1@123.45.67.89'
```

Connection with the ETCP driver to Resource 1 at address 123.45.67.89.

If the resource is on the same device, its number is enough to identify it (e.g. '1').

On a rising edge of EN_C parameter, the CONNECT block establishes the communication with the remote partner.

The VALID parameter is set to TRUE until the communication is available.

Every time the status changes, the output parameter ERROR is set to TRUE during one cycle and the new status is set in the STATUS parameter.

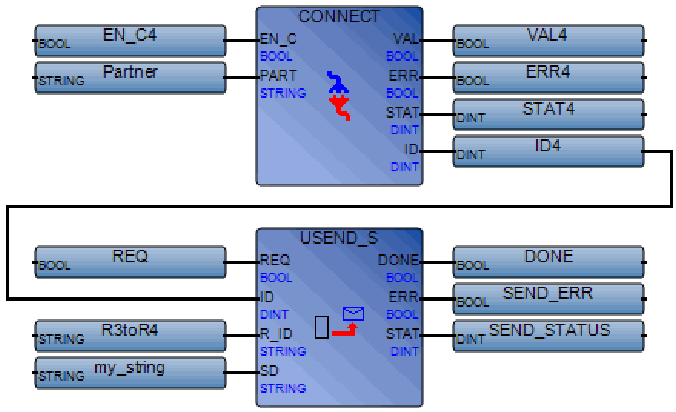
STATUS can take following values:

| STATUS | Description |
|---|---|
| 0 | Connection successfully completed. |
| 1 | Waiting for reply |
| 2 | Too many CFB connect |
| 3 | Not ready for a new connection |
| 4 | Connect failed |
| 5 | Bad partner |

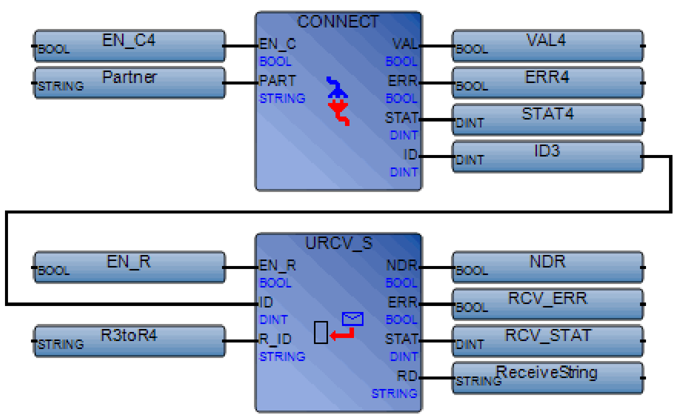
If the connection failed, a new connection is not automatically done, a rising edge must be detected on EN_C parameter.

### Example

The following shows a program from Resource 3 sending a string to Resource 4 on the same device:




The following shows the corresponding program in Resource 4 receiving the string:

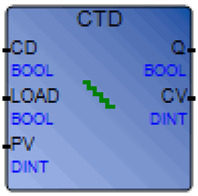



**See Also**

USEND_S

URCV_S

# CTD




Arguments:

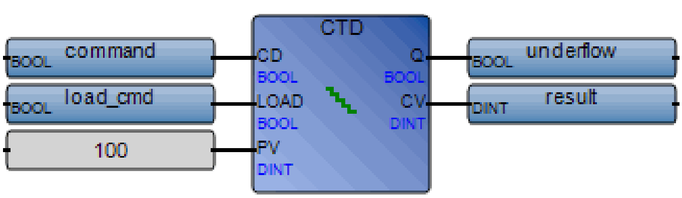
| | | |
|---|---|---|
| CD | BOOL | Counting input<br>(down-counting when CD is a rising edge) |
| LOAD | BOOL | Load command (dominant)<br>(CV = PV when LOAD is TRUE) |
| PV | DINT | Programmed initial value |
| Q | BOOL | Underflow: TRUE when CV <= 0 |
| CV | DINT | Counter result |

Description:

Counts (integer) from a given value down to 0 1 by 1

### Example

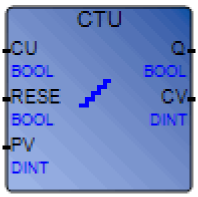
(* FBD program using the CTD block *)




```
(* ST Equivalence:  CTD1 is an instance of block*)
CTD1(trigger,load_cmd,100);
underflow := CTD1.Q;
```

```
result := CTD1.CV;
```

# CTU




Arguments:

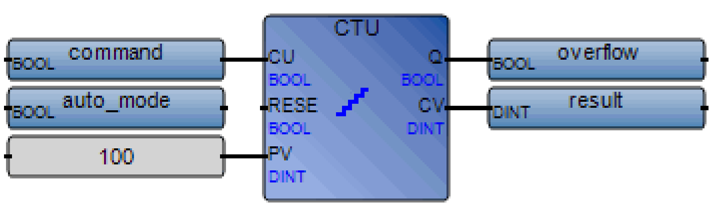
| | | |
|---|---|---|
| CU | BOOL | Counting input (counting when CU is a rising edge) |
| RESE | BOOL | Reset command (dominant) |
| PV | DINT | Programmed maximum value |
| Q | BOOL | Overflow: TRUE when CV >= PV |
| CV | DINT | Counter result |

Description:

Counts (integer) from 0 up to a given value 1 by 1

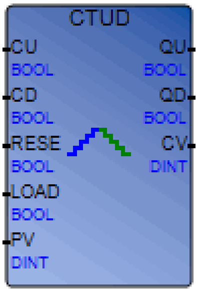
### Example

(* FBD program using the CTU block *)




```
(* ST Equivalence: CTU1 is an instance of CTU block*)
CTU1(trigger,NOT(auto_mode),100);
overflow := CTU1.Q;
result := CTU1.CV;
```

# CTUD




Arguments:

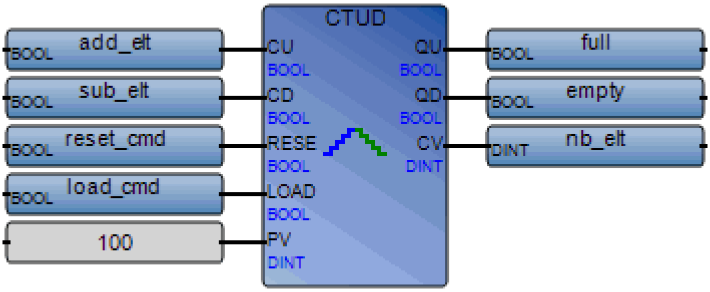
| | | |
|---|---|---|
| CU | BOOL | Up-counting (when CU is a rising edge) |
| CD | BOOL | Down-counting (when CD is a rising edge) |
| RESE | BOOL | Reset command (dominant)<br>(CV = 0 when RESET is TRUE) |
| LOAD | BOOL | Load command (CV = PV when LOAD is TRUE) |
| PV | DINT | Programmed maximum value |
| QU | BOOL | Overflow: TRUE when CV >= PV |
| QD | BOOL | Underflow: TRUE when CV <= 0 |
| CV | DINT | Counter result |

Description:

Counts (integer) from 0 up to a given value 1 by 1 or from a given value down to 0 1 by 1

## Example

(* FBD program using the CTUD block *)




```
(* ST Equivalence: We suppose CTUD1 is an instance of  block*)
CTUD1(trigger1, trigger2, reset_cmd, load_cmd,100);
full := CTUD1.QU;
empty := CTUD1.QD;
nb_elt := CTUD1.CV;
```

# DERIVATE


Arguments:

| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=normal / FALSE=reset |
| XIN | REAL | Input: any real value |
| CYCL | TIME | Sampling period. Possible values range from 0ms to 23h59m59s999ms. |
| XOUT | REAL | Differentiated output |

Description:

Differentiation of a real value.

If the "CYCLE" parameter value is less than the real duration of the cycle time in the virtual machine, the sampling period will use the real duration of the cycle time.

### Example

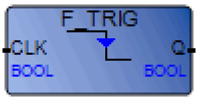
(* FBD program using the DERIVATE block: *)


(* ST Equivalence: DERIVATE1 instance of DERIVATE block *)

```
DERIVATE1(manual_mode, sensor_value, t#100ms);
```

```
derivated_value := DERIVATE1.XOUT;
```

# F_TRIG




Arguments:

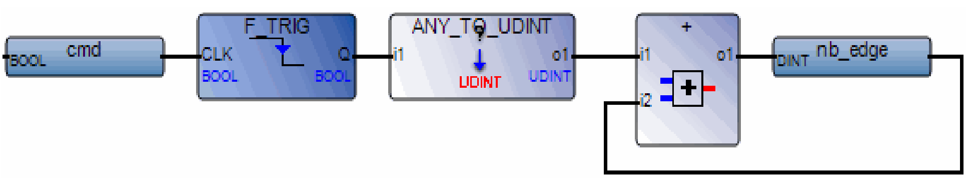
| | | |
|---|---|---|
| CLK | BOOL | Any Boolean Variable |
| Q | BOOL | TRUE when CLK changes from TRUE to FALSE<br>FALSE if all other cases |

Description:

Detects a falling edge of a Boolean variable

## Example

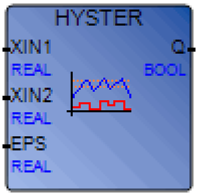
(* FBD program using the F_TRIG block *)




(* ST Equivalence: We suppose F_TRIG1 is an instance of F_TRIG block *)

```
F_TRIG1(cmd);

nb_edge := ANY_TO_DINT(F_TRIG1.Q) + nb_edge;
```

# HYSTER




Arguments:

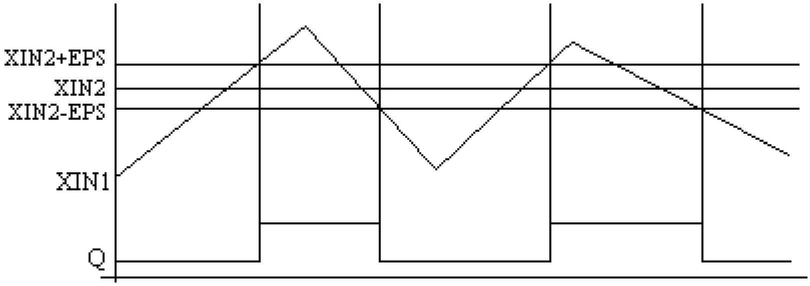
| | | |
|---|---|---|
| XIN1 | REAL | Any real value |
| XIN2 | REAL | To test if XIN1 has overpassed XIN2+EPS |
| EPS | REAL | Hysteresis value (must be greater than zero) |
| Q | BOOL | TRUE if XIN1 has overpassed XIN2+EPS and is not yet below XIN2-EPS |

Description:

Hysteresis on a real value for a high limit.

## Example

Example of a timing diagram:



# INTEGRAL


Arguments:

| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=integrate / FALSE=hold |
| R1 | BOOL | Overriding reset |
| XIN | REAL | Input: any real value |
| X0 | REAL | Initial value |
| CYCL | TIME | Sampling period. Possible values range from 0ms to 23h59m59s999ms. |
| Q | BOOL | Not R1 |
| XOUT | REAL | Integrated output |

Description:

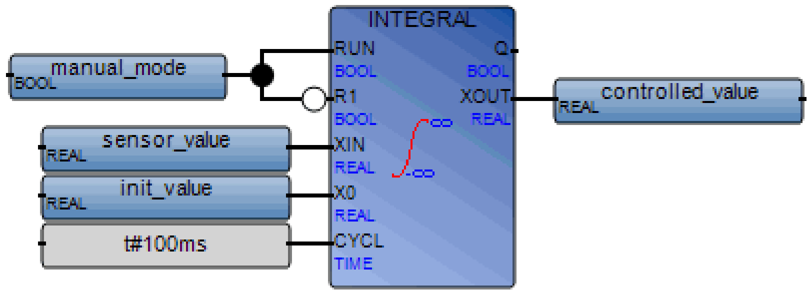
Integration of a real value.

If the "CYCLE" parameter value is less than the real duration of the cycle time in the virtual machine, the sampling period will use the real duration of the cycle time.

When using the Enable EN/ENO option for INTEGRAL blocks in LD POUs, you must reinitialize the internal variables for the R1 input. To reinitialize the R1 input, toggle the value from False to True then back to False.

### Example

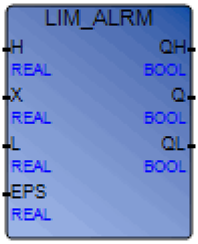
(* FBD Program using "INTEGRAL" Block: *)




(* ST Equivalence: INTEGRAL1 instance of INTEGRAL block *)

```
INTEGRAL1(manual_mode, NOT(manual_mode), sensor_value, init_value,
t#100ms);

controlled_value := INTEGRAL1.XOUT;
```

# LIM_ALRM




Arguments:

| | | |
|---|---|---|
| H | REAL | High limit value |
| X | REAL | Input: any real value |
| L | REAL | Low limit value |
| EPS | REAL | Hysteresis value (must be greater than zero) |
| QH | BOOL | "high" alarm: TRUE if X above high limit H |
| Q | BOOL | Alarm output: TRUE if X out of limits |
| QL | BOOL | "low" alarm: TRUE if X below low limit L |

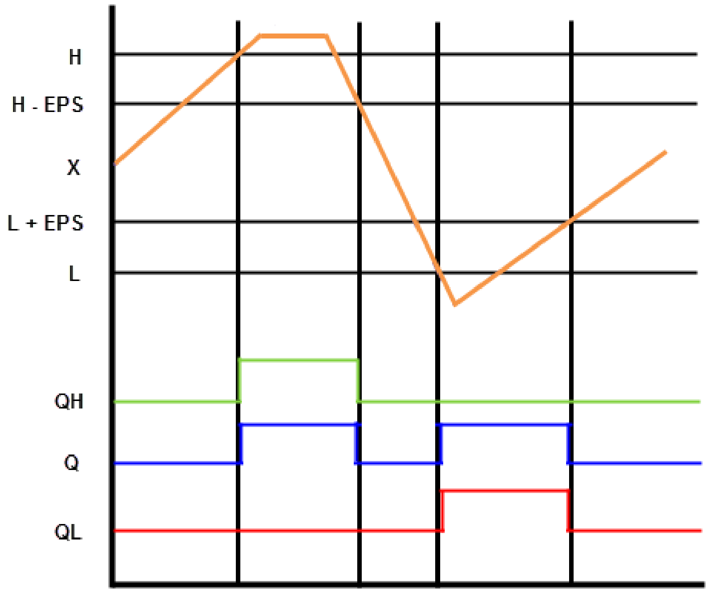
Description:

Hysteresis on a real value for high and low limits.

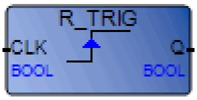
A hysteresis is applied on high and low limits. The hysteresis delta used for either the high or low limit is equal to the EPS parameter.

### Example

Example of timing diagram:


H
H - EPS
X
L + EPS
L
QH
Q
QL

# R_TRIG




Arguments:

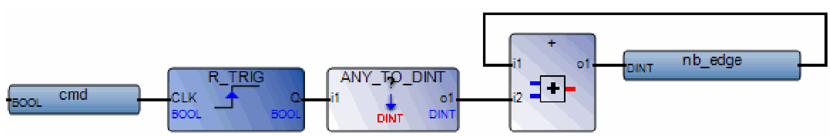
| | | |
|---|---|---|
| CLK | BOOL | Any Boolean Variable |
| Q | BOOL | TRUE when CLK rises from FALSE to TRUE<br>FALSE in all other cases |

Description:

Detects a rising edge of a Boolean variable

## Example

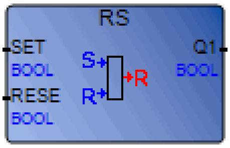
(* FBD program using the R_TRIG block *)




(* ST Equivalence: We suppose R_TRIG1 is an instance of the R_TRIG block *)

```
R_TRIG1(cmd);
nb_edge := ANY_TO_DINT(R_TRIG1.Q) + nb_edge;
```

# RS




Arguments:

| | | |
|---|---|---|
| SET | BOOL | If TRUE, sets Q1 to TRUE |
| RESE | BOOL | If TRUE, resets Q1 to FALSE (dominant) |
| Q1 | BOOL | Boolean memory state |

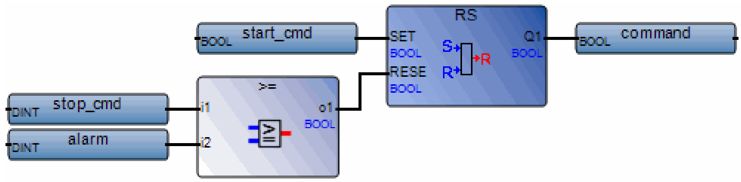
Description:

Reset dominant bistable:

| **Set** | **Reset1** | **Q1** | **Result Q1** |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

### Example

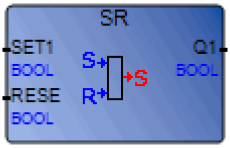
(* FBD Program using the RS block *)




(* ST Equivalence: We suppose RS1 is an instance of RS block *)

```
RS1(start_cmd, (stop_cmd OR alarm));
command := RS1.Q1;
```

# SR




Arguments:

| | | |
|---|---|---|
| SET1 | BOOL | If TRUE, sets Q1 to TRUE (dominant) |
| RESE | BOOL | If TRUE, resets Q1 to FALSE |
| Q1 | BOOL | Boolean memory state |

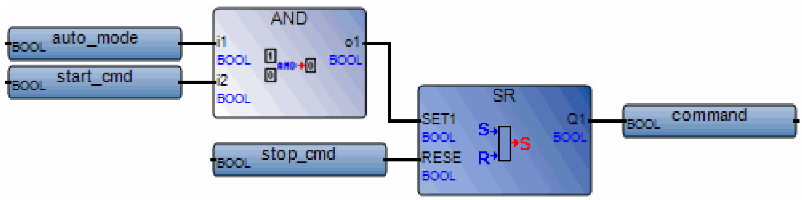
Description:

Set dominant bistable:

| **Set1** | **Reset** | **Q1** | **Result Q1** |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

### Example

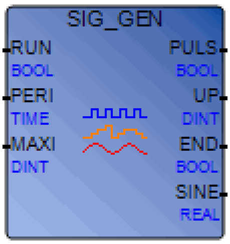
(* FBD Program using the SR block *)




(* ST Equivalence: We suppose SR1 is an instance of SR block *)

```
SR1((auto_mode & start_cmd), stop_cmd);

command := SR1.Q1;
```

# SIG_GEN




Arguments:

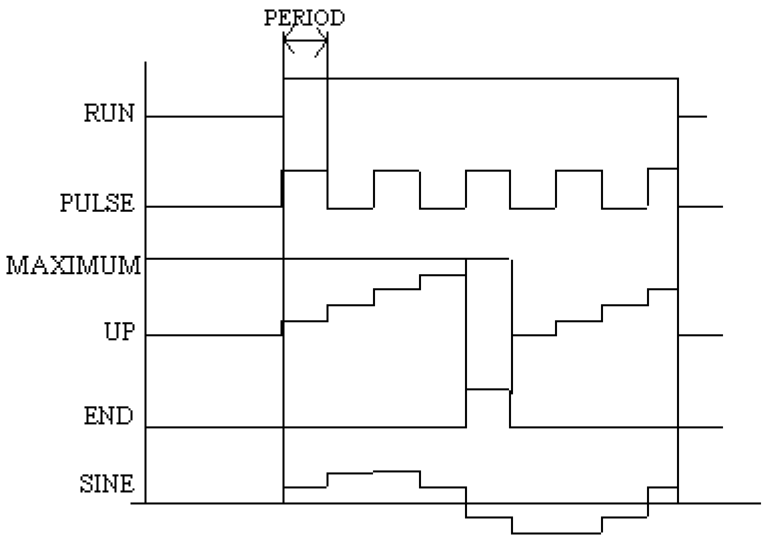
| | | |
|---|---|---|
| RUN | BOOL | Mode: TRUE=running / FALSE=reset to false |
| PERI | TIME | Duration of one sample. Possible values range from 0ms to 1193h2m47s294ms. |
| MAXI | DINT | Maximum counting value |
| PULS | BOOL | Inverted after each sample |
| UP | DINT | Up-counter, increased on each sample |
| END | BOOL | TRUE when up-counting ends |
| SINE | REAL | Sine signal (period = counting duration) |

Description:

Generates various signal: blink on a boolean, a integer counter-up, and real sine wave.

When counting reaches maximum value, it restarts from 0 (zero). So END keeps the TRUE value only during 1 PERIOD.

Timing diagram:


PERIOD
RUN
PULSE
MAXIMUM
UP
END
SINE

# STACKINT


Arguments:

| | | |
|---|---|---|
| PUSH | BOOL | Push command (on rising edge only)<br>add the IN value on the top of the stack |
| POP | BOOL | Pop command (on rising edge only)<br>delete in the stack the last value pushed (top of the stack) |
| R1 | BOOL | Resets the stack to its empty state |
| IN | DINT | Pushed value |
| N | DINT | Application defined stack size |
| EMPT | BOOL | TRUE if the stack is empty |
| OFLO | BOOL | Overflow: TRUE if the stack is full |
| OUT | DINT | Value at the top of the stack<br>OUT equals 0 when OFLO is TRUE |

Description:

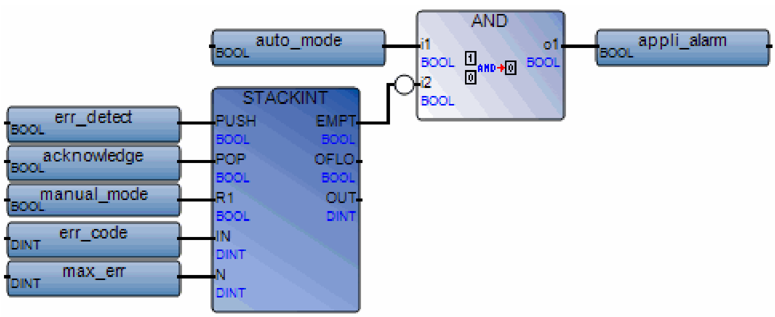
Manages a stack of integer values. The STACKINT function block includes a rising edge detection for both PUSH and POP commands. The maximum size of the stack is 128. The application defined stack size N cannot be less than 1 or greater than 128. This function manages invalid values as follows:

- if N<1, STACKINT assumes a size of 1
- if N>128, STACKINT assumes a size of 128

**Note:** The OFLO value is valid only after a reset (R1 has been set to TRUE at least once and back to FALSE).

### Example

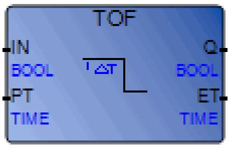
(* FBD program using the STACKINT block: error management *)




(* ST Equivalence: We suppose STACKINT1 is an instance of STACKINT block *)

```
STACKINT1(err_detect, acknowledge, manual_mode, err_code, max_err);
appli_alarm := auto_mode AND NOT(STACKINT1.EMPTY);
err_alarm := STACKINT1.OFLO;
last_error := STACKINT1.OUT;
```

# TOF




Arguments:

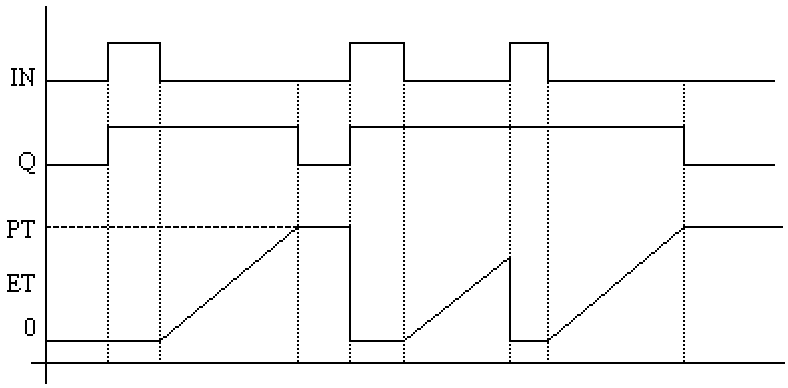
| | | |
|---|---|---|
| IN | BOOL | If falling edge, starts increasing internal timer<br>If rising edge, stops and resets internal timer |
| PT | TIME | Maximum programmed time |
| Q | BOOL | If TRUE: total time is not elapsed |
| ET | TIME | Current elapsed time. Possible values range from 0ms to 1193h2m47s294ms. |

Description:

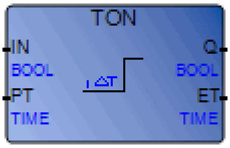
Increase an internal timer up to a given value.

While using the Enable EN/ENO option for LD POUs, execution disregards the TOF function block when EN is FALSE. When EN toggles from FALSE to TRUE, the function block is not reinitialized if IN is TRUE. To reinitialize the TOF function block, make sure IN is FALSE before setting EN to TRUE.

Timing diagram:


IN
Q
PT
ET
0

# TON




Arguments:

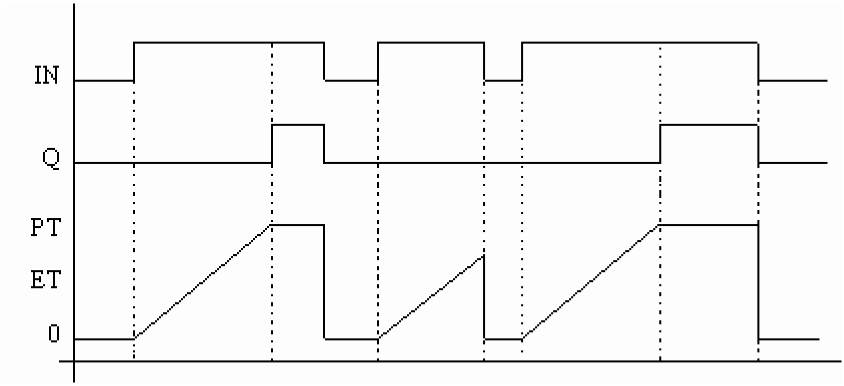
| | | |
|---|---|---|
| IN | BOOL | If rising edge, starts increasing internal timer<br>If falling edge, stops and resets internal timer |
| PT | TIME | Maximum programmed time |
| Q | BOOL | If TRUE, programmed time is elapsed |
| ET | TIME | Current elapsed time. Possible values range from 0ms to 1193h2m47s294ms. |

Description:

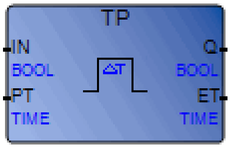
Increase an internal timer up to a given value.

While using the Enable EN/ENO option for LD POUs, execution disregards the TON function block when EN is FALSE. When EN toggles from FALSE to TRUE, the function block is not reinitialized if IN is TRUE. To reinitialize the TON function block, make sure IN is FALSE before setting EN to TRUE.

Timing diagram:



# TP




Arguments:

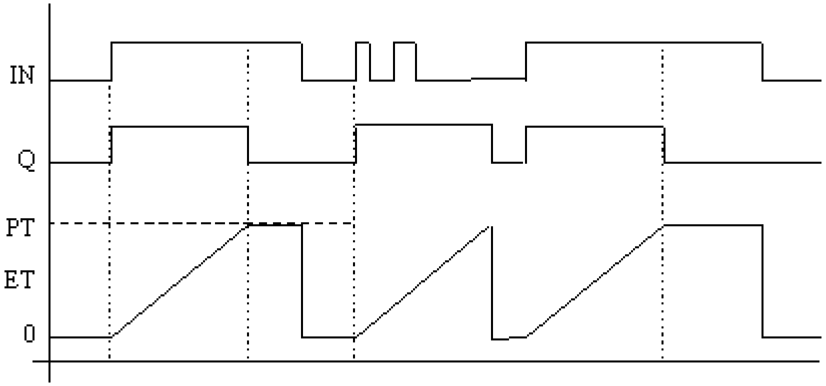
| | | |
|---|---|---|
| IN | BOOL | If rising edge, starts increasing internal timer (if not already increasing)<br>If FALSE and only if timer is elapsed, resets the internal timer<br>Any change on IN during counting has no effect. |
| PT | TIME | Maximum programmed time |
| Q | BOOL | If TRUE: timer is counting |
| ET | TIME | Current elapsed time. Possible values range from 0ms to 1193h2m47s294ms. |

Description:

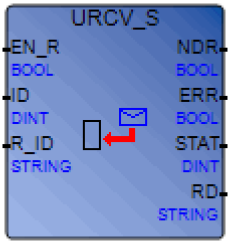
Increase an internal timer up to a given value.

While using the Enable EN/ENO option for LD POUs, execution disregards the TP function block when EN is FALSE. When EN toggles from FALSE to TRUE, the function block is not reinitialized if IN is TRUE. To reinitialize the TP function block, make sure IN is FALSE before setting EN to TRUE.

Timing diagram:



# URCV_S




**Note:** This function is only available for use with multi-task runtimes.

Arguments:

| | | |
|---|---|---|
| EN_R | BOOL | Enable to receive data |
| ID | DINT | Identification of the communication Channel |
| R_ID | STRING | Identification of the remote SFB inside the Channel |
| NDR | BOOL | If TRUE, new string received in RD |
| ERR | BOOL | If TRUE, new non-zero STATUS received |
| STAT | DINT | Last detected status |
| RD | STRING | Received string |

Description:

Receives a string from a remote or local virtual machine (of current Project or another Project).

**Warning:** Connect block must have been called in current cycle before the URCV_S call. This CFB receives a string from one URCV_S instance. Previously received string is overwritten. If string is successfully received then NDR is set to TRUE during one cycle. If an error occurs, the ERROR output parameter is set to TRUE and the status is set in the STATUS parameter.

STATUS can have the following values:

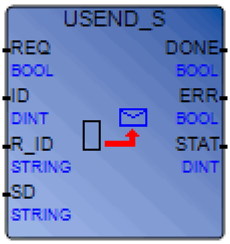
| STATUS | Description |
|---|---|
| 0 | Receive successfully completed |
| 1 | Waiting for message |
| 2 | Invalid identifier |
| 3 | Not ready for receive |
| 6 | Waiting for message |
| 7 | Dialog has failed |

See example in the description of the CONNECT block.

### See Also

USEND_S
CONNECT

# USEND_S




**Note:** This function is only available for use with multi-task runtimes.

Arguments:

| | | |
|---|---|---|
| REQ | BOOL | Send request on rising edge |
| ID | DINT | Identification of the communication channel |
| R_ID | STRING | Identification of the remote CFB inside the channel |
| SD | STRING | String to send |
| DONE | BOOL | If TRUE, function performed successfully |
| ERR | BOOL | If TRUE, new non-zero STATUS received |
| STAT | DINT | Last detected status |

Description:

Sends a string to a remote or local virtual machine (of current Project or another Project).

**Warning:** Connect block must have been called in current cycle before the USEND_S call. This CFB sends a string to one URCV_S instance on rising edge of REQ. If string is successfully sent then DONE is set. If an error occurs, the output parameter ERROR is set to TRUE and the status is set in the STATUS parameter.

STATUS can have the following values:

| STATUS | Description |
|---|---|
| 0 | Send successfully completed |
| 1 | Send in progress |

| | |
|---|---|
| 2 | Invalid identifier |
| 3 | Not ready to send |
| 6 | Dialog has failed |
| 7 | Send has failed |

If the send failed, a new send is not automatically done, a rising edge must be detected on REQ parameter.

See example in the description of the CONNECT block.

**See Also**

URCV_S
CONNECT

# Normative Function Blocks

The IEC 61499 language enables the distribution of individual normative function blocks belonging to an IEC 61499 program across multiple resources.

The IEC 61499 implementation is based on the *Function blocks - Part 1: Architecture* and *Function blocks - Part 2: Software Tools Requirements* documents available from the ANSI webstore.

**ISaGRAF** supports the following normative function blocks:

| | |
|---|---|
| **E_CTU** | Event-driven up counter |
| **E_CYCLE** | Periodic (cyclic) generation of an event |
| **E_D_FF** | D (Data latch) bistable |
| **E_DELAY** | Delayed propagation of an event |
| **E_DEMUX** | Generation of a finite train of separate events (table driven) |
| **E_F_TRIG** | Boolean falling edge detection |
| **E_MERGE** | Merge (OR) of multiple events |
| **E_N_TABLE** | Generation of a finite train of separate events (table driven) |
| **E_PERMIT** | Permissive propagation of an event |
| **E_R_TRIG** | Boolean rising edge detection |
| **E_REND** | Rendez-vous of two events |
| **E_RESTART** | Generation of restart events |
| **E_RS** | Event-driven bistable (Reset dominant) |
| **E_SELECT** | Selection between two events |
| **E_SPLIT** | Split an event |
| **E_SR** | Event-driven bistable (Set dominant) |
| **E_SWITCH** | Switching (demultiplexing) an event |

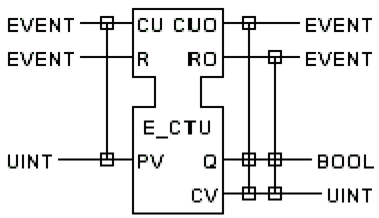
| | |
|---|---|
| **E_TABLE** | Generation of a finite train of events (table driven) |
| **E_TABLE_CTRL** | Generation of a finite train of events (table driven) |
| **E_TRAIN** | Generation of a finite train of events |
| **LocalEventInput** | Automatically assigned to normative function block arguments having the event input direction |

**Note:** When new function blocks are created, these can be called from any language.

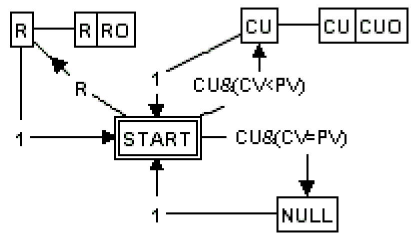
# E_CTU

**Event-driven up counter**

Interface




ECC/Algorithms/Service sequences




```
ALGORITHM R IN ST: (* Reset *)
CV:= 0;
Q:= 0;
END_ALGORITHM
```

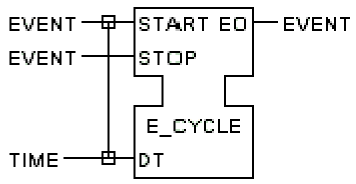
```
ALGORITHM CU IN ST: (* Count up *)
CV:= CV+1;
Q:= (CV=PV);
END_ALGORITHM
```

When an event triggers the E_CTU counter function block, CV (Current Value) begins incrementing from 0 to the maximum value, defined by PV (Programmed Value). When CV reaches the maximum value, Q passes to TRUE and the counter stops incrementing. An E_CTU counter cycle restarts when R (Reset) becomes TRUE.

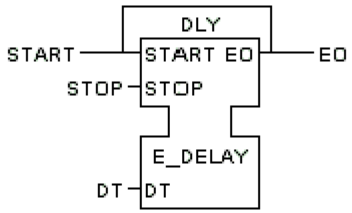
# E_CYCLE

**Periodic (cyclic) generation of an event**

Interface

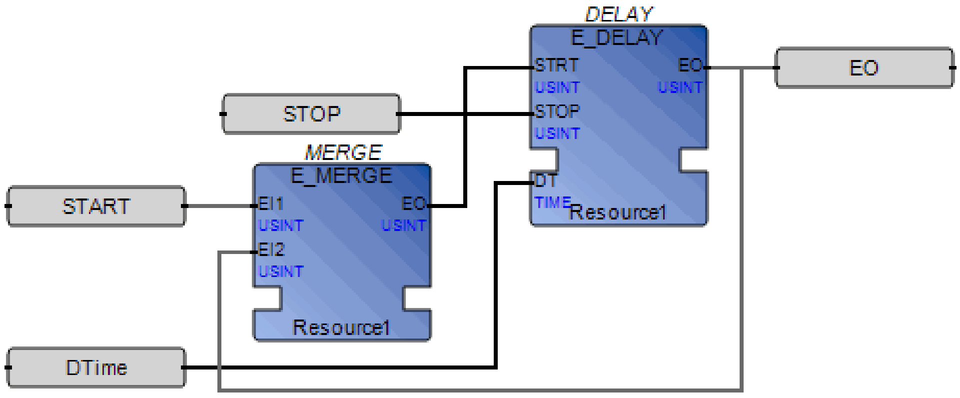



ECC/Algorithms/Service sequences




An event occurs at EO at an interval DT after the occurrence of an event at START, and at intervals of DT thereafter until the occurrence of an event at STOP.

IEC 61499 FBD Definition



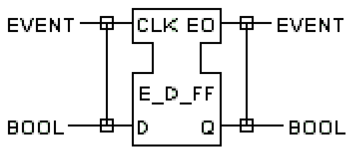

**Related Topics**

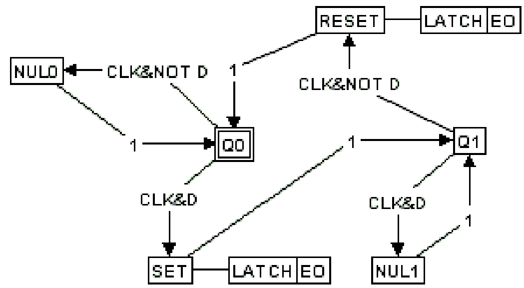
E_DELAY

# E_D_FF

**D (Data latch) bistable**

Interface | ECC/Algorithms/Service sequences

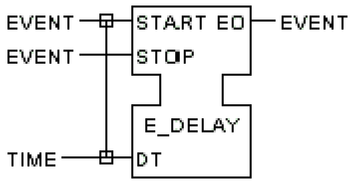



```
ALGORITHM LATCH IN ST :
Q := D ;
END_ALGORITHM
```

# E_DELAY

**Delayed propagation of an event**

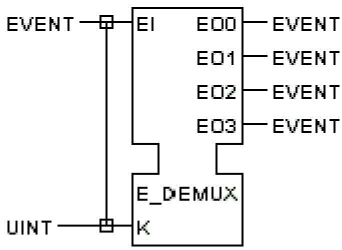



An event at EO is generated at a time interval DT after the occurrence of an event at the START input. The event delay is cancelled by an occurrence of an event at the STOP input. If multiple events occur at the START input before the occurrence of an event at EO, only a single event occurs at EO, at a time DT after the first event occurrence at the START input.

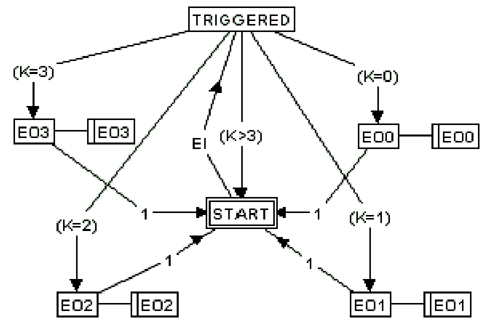
# E_DEMUX

**Generation of a finite train of separate events (table driven)**

Interface




ECC/Algorithms/Service sequences




Implementation using the E_DEMUX function block type as shown is not a normative requirement. Equivalent functionality may be implemented by various means.

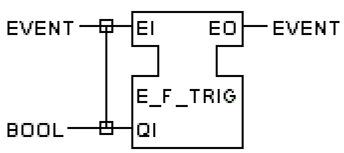
## Related Topics

E_N_TABLE

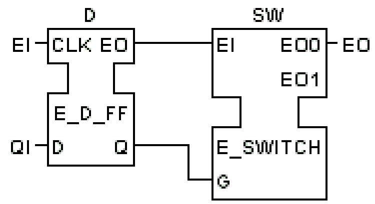
# E_F_TRIG

**Boolean falling edge detection**

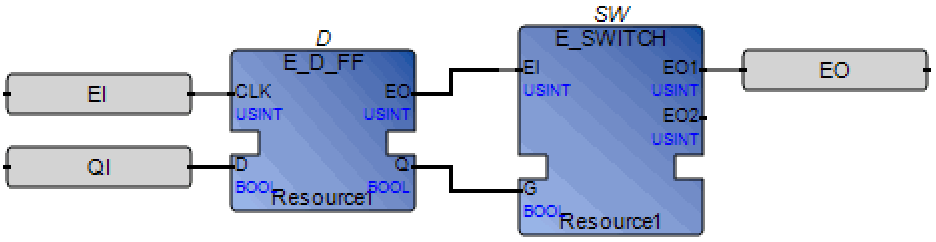
Interface




ECC/Algorithms/Service sequences




IEC 61499 FBD Definition




**Related Topics**

E_R_TRIG

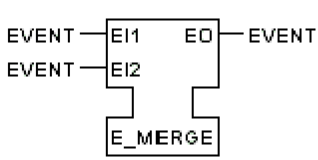
E_D_FF

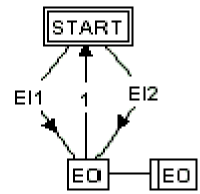
E_SWITCH

# E_MERGE

**Merge (OR) of multiple events**

Interface ECC/Algorithms/Service sequences




The occurrence of an event at any of the inputs EI1, EI2,...,EIn causes the occurrence of an event at EO (n=2 in the above example).

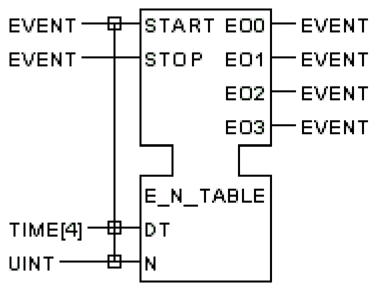
**Related Topics**

E_R_TRIG

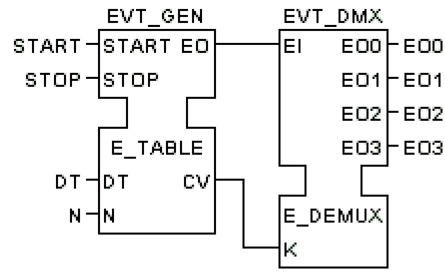
# E_N_TABLE

## Generation of a finite train of separate events (table driven)

Interface



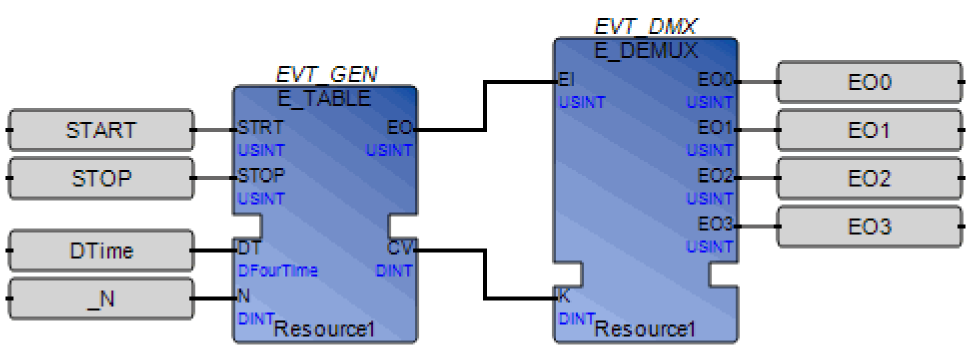

ECC/Algorithms/Service sequences




An event occurs at `EO0` at an interval DT[0] after the occurrence of an event at `EI`. An event occurs at `EO2` an interval DT[1] after the occurrence of the event at `EO1`, etc., until `N` occurrences have been generated or an event occurs at the `STOP` input.

NOTE - In this example implementation, `N` <= 4.

IEC 61499 FBD Definition



**Related Topics**

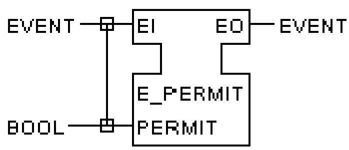
E_TABLE

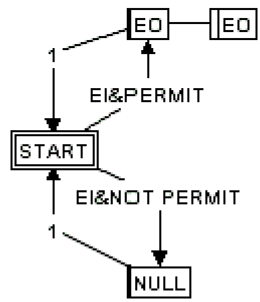
E_DEMUX

# E_PERMIT

**Permissive propagation of an event**

Interface | ECC/Algorithms/Service sequences

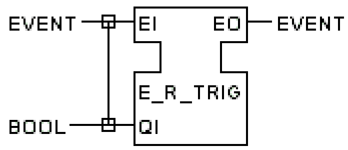


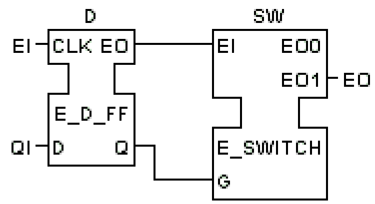
# E_R_TRIG

**Boolean rising edge detection**

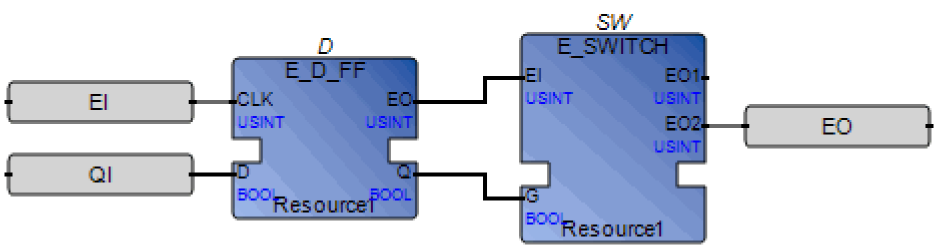
Interface




ECC/Algorithms/Service sequences




IEC 61499 FBD Definition



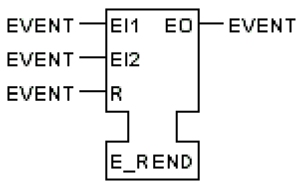

**Related Topics**

E_F_TRIG

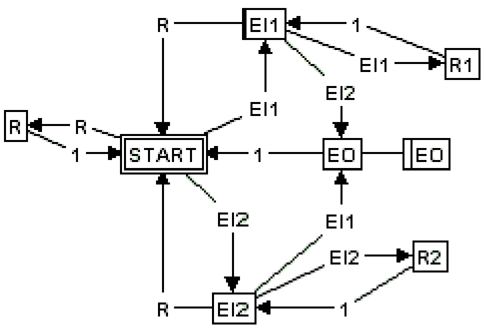
# E_REND

**Rendezvous of two events**

Interface



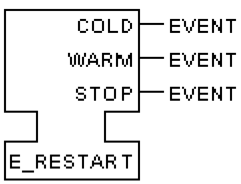

ECC/Algorithms/Service sequences



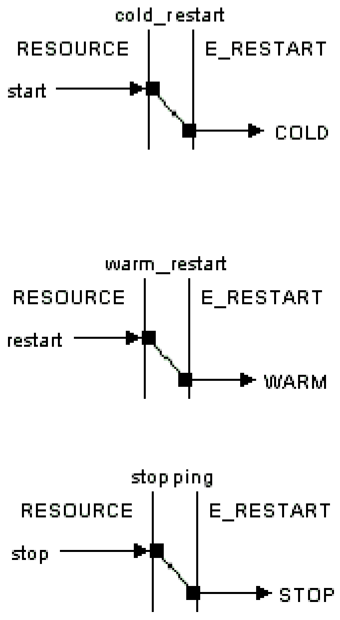
# E_RESTART

**Generation of restart events**

Interface




ECC/Algorithms/Service sequences




**1.** An event is issued at the COLD output upon "cold restart" of the associated resource.

**2.** An event is issued at the WARM output upon "warm restart" of the associated resource.

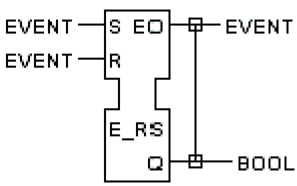
**3.** An event is issued at the STOP output (if possible) prior to "stopping" of the associated resource.

# E_RS

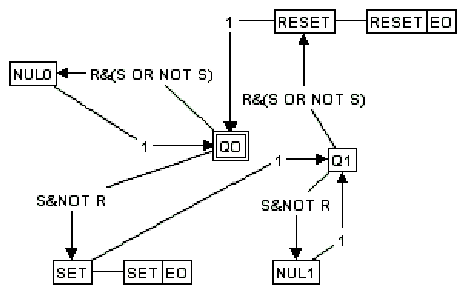
**Event-driven bistable (Reset dominant)**

The output `Q` is set to 1 (TRUE) upon the occurrence of an event at the `S` input, and is reset to 0 (FALSE) upon the occurrence of an event at the `R` input. If simultaneous `S` and `R` events occur, the `R` input is dominant. An event is issued at the `EO` output when the value of `Q` changes.

Interface




ECC/Algorithms/Service sequences

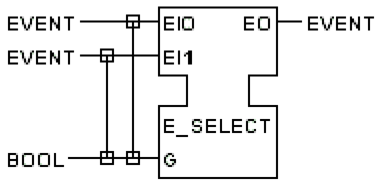



NOTE - Algorithms SET and RESET are the same as for E_SR.

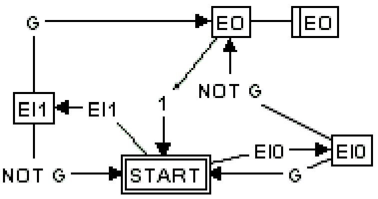
# E_SELECT

**Selection between two events**

Interface



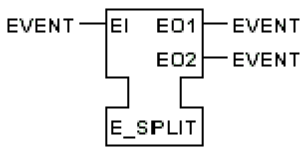

ECC/Algorithms/Service sequences

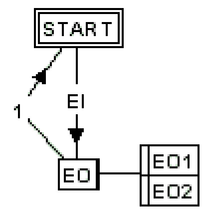


# E_SPLIT

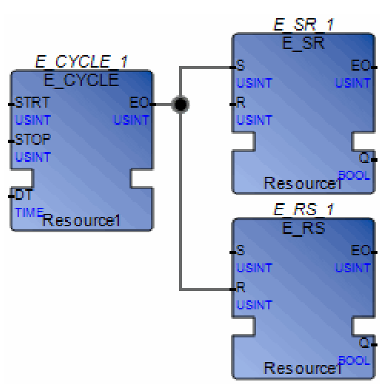
**Split an event**

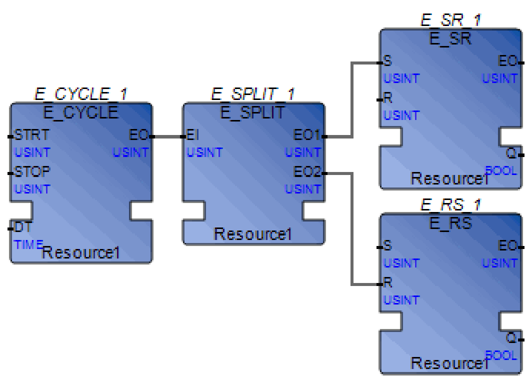
Interface | ECC/Algorithms/Service sequences




**ISaGRAF** automatically performs the E_SPLIT operation during compilation for all event and data outputs. Therefore, the diagram on the left, without the E_SPLIT function block, is equivalent to the diagram on the right.



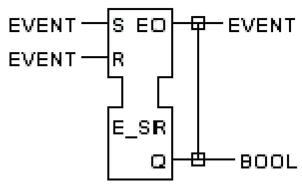



# E_SR

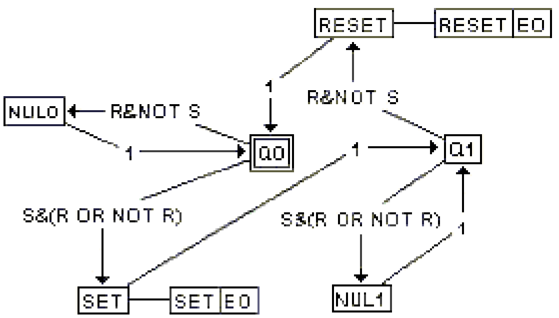
### Event-driven bistable (Set dominant)

The output Q is set to 1 (TRUE) upon the occurrence of an event at the S input, and is reset to 0 (FALSE) upon the occurrence of an event at the R input. If simultaneous S and R events occur, the S input is dominant. An event is issued at the EO output when the value of Q changes.

Interface




ECC/Algorithms/Service sequences



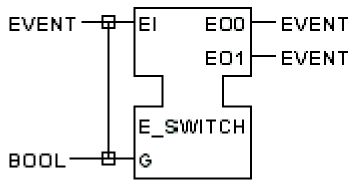

```
ALGORITHM SET IN ST : (* Set Q *)
  Q := TRUE ;
END_ALGORITHM
```

```
ALGORITHM RESET IN ST : (* Reset Q *)
  Q := FALSE ;
END_ALGORITHM
```

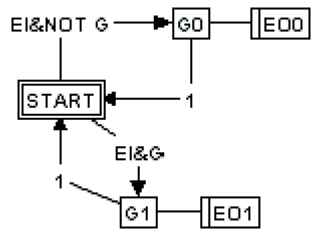
# E_SWITCH

**Switching (demultiplexing) an event**

Interface

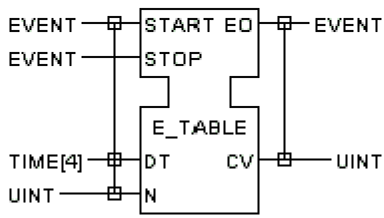


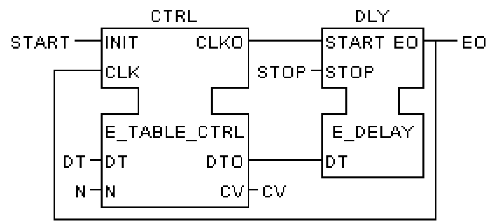

ECC/Algorithms/Service sequences



# E_TABLE

**Generation of a finite train of events (table driven)**

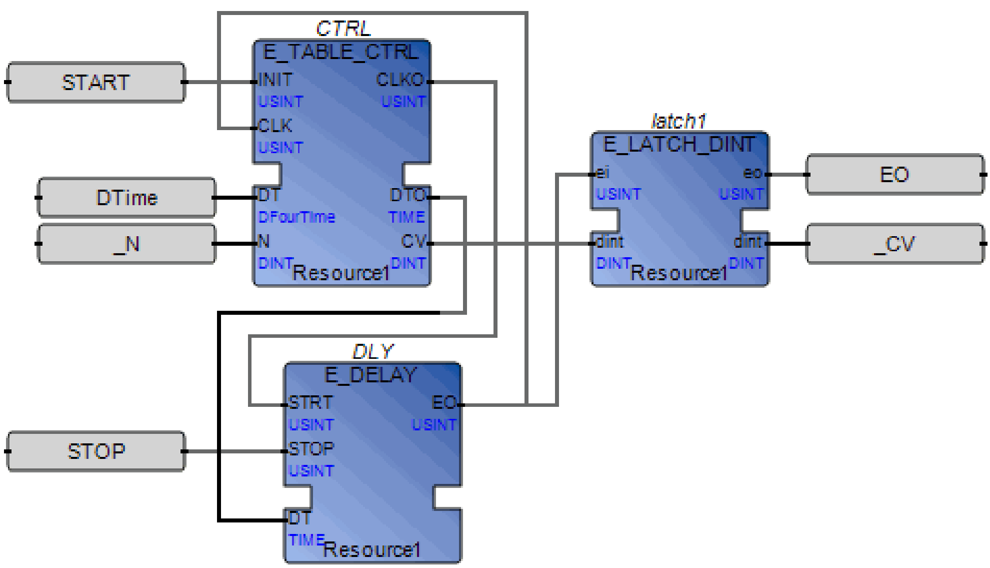
| Interface | ECC/Algorithms/Service sequences |
| --- | --- |



An event occurs at EO at an interval DT[0] after the occurrence of an event at EI. A second event occurs at an interval DT[1] after the first, etc., until N occurrences have been generated or an event occurs at the STOP input. The current event count is maintained at the CV output.

In this example implementation, N <= 4.

IEC 61499 FBD Definition



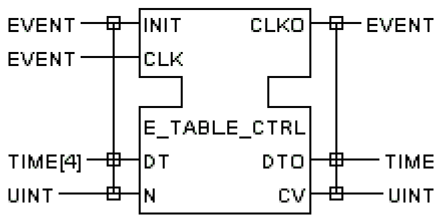
**Related Topics**

E_TABLE_CTRL

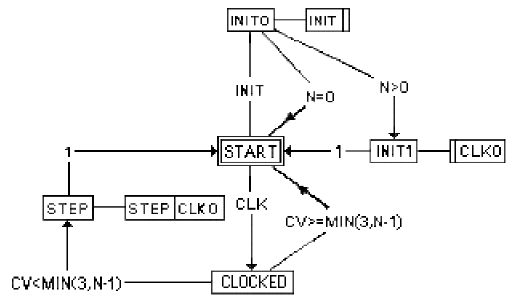
# E_TABLE_CTRL

**Generation of a finite train of events (table driven)**

Interface




ECC/Algorithms/Service sequences




This implementation using the E_TABLE_CTRL function block type is not a normative requirement. Equivalent functionality may be implemented by various means.

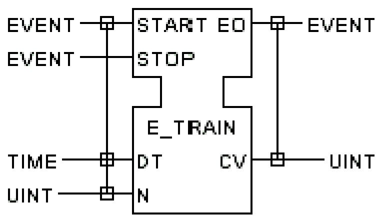
**Related Topics**

E_TABLE

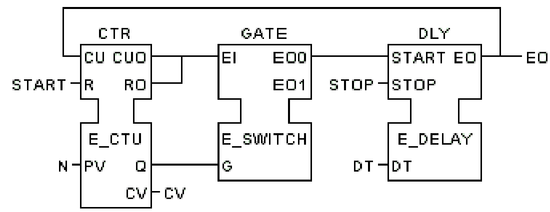
# E_TRAIN

**Generation of a finite train of events**

Interface

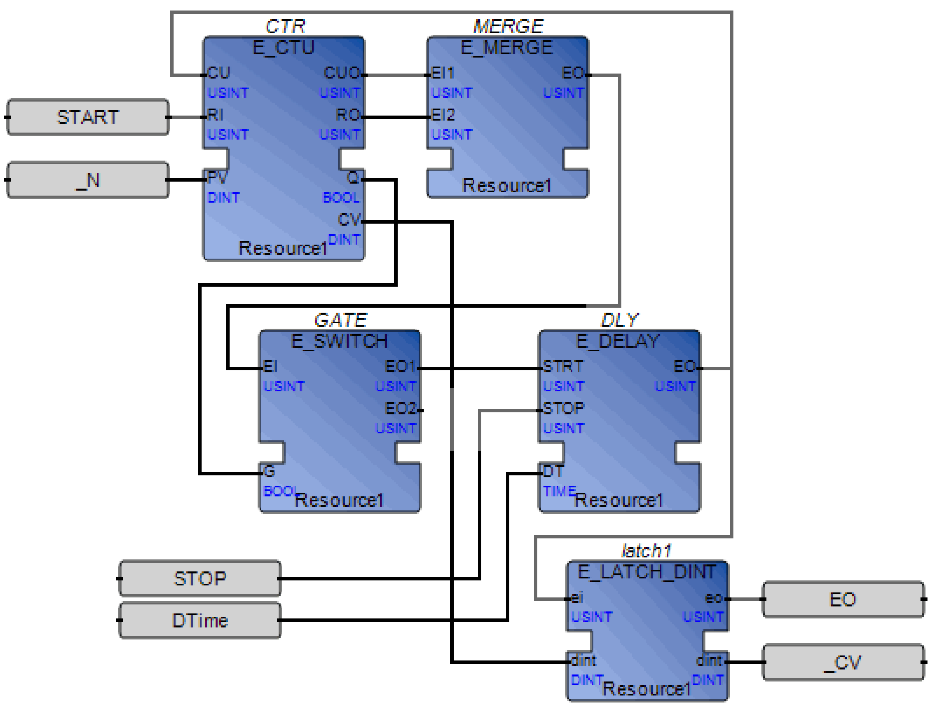



ECC/Algorithms/Service sequences




An event occurs at EO at an interval DT after the occurrence of an event at EI, and at intervals of DT thereafter, until N occurrences have been generated or an event occurs at the STOP input.

IEC 61499 FBD Definition



**Related Topics**

E_CTU
E_SWITCH
E_DELAY

## LocalEventInput

Normative function block arguments having the event input direction are automatically assigned an instance of the `LocalEventInput` function block. The `LocalEventInput` function block is defined as an IEC function block in the standard 61499 library.

```
LocalEventInput
{
  input SINT counter
  local  SINT LocalCounter;
  output BOOL Trigger;
If counter <> LocalCounter then
  LocalCounter = counter;
  Trigger = true;
Else
  Trigger = false;
End_if;
}
```

# Glossary

The glossary contains terms used in **ISaGRAF** and their definitions.

To optimize a search for a definition, click one of the following letter groups in which you want to search.

A - C D - H I - N O - R S - Z

**A - C**

| | |
|---|---|
| **AAM** | Abstract Automation Model. Common interfaces used to access Concrete Automation Model data represented by IEC 61131 and IEC 61499 elements and concepts, as well as Device Management |
| **Access Control** | The use of password-protection to control access to projects, devices, resources, POUs, and targets. For projects, devices, resources, and POUs, access control can also limit access to read mode. |
| **ACP** | Automation Collaborative Platform. A set of software components and services through which plug-ins communicate. |
| **Action** | A collection of operations to perform whose execution differs for each programming language. |
| **Add-in** | Also known as a plug-in, it is a utility, driver, or other software added to a primary application. In the Visual Studio Integrated Development Environment (IDE), an add-in is an Automation-based application that extends the capabilities of the IDE. |
| **Address** | Optional hexadecimal address freely defined for each variable. This address can be used by an external application to access the value of the variable when the resource is executed by the Target. |

| | |
|---|---|
| **Alias** | The property of a variable indicating a short name for a variable. For graphical programs, aliases indicate the parameters in functions and function blocks. |
| **ANY** | Overloaded data type. Enables overloading "C" function block inputs to support specified IEC 61131-3 data types as well as specified complex types such as arrays and structures. |
| **ANY_ELEMENTARY** | Overloaded data type enabling "C" function block inputs to support all of the IEC 61131-3 elementary data types. |
| **Application** | Built project using the Application Builder. |
| **Application Builder** | An integrated development environment used to build control applications, i.e. the workbench. |
| **Array** | Set of elements of the same type referenced by one or more indexes enclosed in square brackets and separated by commas. The index is an integer. Examples: tabi[2] or tabij[2,4]. |
| **Attribute** | The property of a variable indicating whether a variable is read, write, or read/write. |
| **Basic Function Block** | An IEC 61499 function block type using SFC execution control chart (ECC) elements to control the execution of steps/code. |
| | An IEC 61499 function block type using SFC elements to develop an execution control chart (ECC). |
| **Binding** | Bindings are directional links, i.e., access paths, between variables located in different resources. One variable is referred to as the producing variable and the other as the consuming variable. **ISaGRAF** enables external bindings between resources belonging to different projects. |
| **Binding Error Variable** | Variables that enable the management of binding errors at the consumer resource level. |
| **Boolean (BOOL)** | Basic type that can be used to define a variable, a Parameter (POU) or an I/O simple device. A Boolean can be TRUE (1) or FALSE (0). |

| | |
|---|---|
| **Boo Action** | A Boolean variable where the value corresponds to Step activity (0=inactive and 1=active). Possible qualifiers are Action (N), Reset (R), and Set (S).<br>See also Action |
| **Breakpoint** | A mark placed by the user at particular sections of the code. In Debug mode, the application stops when it encounters a breakpoint. Breakpoint implementation varies for each programming language. |
| **BYTE** | Unsigned integer 8-bit format. Basic type that can be used to define a variable, a Parameter (POU) or an I/O Device. |
| **CAM** | (Concrete Automation Model) Concrete project model allowing the usage of a device inside the ACF. Moreover, it may include AAM implementation for IEC 61131 concepts (Project data and POU body), AAM implementation for device management interfaces, compiler, wizard data and templates, deployment representation, and plug-ins specific to the CAM. |
| **C Function** | Function written with the "C" language, called from POUs, in a synchronous manner. |
| **C Language** | High level literal language used to access particularities of the target system. C language can be used to program C functions, function blocks and conversion functions. |
| **Call Stack** | Information which tracks stepping between POUs and called functions. Debug information includes call stack. You can only generate debug information for TIC POUs. |
| **Cell** | Elementary area of the graphic matrix for graphic languages or for the Dictionary. |
| **CFB** | Indicates a C function block |
| **CFU** | Indicates a C function |
| **Channel** | A channel of an I/O simple device represents a hardware I/O point. A channel is either an input or output. To enable use in POUs, variables including directly represented variables are connected to channels. |

| | |
|---|---|
| **Check In** | Sending the contents of **ISaGRAF** elements including projects, I/O devices, resources, and POUs for storage in a version source control database. Checked-in elements can be recovered at a later time. |
| **Child** | A program which is activated by its parent. The child program has only one parent. Only the parent can start or stop child program. A parent can have more than one child. |
| **Clearing a Transition** | The forcing of the clearing of a transition whether the latter is valid or not (i.e all previous steps are active or not). Tokens are moved and actions are executed as for a usual transition clearing. All tokens existing in the preceding steps are removed. A token is created in each of the following steps. |
| **CMG** | Short name for the Configuration Manager |
| **Coil** | A graphic component representing the assignment of an output or an internal variable. |
| **Common Scope** | Scope of a declaration applying to all POUs within a Project. (Only defined words and types can have common scope). |
| **Complex Equipment** | See I/O Complex Device |
| **Configuration** | See Device |
| **Configuration Manager** | (ConfigurationManager.exe) The executable file providing communication services between **ISaGRAF** and target. Responsible for launching, killing, and giving the status of running virtual machines. |
| **Connection** | The link between networks and devices. |
| **Constant Expression** | Literal expression used to describe a constant value. |
| **Contact** | Depending on the type of contact, a graphic component representing the value or function of an input or an internal variable. |
| **Contextual Menu** | Menu that is displayed under the mouse cursor by right-clicking the mouse. |
| **Conversion** | Filter attached to an input or output variable. The conversion is automatically applied each time the input variable is read or the output variable is refreshed. |

| | |
|---|---|
| **Conversion Function** | "C" written Function which describes a conversion. Such a conversion can be attached to any input or output, integer or real variable. |
| **CRC** | Cyclic redundancy checking |
| **Cross Reference Browser** | A tool that finds all references to variables, i.e., cross references, defined in the POUs of a project. The browser provides a total view of the declared variables in the programs of the project and where these are used. |
| **CSV File Format** | (Comma Separated Values) A delimited data format having each piece of information separated by commas and each line ending with a carriage return. The CSV file format can be used for importing or exporting variables data. |
| **Cycle** | The virtual machine executes the programs of a resource as a cycle. All programs of the resource are executed following the order defined by the user, from the first program to the last and again and again. Before the execution of the first program, inputs are read. After the execution of the last program, the outputs are refreshed. |
| **Cycle Timing** | The amount of time given to each resource cycle. If a cycle is completed within the cycle timing period, the system waits until this period has elapsed before starting a new cycle. The cycle consists of scanning the physical inputs of the process to drive, executing the POUs of the resource, then updating physical outputs. The cycle time can differ for each cycle when no cycle timing is specified. When the cycle timing is shorter, the virtual machine waits until this time has elapsed. When the cycle time is longer, the virtual machine immediately scans the inputs but signals with the "overflow" that the programmed time has been exceeded. When the trigger cycles property is false or the cycle time is 0, the virtual machine does not wait to start a new cycle. |
| **Cycle-to-cycle Mode** | Execution mode of a resource where cycles are executed one by one, according to the orders given by the user during debugging. Another execution mode for resources is real-time mode. |

| | |
|---|---|
| **Cyclic Program** | A time independent program that is executed during each cycle. |

## D - H

| | |
|---|---|
| **Database** | The collection of definitions making up a **ISaGRAF** project. The version source control feature stores checked-in information in a separate database. |
| **DATE** | The format of a date is year-month-day, separated by hyphens. Basic type that can be used to define a Variable, a Parameter (POU) or a Device. |
| **Debug Information** | For use when debugging using the step-by-step mode. Debug information includes call stack information which tracks stepping between POUs and called functions. You can only generate debug information for TIC POUs. |
| **Debugging** | The process of detecting defects in a project that includes setting and clearing breakpoints, step-by-step debugging, and cycle-to-cycle debugging. |
| **Declared Array** | A user-defined array defined as a data type. See also Undeclared Array |
| **Declared Instance (of a function block)** | A function block having assigned instances, i.e., declared in the dictionary. |
| **Defined Word** | Word that is an expression. This word can be used in POUs. At compiling time the word is replaced by the expression. A defined word can not use a defined word. |
| **Dependency (on a library)** | The state where a project uses, i.e., depends, on functions or function blocks defined in a library. |
| **Design (mode)** | An editing mode during which the Application Builder is not connected to the runtime module. |
| **Device** | A representation of the equipment, i.e., programmable logic controller, running the virtual machines.<br>See also Target |
| **Device Management** | Provides the communication infrastructure with the Run-time Engine. |

| | |
|---|---|
| **Dictionary** | The view displaying the variables, function and function block parameters, types, and defined words used in the programs of a Project. |
| **Dimension** | The size (number of elements) of an array. For example: [1..3,1..10] - represents a two-dimensional array containing a total of 30 elements. |
| **Direction** | Variables and I/O devices have a direction. For the property of a variable, direction indicates whether a variable is an input, output, or internal. The direction of an I/O device can be input or output. |
| **Directly Represented Variable** | A variable is generally declared before its use in one POU. Inputs and outputs can be used without any declaration respecting a defined syntax. It corresponds to direct represented variables. Example: %QX1.6, %ID8.2 |
| **Double Integer (DINT)** | Signed double integer 32-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| **Double Word (DWORD)** | Unsigned double word 32-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| **Driver** | See I/O Driver, Network Driver |
| **Dynamic Behavior** | Continuous and sequential execution of the steps and operations of a program during an execution cycle. |
| **Edge** | See Falling Edge, Rising Edge |
| **ETCP** | (ETCP.exe) **ISaGRAF** network driver that uses the TCP / IP stack. |
| **Execution Control Initial State (EC initial state)** | The execution control state that is active upon initialization of an execution control chart. |
| **Execution Control State (EC state)** | The situation in which the behavior of a basic function block with respect to its variables is determined by the algorithms associated with a specified set of execution control actions. |
| **Execution Control Transition (EC transition)** | The means by which control passes from a predecessor execution control state to a successor execution control state. |

| | |
|---|---|
| **Execution Mode** | The mode in which a resource is executed: real-time, cycle-to-cycle, and step-by-step. |
| **Expression** | Set of operators and identifiers. |
| **Failover Mechanism** | A redundant operational mode where a secondary hardware and software takes over when the primary system becomes unavailable. |
| **Falling Edge** | A falling edge of a boolean variable corresponds to a change from TRUE (1) to FALSE (0). |
| **FBD** | Function Block Diagram. Programming language. |
| **File Mode** | The mode where you save version source control information to a repository located on a local or remote computer.<br>See also Server Mode |
| **Function** | POU which has input parameters and one output parameter. A function can be called by a program, a function or a function block. A function has no instance. It means that local data are not stored, and are generally lost from one call to the other. |
| **Function Block** | POU which has input and output parameters and works on internal data (parameters). A program can call an instance of a function block. A function block instance cannot be called by a function (no internal data for a function). A function block can call another function block (instantiation mechanism is extended to the function blocks called). |
| **Global Scope** | Scope of a declaration applying to all POUs of one resource. |
| **Global Variable** | A variable whose scope is global. |
| **Hidden Parameter** | Input parameters of a function block that are not displayed in programs. |
| **Hierarchy** | Architecture of a Project, divided into several POUs. The hierarchy tree represents the links between parent programs and children programs.<br>See also Parent Program |

## I - N

| | |
|---|---|
| **Identifier** | Unique word used to represent a variable or a literal expression in the programming. |

| | |
|---|---|
| **IFB** | Indicates an IEC 61131 function block |
| **IFU** | Indicates an IEC 61131 function |
| **Initial Situation** | Set of the initial steps which represents the context of the program when it is started. |
| **Initial Step** | A Step that is activated when the program starts. |
| **Initial Value** | Value which has a variable when the virtual machine starts the execution of the resource. The initial value of a variable can be the default value, a value given by the user when the variable is defined or the value of the retain variable after the virtual machine has stopped. |
| **Input** | Direction of a variable or an I/O device. An input variable is connected to an input channel of an input device. |
| **Input Parameter** | Input argument of a function or a function block. These parameters can only be read by function or function block. A parameter is characterized by a type. |
| **Instance (of a Function Block)** | Copy of the internal data of a function block which persists from one call to the other. This word is used, by extension, to say that a program calls a function block instance and not the function block itself. |
| **Instruction** | An elementary operation of a program, entered on one line of text. |
| **Integer (INT)** | Signed integer 16-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| **Internal** | Attribute of a variable, which is not linked to an I/O device. Such a variable is called an internal variable. |
| **I/O Channel** | See Channel |

| | |
|---|---|
| **I/O Complex Device** | Element grouping several simple I/O devices. This provides the means for manufacturers to mix types and directions. The implementation of the I/O driver of an I/O complex device corresponds to the implementation of the drivers of all contained I/O simple devices. OEM parameters enable providing parameters to I/O complex devices. |
| **I/O Device** | Element grouping several channels of the same type and direction. These can be either an I/O Simple Device or an I/O Complex Device. |
| **I/O Driver** | "C" code which makes the interface between a virtual machine and the devices. The driver can be statically linked to the virtual machine or in a separate DLL (such as for the Windows NT target). Two types of drivers are available for use in **ISaGRAF**: generic and advanced. |
| **I/O Simple Device** | An I/O simple device corresponds to a piece of equipment having inputs or outputs, such as an I/O board.<br>OEM parameters enable providing parameters to I/O simple devices. Integrators define I/O simple devices. |
| **I/O Variable** | Variable connected to a channel of an I/O device. An array can be connected to an I/O device if all elements are connected to contiguous channels, the type of the array must be the same type as the I/O device. |
| **I/O Wiring** | Definition of the links between the variables of the Project and the channels of the I/O devices existing on the Target system. |
| **ISaRSI** | (IsaRSI.exe) Enhanced serial port driver. The network driver that provides communication with **ISaGRAF** on a serial port. Similar to ETCP. |
| **ITA** | Indicates an array |
| **ITS** | Indicates a structure |
| **IXLSma Server** | (IxlSmaServer.exe) Provides service for performing IXL read operations, using the HSD driver with the SMA method. This method is independent from the virtual machine cycle and is thus faster. |
| **Keyword** | Reserved identifier of the language. |

| | |
|---|---|
| **Label** | The identifier for an instruction within a program. Labels can also be used for jump operations. |
| **Language Container** | A workspace enabling the development of graphic or textual POUs programmed using one of the available programming languages. Individual language containers can only use one programming language. When editing a container, the toolbox displays the corresponding elements for the specific programming language. The multi-language editor (MLGE) enables the creation of language containers. |
| **LD** | Ladder Diagram. Programming language. |
| **LD Action** | An action where you program an LD diagram in the level 2 window of and SFC program or basic IEC 61499 function block. Possible qualifiers are Action (N), Reset (R), Set (S), Pulse on Deactivation, and Pulse on Activation.<br>See also Action |
| **Library** | Special projects made up of devices and resources in which you define functions and function blocks for reuse throughout **ISaGRAF** projects. Libraries also enable you to modularize projects and to isolate functions and function blocks so that these can be validated separately. |
| **Link** | A graphic component connecting elements in a diagram. |
| **Literal** | A lexical unit that directly represents a value. |
| **Local Scope** | Scope of a declaration applying to only one POU. |
| **Locked I/O** | Input or output variable, disconnected logically from the corresponding I/O device, by a "lock" command sent by the user from the debugger. |
| **Long Integer (LINT)** | Signed integer 64-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| **Long Real (LREAL)** | Type of a variable, stored in a floating IEEE double precision 64-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| **Long Word (LWORD)** | Unsigned long word 64-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |

| | |
|---|---|
| **Maximum time** | Time of the longest cycle since the virtual machine has started the execution of the programs of a resource. |
| **Memory for Retain** | Run-time setting for a resource indicating the location where retained values are stored (the required syntax depends on the implementation). |
| **Message** | See STRING |
| **MLGE** | Multi-language Editor. |
| **Monitoring** | A process by which the user views virtual machine running states, system events, target capability, network card status and various online statistics in a read format. |
| **MSI** | Windows installers (.msi) used to install applications and files typically used by the end user of the application. |
| **Network** | The term network is used in different contexts:<br>- The means of communication between the target platform and their clients.<br>- For the execution order of graphic programs, a sequence of connected blocks.<br><br>See also Sub-network |
| **Network Driver** | "C" code which makes the interface between the Target network layer and the network. |
| **Non-stored Action** | A list of statements, executed at each Target cycle, when the corresponding step is active. |

## O - R

| | |
|---|---|
| **OEM** | Original Equipment Manufacturer |
| **OEM Parameter** | Parameters attached to an IO device. A parameter is characterized by a type. An OEM parameter is defined by the designer of the device. It can be a constant, or a variable parameter entered by the user during the I/O connection. |
| **Online Mode** | Mode in which the Application Builder is connected to a target enabling target management, monitoring and debugging. |
| **Operator** | Basic logical operation such as arithmetic, boolean, comparator, and data conversion. |

| | |
|---|---|
| **Output** | Direction of a variable or an I/O device. An output variable is connected to an output channel of an output device. |
| **Output Parameter** | Output argument of a function or function block. These parameters can only be written by a function or function block. A function has only one output parameter. A parameter is characterized by a type. |
| **Overflow** | Integer value which corresponds to the number of times the cycle time has been exceeded. Always 0, if cycle time is 0. |
| **Overloading** | Overloading a "C" function block input enables a function block call to perform various tasks depending on the context. The ANY and ANY_ELEMENTARY data types enable overloading. |
| **Package** | The Target Definition Builder enables OEMs to provide packages containing the drivers of several I/O devices and/or "C" functions and function blocks available for a specific target. |
| **Parameter (POU)** | See Input Parameter, Output Parameter, OEM Parameter, and Hidden Parameter |
| **Parent Program** | A program which controls other programs, called its children. See also Child |
| **Platform Builder** | Defines, configures and generates the source code making of the runtime engine. The target will have the proper source code, the PLC definition will be use by the application builder to limit or add features, and to generate a report containing a description of the final product contents. Custom functions, function blocks, extended data types, field bus drivers, and comments can be added with the platform builder. |
| **PLC** | Programmable Logic Controller |
| **Plug-in** | A module or package integrated into a bigger platform that enables the extension of the application. |
| **POU** | Program Organization Unit: set of instructions that are programs, a functions or function blocks. |
| **Power Rail** | Main left and right vertical rails at the extremities of a ladder diagram. |

**Primary Device** For failover mechanisms, the device having the previously downloaded application. By default, this device remains active until a failure when the standby secondary device takes over.

**Program** See POU. A program belongs to a resource. It is executed by the virtual machine, depending on its location (order) in the resource.

**Project** Set of devices and links between their resources.

**Project Updater** A program allowing to convert projects developed using previous versions for use within the latest version. Each time you upgrade to a newer version, you need to update projects.

**PROPI** PROPI is an interface enabling you to send commands directly to **ISaGRAF** via a custom application. For instance, you could use the PROPI interface when using **ISaGRAF** in the background.

**Pulse Action** A list of statements executed only once when the corresponding step is activated.

**Qualifier** Determines the way the action of a step is executed. The qualifier can be N, S, R, P0 or P1.

**Real** Type of a variable, stored in a floating IEEE single precision 32-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device.

**Real Device** I/O device physically connected to an I/O driver on the target. See also Virtual Device

**Real-time Mode** The run time normal execution mode of a resource where target cycles are triggered by the cycle timing. Another execution mode for resources is cycle-to-cycle mode.

**Resource** The POUs and definitions making up a virtual machine.

**Resource Name** The unique identifier of a resource within a device.

**Retain** Attribute of a variable. The value of a retain variable is saved by the virtual machine at each cycle. The value stored is restored if the virtual machine stops and restarts.

| | |
|---|---|
| **Return** | Graphic component of a program representing the conditional end of a program. |
| **Return Parameter** | See Output Parameter |
| **Rising Edge** | A rising edge of a Boolean variable corresponds to a change from FALSE (0) to TRUE (1). |
| **Rung** | Graphic component of a program representing a group of circuit elements leading to the activation of a coil in an LD diagram. |
| **Run-time Engine** | Solves application logic and drive I/O points. This portable engine features modular architecture. |
| **Run-time Error** | Application error detected by the Target system at run time. |

## S - Z

| | |
|---|---|
| **Scope** | See Global Scope, Common Scope, Local Scope |
| **Secondary Device** | For failover mechanisms, the device having the duplicated application. By default, this device remains on standby until the primary active device fails. |
| **Section** | Program, function and function block sections are where are localized POU of a resource. POUs located in the Program section are executed by the virtual machine. |
| **Security State** | The indication of the level of access control that is applied to a resource, a POU, or a target. |
| **Selection List** | Also known as a 'combo-box'.<br><br>When a selection list is provided for a particular cell, clicking on its right part (down arrow), displays the available choices. To make a selection, perform one of the following operations:<br>- click on the item (use the scroll bar first if the required choice is not visible)<br>- move in the list using the cursor keys and press Enter<br>- type the first letter (if more than one item starts with this letter, press the letter again to select the next occurrence). |
| **Separator** | Special character (or group of characters) used to separate the identifiers in a literal language. |

| | |
|---|---|
| **Sequential Program** | A program that is executed according to the dynamic behavior of the programming language and where the time variable explicitly synchronizes operations. |
| **Server** | Part of the target that receives requests from IXL to retrieve information about the resource run by the virtual machine. |
| **Server Mode** | (Client/server mode) The mode where you save version source control information in a server repository. Before using this mode, you need to set up the repository server and connect with the server.<br>See also File Mode |
| **SFB** | Indicates a function block |
| **SFC** | Sequential Function Chart. Programming language. |
| **SFC Action** | An action with an associated SFC child program. Possible qualifiers are Action (N), Reset (R), and Set (S).<br>See also Action |
| **SFU** | Indicates a function |
| **Shape** | The spatial form or appearance of an object. |
| **Short Integer (SINT)** | Signed integer 8-bit format. Basic type that can be used to define a Variable, a Parameter (POU) or a Device. |
| **Simulation Mode** | Mode in which virtual machines execute the code of individual resources and the Windows platform performs aspects such as POU execution. |
| **SIT** | Indicates a Standard IEC 61131 type. |
| **Solution Explorer** | A view with a tree-like structure enabling the management of items such as devices, programs, functions, function blocks and dictionaries. |
| **ST** | Structured Text. Programming language. |
| **ST Action** | An action where you define ST code in the level 2 window of and SFC program or basic IEC 61499 function block. Possible qualifiers are Action (N), Reset (R), Set (S), Pulse on Deactivation, and Pulse on Activation.<br>See also Action |

| | |
|---|---|
| **Standard IEC 61131 Types** | Boolean (Bool), Short Integer (SINT), Unsigned Short Integer (USINT), BYTE, Integer (INT), Unsigned Integer (UINT), WORD, Double Integer (DINT), Unsigned Double Integer (UDINT), Double Word (DWORD), Long Integer (LINT), Unsigned Long Integer (ULINT), Long Word (LWORD), Real, Long Real (LREAL), TIME, DATE, STRING.<br>See also Type |
| **Statement** | Basic ST complete operation. |
| **Step** | A basic graphic component representing a steady situation of the process. A step is referenced by a name. The activity of a step is used to control the execution of the corresponding actions.<br>See also Action |
| **Step-by-step Mode** | A mode used while debugging POUs where you set breakpoints at specific lines of code or rungs causing the application to stop when reached. |
| **STRING** | Character string. Basic type that can be used to define a Variable, a Parameter (POU) or a Device. |
| **Structure** | Corresponds to a type which has previously been specified to be a data structure, i.e. a type consisting of a collection of named elements (or fields). Each field can be a basic type, a basic structured type, a structure or an array. A field of a variable with a structure type is accessible using the following syntax: VarName.a, VarName.b[3], VarName.c.d |
| **Sub-network** | For the execution order, a sequence of blocks encapsulated by a region element.<br><br>See also Network |
| **Sub-program** | A program called by a Parent Program. A sub-program is also called a Child program. To call sub-programs written in another language, use a function. A function can be called by any POU. |

| | |
|---|---|
| **Symbol Table** | The file corresponding to the variables and function blocks defined for a resource. This file is downloaded onto the target. The symbol table is set to one of two formats: complete table or reduced table. The complete table contains all defined variables, whereas, the reduced symbol table only contains the names of variables having a defined Address cell. |
| **Symbols Monitoring Information** | When debugging or simulating, code required to enable graphically displaying the output values of functions and operators in graphical programs. |
| **System Events** | Log of execution events occuring on the target platform. |
| **System Variable** | System variables hold the current values of all system variables for a resource. You can read from or write to system variables. These variables are defined in the dsys0def.h file. For example, the current cycle time is a system variable that can only be read by a program. |
| **Target** | The hardware platform onto which you download an application.<br>See also Device |
| **Target Definition Builder** | The Target Definition Builder enables the description of targets (main definition and options of the embedded software), complex data types (such as defined in IEC languages), "C" functions, function blocks and conversion functions, and I/O devices or network drivers for IXL communication. |
| **Target Management** | Operations that control the application of a target including downloading, uploading, starting and stopping resources, and performing online changes. |
| **TIC Code** | Target Independent Code produced by the **ISaGRAF** compiler for execution on virtual machines. |
| **Timer (TIME)** | Unit of a timer is the millisecond. Basic type that can be used to define a Variable, a Parameter (POU) or a Device. |
| **Token (SFC)** | Graphical marker used to show the active steps of an SFC program. |

| | |
|---|---|
| **Toolbox** | The utility containing the elements and shapes available for language and ISaVIEW containers. For language containers, the available elements differ for the individual programming languages. |
| **Tool Window** | A standard Microsoft Windows control that enables application creation and editing. |
| **Top Level Program** | Program put at the top of the hierarchy tree. A top level program is activated by the system.<br>See also Parent Program |
| **Transition** | A basic graphic component representing the condition between different steps. A transition is referenced by a name. A Boolean condition is attached to each transition. |
| **Trigger Cycles** | Resource property indicating whether a resource cycle executes according to a defined cycle timing. |
| **Type** | Data types are defined for many items in **ISaGRAF** projects:<br>- variables<br>- function or function block parameters<br>- I/O simple devices<br>See also Standard IEC 61131 Types, User Types |
| **Undeclared Array** | An undeclared array is defined as a variable in a dictionary instance. See also Declared Array |
| **Unsigned Double Integer (UDINT)** | Unsigned double integer 32-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| **Unsigned Integer (UINT)** | Unsigned integer 16-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| **Unsigned Long Integer (ULINT)** | Unsigned integer 64-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |
| **Unsigned Short Integer (USINT)** | Unsigned integer 8-bit format. Basic type that can be used to define a Variable, a Parameter (POU) or a Device. |
| **User Data** | User Data are any data of any format (file, list of values) which have to be merged with the generated code of the resource in order to download them into the target PLC. Such data are not directly operated by the virtual machine and is commonly dedicated to other software installed on the target PLC. |

| | |
|---|---|
| **User Types** | Types that the user can define using basic types or other user types. User types can be arrays or structures. |
| **User-Defined Function Block** | A custom function block. You create user-defined function blocks in the Function Blocks section for a resource. |
| **Validity of a Transition** | Attribute of a Transition. A transition is validated (or enabled) when all the preceding steps are active. |
| **Variable** | Unique identifier of elementary data which is used in the programs of a Project. |
| **Variable Group** | Grouping of variables enabling managing and logically sorting these within a resource. Variable groups are displayed in the dictionary's variables tree. |
| **Variable Name** | A unique identifier, defined in **ISaGRAF**, for a storage location containing information used in exchanges between resources. |
| **Version Source Control** | A tool that manages the changing versions of **ISaGRAF** elements including projects, I/O devices, resources, and POUs by saving them to a version source control database. Saving these elements to a control database enables you to retrieve older versions of the elements at a later time. |
| **Virtual Device** | I/O device which is not physically connected to an I/O driver on the target.<br>See also Real Device |
| **Virtual Machine** | (IsaVM.exe) The operating system process or thread that executes the previously downloaded application. |
| **VS2008** | Microsoft Visual Studio 2008. |
| **Wiring** | The property of a variable indicating the I/O channel to which the variable is wired. |
| **WORD** | Unsigned word 16-bit format. Basic type that can be used to define a variable, a Parameter (POU) or a Device. |

| | |
|---|---|
| **Workstation** | A communication pathway to external systems and third party tools connected across the field bus. |
| **Zip Source** | An exchange file of 7-Zip (.7z) compressed format containing XML files for exported **ISaGRAF** elements. From the compilation options for a resource, you can choose to embed a zip source file on the target. This source file can be uploaded from the target at a later time. |

# Licensing

**ISaGRAF** enables the creation of virtual machines running on hardware components, called targets.

There are three types of software licenses available for **ISaGRAF**:

- Free version, delivered with the product and available for testing the product. This version enables using only the ISaFREE_TPL project template with the ISAFREE-TGT target and projects can have a maximum size of 3200 bytes.

- Integrated license, included in the installation of the **ISaGRAF** software. The product is licensed upon installation. The Integrated license is available as a Full license or a Limited license. A Full license is a fully operational version of the product while a Limited license can only have one device.

- Engineering license, obtained by manually activating an unlicensed version of the product. The Engineering license is available as a Full license or a Limited license. A Full license is a fully operational version of the product while a Limited license can only have one device.

The Integrated and Engineering licenses are available for the following activation periods:

- Lifetime (does not expire)
- 1 month
- 6 months
- 12 months

**To access Licensing**

1. From the Help menu, click **Licensing CAM 5.**

The Licensing for the **ISaGRAF 5** Concrete Automation Model is displayed.

**To obtain an authorized Engineering license**

1. From the Help menu, click **Licensing CAM 5.**

   The Licensing for the **ISaGRAF 5** Concrete Automation Model is displayed along with three User Codes.

2. Send an e-mail containing the desired activation period and the three User Codes to the support team:
   support@ISaGRAF.com

3. The support team will email you back Registration Keys 1 and 2.

4. Insert the Registration Keys in their appropriate regions and click **Validate**.

   **ISaGRAF** is now licensed.

**To remove an authorized license**

1. From the Help menu, click **Licensing CAM 5.**

   The Licensing for the **ISaGRAF 5** Concrete Automation Model is displayed along with three User Codes.

2. Send an email containing the three User Codes to the support team:
   support@ISaGRAF.com

3. The support team will email you back Registration Keys 1 and 2.

4. Insert the Registration Keys in their appropriate regions and click **Validate**.

   A confirmation code appears.

5. Send an email containing the confirmation code to the support team:
   support@ISaGRAF.com

   **ISaGRAF** is no longer licensed.

# Windows Runtime Modules

Windows run-time modules enable your control applications, developed with **ISaGRAF** on the development platform, to execute on Windows® 7 or Windows® 8 target platforms. For these operating systems, both the development platform and target platform can be the same. The run-time modules form the containers into which the applications you build are deployed. The Windows runtime modules for **ISaGRAF 5.5x** support the following additional features:

The IEC 61850 standard for electrical substation automation

For basic and composite IEC 61499 function blocks, enable performing the following online changes:

- Add, delete, rename, and reorder programs

- Add, delete, and rename steps and transitions, as well as modify initial steps or flow between elements

- Add, delete, and move actions blocks within steps of SFC programs. You can also change the qualifier for action blocks.

- Add, delete, and move function blocks

- Add and delete function block instances

- Modify and rename user-defined functions and function blocks

- Add, delete, and modify the parameters of user-defined functions and function blocks

A Failover mechanism enabling the duplication of all resources belonging to a device where these are attached to a second device running on a mirror target, completely independent of the other.

The definition of the resource cycle time in microseconds, using a decimal point, for the QNX 6, Linux, VXWorks, OS Mono, and OS Multi targets.

Virtual machines consider possible cycle time drift when calculating hard real-time.

Searches on targets for C functions and functions blocks, I/O drivers, and conversion functions

The Windows runtime module is available in the following format:

- ISAFREE_TGT, a multitask implementation where the virtual address is coded on 32 bits but the code will then increase 50 percent

# ISAFREE-TGT

The ISAFREE_TGT target uses the following files on the Windows® 7 or Windows® 8 platforms:

| File | Description |
|---|---|
| ISaGRAF.exe | Configuration Manager. This is the file to be launched (manually or during start-up) on your system in order to communicate with the workbench. This task is responsible for launching, killing and giving the state of the Kernels running on your system.<br><br>ISaGRAF will automatically launch ETCP. |
| IsaVM.exe | Kernel or Virtual Machine. This is the "task" that executes the resource code. |
| IsaRSI.exe | The network driver used when developing an IXL client using serial communication with the workbench. IsaRSI also manages the communication between local Kernels and other remote Kernels . In this case, The IsaRSI driver emulates the behavior of a field-bus. We can speak of IsaRSI as a virtual field-bus. |
| ETCP.exe | Enhanced TCP/IP protocol. Optional. The network driver used for the communication with the workbench on Ethernet. In this case, The ETCP driver emulates the behavior of a field-bus. We can speak of ETCP as a virtual field-bus. |
| ISaIXL.dll | This *.dll file appears only for the Windows and RTX targets. On other systems (VxWorks, QNX, and Linux), this is statically linked to the above components and does not appear in this directory. This corresponds to the services used to communicate with Kernels. |
| ISaSys.dll | This *.dll file appears only for the Windows and RTX targets. On other systems (VxWorks, QNX, and Linux), this is statically linked to the above components. This corresponds to the system layer. It is this software layer that is ported by integrators to port the Kernel on other systems. |
| IsaAFB.dll | This *.dll file appears only for the Windows target. On other systems (VxWorks, QNX, Linux, and RTX), this *.dll is statically linked to the above components. Contains the advanced control functions and function blocks, including C functions and C function blocks |

| File | Description |
|---|---|
| IsaSER.dll | This *.dll file appears only for the Windows target. On other systems (VxWorks, QNX, Linux, and RTX), this *.dll is statically linked to the above components. contains serial communnication (TCP or RS232) C functions. |
| IsaNDT.dll | This *.dll file appears only for the Windows target. On other systems (VxWorks, QNX, Linux, and RTX), this *.dll is statically linked to the above components. Contains arithmetic and bit operation C functions used with data types such as BYTE, WORD, DWORD, LWORD, and LREAL. |
| ModbusTcpClient.dll | This *.dll file appears only for the Windows target. On other systems (QNX, Linux), this *.dll file is statically linked to the above components. Contains a Modbus TCP/IP IO driver (client). |
| ModbusTcoServer.dll | This *.dll file appears only for the Windows target. On other systems (QNX, Linux), this *.dll file is statically linked to the above components. Contains a Modbus TCP/IP IO driver (server) |
| isamsg.fcr | Text file describing messages of errors that may occur in the Kernel during the execution of resource code. |
| isamsg.wng | Text file describing messages of warnings that may occur in the Kernel during the execution of resource code. |
| isawnt.fcr | Text file describing messages of errors that may occur in the standard Kernel during the execution of resource code specific to the OS. |
| isawnt.wng | Text file describing messages of warnings that may occur in the standard Kernel during the execution of resource code specific to the OS. |

# Target Features

The following target features are available for the **ISaGRAF 5.5x** ISAFREE-TGT target:

| Feature | Description |
| --- | --- |
| Memory Size | The memory size of the target. |
| Enhanced Target | Enables the use of **ISaGRAF 5.30** enhanced features. Possible values are True or False. |
| Password | Enables supporting passwords on a target. This prevents any user from connecting to the target without the proper password. Possible values are True or False. |
| Ladder Diagram Optimized Code | Enables optimizing the code generated in the LD language to improve performance. Possible values are True or False. |
| Binding | Enables communication between targets through bindings. Possible values are True or False. |
| Multiple Resources | Enables setting the maximum number of resources supported by the target. Only available for multi-task porting. Possible values are True or False. |
| Online Change | Enables online modifications during the execution of a user application. Possible values are True or False. |
| Retain | Enables retaining variable values so that each retained variable persists through time. Possible values are True or False. |
| Micro Cycle Time | Enables microsecond precision for cycle time on the target instead of millisecond precision. Possible values are True or False. |
| Interrupts | Enables the support of time-based or hardware-based user interrupts. Possible values are True or False. |
| Flexible array and FB parameters by reference | Enables passing input parameters and flexible array parameters by reference to "C" function blocks. Possible values are True or False. |
| POU TIC greater than 64 KB | Removes the limitation of 64 KB per POU and is only available for large memory targets. Possible values are True or False. |

| Feature | Description |
|---|---|
| Binding Network Instances | Enables the capacity to instantiate multiple networks (for example enables using and configuring multiple ETCP networks). Possible values are True or False. |
| SFC transition priority | Enables supporting user-defined priorities for the parallel branches of transitions. Possible values are True or False. |
| Wiring on complex variable members | Enables connecting a member of a complex variable to an I/O board channel. Possible values are True or False. |
| IO Device channel OEM parameters | Enables defining OEM parameters on I/O channels. Possible values are True or False. |
| Online change support for initialization of C FB instances | Enables the target to support the addition or removal of "C" function block instances having initialization or exit functionality when performing online changes. Adding or removing such function blocks while performing online changes may impact cycle time or RAM consumption. Possible values are True or False. |
| Partial access of ANY_BIT variables | Enables read and write access on integer-type variables and sub-variables using specific TICs. Possible values are True or False. |

**Note:** Existing projects may contain older target versions not supporting all of these target features.

**To view the target features**

You cannot modify the ISAFREE-TGT target features.

1. In the Solution Explorer, right-click the device, and then click **Open**.

2. From the Device View, in the breadcrumbs trail, click  and select **Target Features**.

The target features are displayed in the Device View.

# Installing Windows Run-time Modules

You install Windows runtime modules by copying them, then pasting them onto the target computer running either the Windows® 7 or Windows® 8 operating systems.

**Note:** When installing a run-time module, make sure that the complete path for the directory contains no dashes.

### Startup Parameters

You can specify startup parameters as entries in an initialization file (.ini) or in a command line. When defining parameters, the initialization file name must match the executable file name. For example, for the executable ISaGRAF.exe, you name the initialization file "ISaGRAF.ini".

| Component Name | Parameter | Description |
|---|---|---|
| **[ETCP]** | `CruCnxTimeOut` | Timeout for connection.<br>The default value is 1.5s (1500 ms). |
| | `SockCruPortId` | CRU server TCP/IP port.<br>The default value is 1131. |
| | `ChNbr` | CRU server Number of channels.<br>The default value is 72. |
| | `SockVruPortId` | VRU server TCP/IP port.<br>The default value is 1113. |
| | `RctNb` | VRU Resource connection table size, the number of remote producing resources for local ETCP task,i.e., the number of binding links pointing to the configuration.<br>The default value is 100, meaning that an ETCP task can manage up to 100 remote producing resources. |

| | |
|---|---|
| `IsctNb` | VRU Exported socket connection table size, the number of remote producing resources. However, each producing resource must only be counted once if it produces to several local resources.<br>The default value is 100, meaning that an ETCP task can manage up to 100 remote producers for all of its local consumers. |
| `EsctNb` | VRU Resource binding information table, the number of local producing resources (resources producing to distant platforms only) plus the number of distant consuming resources:<br>- if both R1 and R2 on C1 consume data from R3 on C2, there will only be one connection between C1 and C2<br>- if R1 on C1 consumes data from R2 and R3 on C2, there will be two connections between C1 and C2<br><br>EsctNb is also the number of binding links pointing out of the configuration plus the number of local producing resources (resource producing to distant platforms only).<br>The default value is 100 meaning that an ETCP task can manage 100 remote connections minus the number of local producing resources. |
| `RBitNb` | VRU Resource binding information table, host data concerning each remote producer This parameter value should be the same as IsctNb value because an ISCT entry is used jointly with a RBIT entry.<br>The default value is 100. |

| | |
|---|---|
| `NCRBSize` | VRU None Converted reception buffer size, the total amount of bound bytes the ETCP task has to buffer for all remote producers Each producer has a 12-byte header.<br>The default value is 512 meaning that the ETCP task can handle 500 bound bytes for each remote producer. |
| `NbrSend` | VRU Number of emission.<br>The default value is 1. |
| `Cycle` | ETCP server cycle<br>The value of this parameter is in milliseconds.<br>The default value is 1 ms. |
| `PingTimeOut` | ETCP Ping time-out<br>The value of this parameter is in milliseconds<br>The default value is 60000 ms. |
| `NbIxlClt` | ETCP Number of IXL Client<br>The default value is 3. |
| `KVBETCP` | Mutex time out for management of shared memory<br>The default value is 1000 ms. |
| `SockMaxNbPendingCnx` | Max nb of pending cnx<br>The default is system dependant. |
| `SockTcpNoDelay` | Naggle algorithm<br>The default is disabled to provide better lattency times.<br>The default is Yes. |
| `SockGenKeepAlive` | Generation of keep alive packet<br>By default, parameter does not generate keep alives. |
| `SockSendBuffSize` | Socket send buffer size<br>The default is 0. |
| `SockRecvBuffSize` | Socket receive buffer size<br>The default is 0. |
| `ConsNb` | Remote configuration consumer number<br>The default is 100. |

| | | |
|---|---|---|
| | `BindingPort` | The port number.<br>The default is 1113. |
| **[HSD]** | `MaxMsgConnect` | Maximum msg. in the cnx msg queue<br>The default value is 3. |
| | `RcvMaxMsg` | Maximum number of message in message queue<br>The default value is 3. |
| | `SendMaxMsg` | Maximum number of message in message queue<br>The default value is 3. |
| | `SndSz` | Size of exchanged messages<br>The default value is 16416 bytes (.ISA_IXL_BUFCTSSZ + 2*ISA_IXL_MSGPROC_HDRSZ). |
| | `RcvSz` | Size of exchanged messages<br>The default value is 32800 bytes (ISA_IXL_BUFSTCSZ + 2*ISA_IXL_MSGPROC_HDRSZ). |
| | `TimeOut` | The value of this parameter is in milliseconds.<br>The default value is 5000 ms. |
| | `NtfSignal` | Startup parameter for notification management (signal)<br>The default value is 256. |
| | `Priority` | Priority of the cnx (0 High - 255 Low)<br>The default value is 10. |
| | `SemTimeOut` | Mutex time out for management of shared memory<br>The value of this parameter is in milliseconds.<br>The default value is 11 ms. |
| **[APP]** | `stgMode` | Configuration manager starting mode<br>The default value is 1. /* Automatic restoration */ |
| | `CycleTimeMin` | Minimum cycle time for this task<br>The value of this parameter is in milliseconds.<br>The default value is 10 ms. |

| | | |
|---|---|---|
| | `CycleTimeMax` | Maximum cycle time for this task<br>The value of this parameter is in milliseconds.<br>The default value is 50 ms. |
| | `ResNbr` | Number of resource on one config<br>The default value is 8. |
| | `MisNbr` | Maximum number of miscellaneous tasks<br>The default value is 8. |
| | `s` | Get number of resource to start |
| | `ETCP` | Extra startup parameters<br>No default value defined. |
| | `SyncTime` | Wait synchronization<br>The value of this parameter is in milliseconds.<br>The default value is 5000 ms. |
| | `RSI` | IsaRSI startup parameters.<br>No default value defined. |
| | `PrjPath` | Set project path<br>The default value is the path to the folder containing the ISaGRAF.exe. |
| | `WngPeriod` | Get the warning period<br>The value of this parameter is in milliseconds.<br>The default value is 60000 ms. |
| | `NOETCP` | Disables ETCP<br>The default is ETCP active. |
| **[IXL]** | `SndSz` | Size of exchanged messages<br>The default value is 16416 bytes (ISA_IXL_BUFCTSSZ + 2*ISA_IXL_MSGPROC_HDRSZ). |
| | `RcvSz` | Size of exchanged messages<br>The default value is 32800 bytes (ISA_IXL_BUFSTCSZ + 2*ISA_IXL_MSGPROC_HDRSZ). |
| | `NtfSignal` | Startup parameter for notification management (signal)<br>The default value is 256. |

| | | |
|---|---|---|
| | `DrvNbr` | Maximum number of drivers<br>The default value is 5. |
| | `CnxNbr` | Maximum number of connections<br>This is the number of connections between the IXD (exchange dispatcher) to the resource and the configuration manager. It opens 2 connection per resource + 1 for the CMG..<br>The default value is 24. |
| **[IXS]** | `DrvNbr` | Maximum number of drivers<br>The default value is 5. |
| | `CnxNbr` | Maximum number of connections<br>The default value is 24. |
| **[ISXLETCP]** | `IpcMsgLength` | Message size in IPC queues<br>The default value is 1024. |
| | `IpcMsgNbr` | Number of messages in IPC queues<br>The default value is 5. |
| | `SidMessageLength` | ISXL notified method emission & reception message buffer<br>The default value is 2048. |
| | `NtfSignal` | Startup parameter for notification management (signal)<br>The default value is 256. |
| | `NtfMessageLength` | ISXL notified method reception message buffer<br>The default value is 2048. |
| | `SignalCode` | Max message queue size in-between ETCP client-server<br>The default value is 0. |
| **[ISXLRSI]** | `IpcMsgLength` | Message size in IPC queues<br>The default value is 1024. |
| | `IpcMsgNbr` | Number of messages in IPC queues<br>The default value is 5. |

| | | |
|---|---|---|
| | `SidMessageLength` | ISXL notified method emission & reception message buffer<br>The default value is 2048. |
| | `NtfSignal` | Startup parameter for notification management (signal)<br>The default value is 256. |
| | `NtfMessageLength` | ISXL notified method reception message buffer<br>The default value is 2048. |
| | `SignalCode` | Max message queue size in-between ETCP client-server<br>The default value is 0. |
| **[IXD]** | `ChNbr` | Maximum number of connections<br>The default value is 24. |
| | `DataMsgSize` | Size of buffer for message processing<br>The default value is default is 32800 bytes (ISA_GETMAX(ISA_IXL_MSGPROC_BUFRCVSZ, ISA_IXL_MSGPROC_BUFSNDSZ)). |
| | `VarDescNbr` | Number of variables description<br>The default value is default is 3283 ((ISA_GETMIN(ISA_IXL_MSGPROC_BUFRCVSZ, ISA_IXL_MSGPROC_BUFSNDSZ)) / 5). |
| | `MsgByChannel` | Pending message number per connection<br>The default value is 4. |
| | `IxlTimeout` | Ixl Timeout<br>The value of this parameter is in milliseconds.<br>The default value is 11000 ms. |
| | `IxsTimeout` | Ixs Timeout<br>The value of this parameter is in milliseconds<br>The default value is 11000 ms. |
| | `CnxTimeLoop` | Connection time loop<br>The value of this parameter is in milliseconds.<br>The default value is 1000 ms. |

| | | |
|---|---|---|
| **[KERNEL]** | `RtnRead` | Reads retain when start<br>The default value is 1. /* true */ |
| | `bkupType` | Resource backup location type for restoration<br>The default value is 1. /* Load from hard support (disk,...) */ |
| **[RSI]** | `RSIAddress` | Slave RSI<br>The default value is 1. |
| | `Port` | Slave Port.<br>The default value is 0 /* Null */ |
| | `CycleTime` | Cycle time. The value of this parameter is in milliseconds<br>The default value is 1ms. |
| | `ChNbr` | Maximum number of channels<br>The default value is 50. |
| | `NbIxlClt` | RSI Number of IXL Client<br>The default value is 3. |

**Example of initialization file (ISaGRAF.ini)**

```
[ETCP]

ChNbr=256

IsctNb=256

EsctNb=256

RBitNb=256

NCRBSize=16384


[APP]

ResNbr=100

RSI=COM1
```

```
[RSI]

PORT=COM1:19200:N:1:OFF

RSIAddress=1
```

**To install a Windows runtime module**

1. Copy the entire target directory.
2. On the target platform, paste the directory.

The Windows runtime module is ready.

### See Also

Setting Networks and Connections

# Setting Networks and Connections

The ISAFREE_TGT target supports three types of networks:

- ETCP
- HSD
- ISaRSI

# ETCP

The Enhanced TCP/IP protocol (ETCP) is the network driver used for communication with **ISaGRAF** on Ethernet. In this case, the ETCP driver emulates the behavior of a field-bus. You can consider ETCP as a virtual field-bus.

ETCP automatically starts with the target. However, you can choose to disable the ETCP when installing a runtime module.

## Startup Parameters

You can specify startup parameters as entries in the APP section of the driver initialization file (ETCP.ini) or in a command line:

| | |
|---|---|
| **ChNbr** | CRU server Number of channels. The default value is 72. |
| **IsctNb** | The number of remote producing resources. However, each producing resource must only be counted once if it produces to several local resources. The default value is 100, meaning that an ETCP task can manage up to 100 remote producers for all of its local consumers. |
| **EsctNb** | The number of local producing resources (producing to remote platforms only) plus the number of distant consuming resources:<br>- If both R1 and R2 on C1 consume data from R3 on C2, there will only be one connection between C1 and C2<br>- If R1 on C1 consumes data from R2 and R3 on C2, there will be two connections between C1 and C2 |
| **RBitNb** | Host data concerning each remote producer. It should be the same as **IsctNb** value because an ISCT entry is used jointly with a RBIT entry. The default value is 100. |
| **NCRBSize** | VRU Non-converted reception buffer size, the total amount of bound bytes the ETCP task has to buffer for all remote producers. Each producer has a 12-byte header. The default value is 512 meaning that the ETCP task can handle 500 bound bytes for each remote producer. |
| **NbIxlClt** | ETCP Number of IXL Client. The default value is 3. |

| | |
|---|---|
| **Cycle** | ETCP server cycle time, in milliseconds. The default value is 1. |
| **NbrSend** | VRU Number of emissions. The default value is 1. |

### Network Properties

The ETCP network driver has no network properties.

### Connection Properties

You specify connection properties by selecting the connection in the Deployment View and entering the required values in the Properties window. The ETCP network driver has one connection property:

| | |
|---|---|
| IP Address | The IP address or name of the computer. |

# HSD

The Host System Driver (HSD) is the IXL driver used for communication between local processes with **ISaGRAF**. HSD network connections are used when defining bindings between resources on the same device.

The **ISaGRAF** target automatically launches the HSD driver.

## Startup Parameters

You can specify startup parameters as entries in the HSD section of the ISaGRAF target initialization file (isagraf.ini) or in a command line:

| | |
|---|---|
| **MaxMsgConnect** | The maximum number of messages in the connection message queue. The default value is 3. |
| **RcvMaxMsg** | The maximum number of messages in the message queue. The default value is 3. |
| **SendMaxMsg** | The maximum number of messages in the message queue. The default value is 3. |
| **SndSz** | The size of the exchanged messages, in bytes. The default value is 544. |
| **RcvSz** | The size of the exchanged messages, in bytes. The default value is 544. |
| **TimeOut** | The time period before a timeout occurs, in milliseconds; the default value is 5000 ms. |

## Network Properties

The HSD driver has no network properties.

## Connection Properties

The HSD driver has no connection properties.

# ISaRSI

The network driver used when developing an IXL client using serial communication with **ISaGRAF**.

## Startup Parameters

You can specify startup parameters as entries in the driver initialization file (ISaRSI.ini), the ISaGRAF initialization file (ISaGRAF.ini), or in a driver command line:

| | |
|---|---|
| **CycleTime** | Polling of the ISaRSI task, in milliseconds. The default value is 1. |

## Network Properties

You specify network properties by selecting the network in the Deployment View and entering the required values in the Properties window.

| | |
|---|---|
| **Port** | The **ISaGRAF** communication port. The default value is COM1. |
| **Baud Rate** | The baud data transfer rate. The default value is 19200. |
| **Parity** | The type of parity used. Possible values are N for none, E for even, and O for odd; The default value is N. |
| **Stop Bit** | The number of stop bits used to indicate the end of a transmission. Possible values are 1 or 2; The default value is 1. |
| **HardwareFlowControl** | The control of the flow of data transmission between the network hardware. Possible values are True or False; The default value is False. |

## Connection Properties

The ISaRSI network driver has no connection properties.

**To start the ISaRSI network driver**

- Double-click the executable file (ISaRSI.exe) located:

  %PROGRAMFILES(X86)%\ISaGRAF\6.x\CAM ISaGRAF 5\5.3\ISaGRAF Free RunTime

# Configuring I/O Devices

When configuring I/O devices, you connect I/O variables to I/O channels. You connect these variables and channels in the I/O wiring view. The hierarchical structure displayed in the I/O wiring view appears the same with differences depending on the driver:

| | | |
|---|---|---|
| Simple device | | |
| Parameters | | (Only displayed if the I/O device has defined parameters) |
| | Parameter_*n* | |
| Wired Channel | | |
| | Direct | Alternatively: Reverse (for Boolean values) |
| | Gain | =1/1 (* for Numeric Values *) |
| | Offset | =0 (* for Numeric Values *) |
| | Conversion | =None |
| Complex Device | | |

The following drivers are available for use with the Windows-TGT_L target:

- Modbus/TCP Client Implementation
- Modbus/TCP Server Implementation

# Modbus/TCP Client Implementation

The Modbus/TCP client driver includes twenty-six devices each dedicated to a particular data type and using a particular Modbus message. A twenty-seventh device, called a status device is not associated with a data variable but to a data structure describing the state of communication.

A project has a maximum of 256 device instances. For each device, you need to specify properties.

The data type must match the data sent by the Modbus server since the driver simply fills the variable with the returned data. For example, the FLOAT type is used when the server sends 32-bit floating point data.

The Modbus/TCP protocol is Big-Endian, that is a number larger than one byte is sequenced from highest to lowest byte. However, in the case of bad server implementation, the Endian type of the data can be selected, big or little in the user parameters, except for BOOL type devices.

When more than one channel is linked to a device, the driver simply writes the data received sequentially from the Modbus server to the variable values.

The following table shows these devices, their name, data type and direction and their associated Modbus message. The maximum channels indicates the greatest number of variables that can be handled by the device (information extracted from the Schneider Electric specification for the message).

| **Device** | **Data Type** | **Direction** | **Modbus message** | **Max Chan.** |
|---|---|---|---|---|
| Client_RC | BOOL | input | Read Coils (fct 1) | 2000 |
| Client_RID | BOOL | input | Read Input Discretes (fct 2) | 2000 |
| Client_RMR_INT | INT | input | Read Multiple Registers (fct 3) | 125 |
| Client_RMR_UINT | UINT | input | Read Multiple Registers (fct 3) | 125 |
| Client_RMR_DINT | DINT | input | Read Multiple Registers (fct 3) | 62 |

| | | | | |
|---|---|---|---|---|
| Client_RMR_UDINT | UDINT | input | Read Multiple Registers (fct 3) | 62 |
| Client_RMR_REAL | REAL | input | Read Multiple Registers (fct 3) | 62 |
| Client_RIR_INT | INT | input | Read Input Registers (fct 4) | 125 |
| Client_RIR_UINT | UINT | input | Read Input Registers (fct 4) | 125 |
| Client_RIR_DINT | DINT | input | Read Input Registers (fct 4) | 62 |
| Client_RIR_UDINT | UDINT | input | Read Input Registers (fct 4) | 62 |
| Client_RIR_REAL | REAL | input | Read Input Registers (fct 4) | 62 |
| Client_WC | BOOL | output | Write Coil (fct 5) | 1 |
| Client_WSR_INT | INT | output | Write Single Register (fct 6) | 1 |
| Client_WSR_UINT | UINT | output | Write Single Register (fct 6) | 1 |
| Client_FMC | BOOL | output | Force Multiple Coils (fct 15) | 800 |
| Client_WMR_INT | INT | output | Write Multiple Registers (fct 16) | 100 |
| Client_WMR_UINT | UINT | output | Write Multiple Registers (fct 16) | 100 |
| Client_WMR_DINT | DINT | output | Write Multiple Registers (fct 16) | 50 |
| Client_WMR_UDINT | UDINT | output | Write Multiple Registers (fct 16) | 50 |
| Client_WMR_REAL | REAL | output | Write Multiple Registers (fct 16) | 50 |
| Client_RW_M_INT | INT | output | Fct 3 normally – Fct 16 if value change | 100 |

| | | | | |
|---|---|---|---|---|
| Client_RW_M_UINT | UINT | output | Fct 3 normally – Fct 16 if value change | 100 |
| Client_RW_M_DINT | DINT | output | Fct 3 normally – Fct 16 if value change | 50 |
| Client_RW_M_UDINT | UDINT | output | Fct 3 normally – Fct 16 if value change | 50 |
| Client_RW_M_REAL | REAL | output | Fct 3 normally – Fct 16 if value change | 50 |
| ClientStatus | CLIENT_STAT | input | n/a | 256 |

The Read/Write (RW) drivers are bidirectional meaning that these behave as an output driver when the data channel has changed, otherwise, these behave as an input driver. These driver properties are the same as an output device except that they do not have the SendOnChange property.

The ClientStatus device updates its values during each cycle of the **ISaGRAF** target. This device completes the CLIENT_STAT structure shown below:

```
struct CLIENT_STAT

{

  uchar Connected;// Connected to a server or not

  uint32 MessageTxCount;// Client requests sent

  uint32 MessageRxCount;// Server response message received

  uint32 ExceptionRxCount;// Server sent back an Exception response

  uint32 TxErrorCount;// Transmit failures

  uint32 RxErrorCount;// Read failures

  int16 LastError;// Socket last error code

}
```

A project can have one of these structures per device.

## Target Preparation

The ModbusTCP_Driver.DLL file must be copied in the directory where the executable file for the target is located.

You start the target by executing the **ISaGRAF** process, located in the Cmds sub-directory for the target.

The isagraf.ini initialization file can be used to specify some target parameters. Refer to the *Startup Parameters Configuration* section in the **ISaGRAF** Development Kit Guide documentation.

## Importation of Drivers in the Workbench

To enable access to the Modbus/TCP client driver, you need to import the definitions of the Modbus devices into **ISaGRAF**, defined in the following file:

Windows_ModbusTcpClient.txt

Enter your project functionality and variables. Follow the instructions as described in the manual to instantiate a driver. Then, connect the desired variables to the corresponding Modbus device. **ISaGRAF** only allows connecting a variable whose type matches the data type of the device.

**To import the Plc definition file for Modbus devices**

1. From the Solution Explorer, right-click the project, point to **Import**, and then click **Import Target Definitions**.
2. In the Open dialog box, browse to locate the *Windows_ModbusTcpClient.txt* Plc Definition file, then click **Open**.

## Properties of Modbus/TCP Client Devices

You can change the properties of the device in the I/O Device tool of the workbench. You access them by selecting the device in the browser located on the left side of the module window. The following properties apply to devices depending on their type:

| | |
|---|---|
| IPaddress | The IP address of the Modbus server (slave) to communicate with. The format of this property is String; its value ranges from 0.0.0.0 to 255.255.255.255; its default value is 127.0.0.1. |
| UnitIdentifier | Formerly the slave address, sent on each message in the prefix. The format of this property is Char; its value ranges from 0 to 255; its default value is 1. |
| StartAddress | The register offset of the data in the server. The format of this property is WordHexa; its value ranges from 0 to 65535; its default value is 0. |
| TimeOut | The time period in which to wait for a response, in milliseconds. The format of this property is Word; its value ranges from 1 to 65535; its default value is 2000. |
| SendOnChange | Applies only to output devices. The indication that the message is sent only when channel's data has changed. The format of this property is BOOL; possible values are TRUE or FALSE. FALSE indicates that the message is sent at each cycle. The default value is TRUE. |
| DataIsBigEndian | Applies only to input and output devices other than BOOL type as well as read/write devices. The indication that the data read from the server is interpreted as Big-endian. Possible values are TRUE or FALSE. FALSE indicates that the data read is interpreted as Little-endian. The default value is TRUE. |
| UseTCP | The indication that the communication method for the transport layer is TCP. The format of this property is BOOL; possible values are TRUE or FALSE. FALSE indicates that the communication method for the transport layer is UDP (not currently implemented). The default value is TRUE. |
| PortNumber | The transport layer port number to use. The format of this property is WordHexa; its value ranges from 0 to 65535; its default value is 502. |
| RequestPeriod | The time interval between sent Modbus requests, in milliseconds. The format of this property is Word; its value ranges from 1 to 65535; its default value is 1000. |

The ClientStatus device has no properties. A project has only one instance of this driver and you need to hook as many CLIENT_STAT type channels as the largest ModbusTCP device index in your project. For example, when a project has five ModbusTCP devices along with other driver devices where the largest ModbusTCP device index is twenty-five, you need to hook twenty-five CLIENT_STAT channels to a ClientStatus device in order to monitor all five ModbusTCP devices.

## Modbus/TCP Prefixes

Each Modbus/TCP message contains a seven-byte prefix:

Byte 0: transaction identifier - copied by server - usually 0

Byte 1: transaction identifier - copied by server - usually 0

Byte 2: protocol identifier = 0

Byte 3: protocol identifier = 0

Byte 4: length field (upper byte) = 0 (since all messages are smaller than 256)

Byte 5: length field (lower byte) = number of bytes following

Byte 6: unit identifier (previously ‘slave address’)

The following example shows the ‘read 1 register at offset 4 from UI 9’ transaction returning a value of 5:

request: 00 00 00 00 00 06 09 03 00 04 00 01

response: 00 00 00 00 00 05 09 03 02 00 05

The MODBUS ‘slave address’ field is replaced by a single byte ‘Unit Identifier’ which may be used to communicate via devices such as bridges and gateways which use a single IP address to support multiple independent end units.

The transaction identifier will be a word variable that increments at each message sent. The server shall respond with it in it’s response prefix.

## Modbus/TCP Message Descriptions

### Read coils (FC 1)

**Request**

Byte 0: FC = 01

Byte 1-2: Reference number

Byte 3-4: Bit count (1-2000)

**Response**

Byte 0: FC = 01

Byte 1: Byte count of response (B=(bit count+7)/8)

Byte 2-(B+1): Bit values (least significant bit is first coil!)

**Exceptions**

Byte 0: FC = 81 (hex)

Byte 1: exception code = 01 or 02

**Example**

Read 1 coil at reference 0 (00001 in Modicon 984) resulting in value 1

01 00 00 00 01 => 01 01 01

The format of the return data is not consistent with a Big-Endian architecture. Also, this request can be quite computation-intensive on the slave if the request calls for multiple words where these are not aligned on 16-bit boundaries.

### Read input discretes (FC 2)

**Request**

Byte 0: FC = 02

Byte 1-2: Reference number

Byte 3-4: Bit count (1-2000)

**Response**

Byte 0: FC = 02

Byte 1: Byte count of response (B=(bit count+7)/8)

Byte 2-(B+1): Bit values (least significant bit is first coil!)

**Exceptions**

Byte 0: FC = 82 (hex)

Byte 1: exception code = 01 or 02

**Example**

Read 1 discrete input at reference 0 (10001 in Modicon 984) resulting in value 1

02 00 00 00 01 => 02 01 01

The format of the return data is not consistent with a Big-Endian architecture. Also, this request can be quite computation-intensive on the slave if the request calls for multiple words where these are not aligned on 16-bit boundaries.

### Read multiple registers (FC 3)

**Request**

Byte 0: FC = 03

Byte 1-2: Reference number

Byte 3-4: Word count (1-125)

**Response**

Byte 0: FC = 03

Byte 1: Byte count of response (B=2 x word count)

Byte 2-(B+1): Register values

**Exceptions**

Byte 0: FC = 83 (hex)

Byte 1: exception code = 01 or 02

**Example**

Read 1 register at reference 0 (40001 in Modicon 984) resulting in value 1234 hex

03 00 00 00 01 => 03 02 12 34

### Read input registers (FC 4)

**Request**

Byte 0: FC = 04

Byte 1-2: Reference number

Byte 3-4: Word count (1-125)

**Response**

Byte 0: FC = 04

Byte 1: Byte count of response (B=2 x word count)

Byte 2-(B+1): Register values

**Exceptions**

Byte 0: FC = 84 (hex)

Byte 1: exception code = 01 or 02

**Example**

Read 1 input register at reference 0 (30001 in Modicon 984) resulting in value 1234 hex

04 00 00 00 01 => 04 02 12 34

### Write coil (FC 5)

**Request**

Byte 0: FC = 05

Byte 1-2: Reference number

Byte 3: = FF to turn coil ON, =00 to turn coil OFF

Byte 4: = 00

**Response**

Byte 0: FC = 05

Byte 1-2: Reference number

Byte 3: = FF to turn coil ON, =00 to turn coil OFF (echoed)

Byte 4: = 00

**Exceptions**

Byte 0: FC = 85 (hex)

Byte 1: exception code = 01 or 02

**Example**

Write 1 coil at reference 0 (00001 in Modicon 984) to the value 1

05 00 00 FF 00 => 05 00 00 FF 00

### Write single register (FC 6)

**Request**

Byte 0: FC = 06

Byte 1-2: Reference number

Byte 3-4: Register value

**Response**

Byte 0: FC = 06

Byte 1-2: Reference number

Byte 3-4: Register value

**Exceptions**

Byte 0: FC = 86 (hex)

Byte 1: exception code = 01 or 02

**Example**

Write 1 register at reference 0 (40001 in Modicon 984) of value 1234 hex

06 00 00 12 34 => 06 00 00 12 34

### Force multiple coils (FC 15)

**Request**

Byte 0: FC = 0F (hex)

Byte 1-2: Reference number

Byte 3-4: Bit count (1-800)

Byte 5: Byte count (B = (bit count + 7)/8)

Byte 6-(B+5): Data to be written (least significant bit = first coil)

**Response**

Byte 0: FC = 0F (hex)

Byte 1-2: Reference number

Byte 3-4: Bit count

**Exceptions**

Byte 0: FC = 8F (hex)

Byte 1: exception code = 01 or 02

**Example**

Write 3 coils at reference 0 (00001 in Modicon 984) to values 0,0,1

0F 00 00 00 03 01 04 => 0F 00 00 00 03

The format of the input data is not consistent with a Big-Endian architecture. Also, that this request can be quite computation-intensive on the slave if the request calls for multiple words where these are not aligned on 16-bit boundaries.

### Write multiple registers (FC 16)

**Request**

Byte 0: FC = 10 (hex)

Byte 1-2: Reference number

Byte 3-4: Word count (1-100)

Byte 5: Byte count (B=2 x word count)

Byte 6-(B+5): Register values

**Response**

Byte 0: FC = 10 (hex)

Byte 1-2: Reference number

Byte 3-4: Word count

**Exceptions**

Byte 0: FC = 90 (hex)

Byte 1: exception code = 01 or 02

**Example**

Write 1 register at reference 0 (40001 in Modicon 984) of value 1234 hex

10 00 00 00 01 02 12 34 => 10 00 00 00 01

# Modbus/TCP Server Implementation

The Modbus/TCP server has twenty-one devices each dedicated to a particular data type and using a particular Modbus message. A twenty-second device, called a status device is not associated with a data variable but to a data structure describing the state of the communication for each device in the project.

A project has a maximum of 256 device instances. For each device, you need to specify properties. A maximum of 256 responded Modbus requests per cycle are possible.

On a write, the data type must match the data sent by the Modbus client since the driver simply fills the variable with the received data. For example, the FLOAT type is used when the server sends 32-bit floating point data.

The Modbus/TCP protocol is Big-Endian, that is a number larger than one byte is sequenced from highest to lowest byte. However, in the case of bad server implementation, the Endian type of the data can be selected, big or little in the user parameters, except for BOOL type devices.

When more than one channel is linked to a device, the driver simply writes the data received sequentially from the Modbus server to the variable values.

The following table shows these devices, their name, data type and direction and their associated Modbus message. The maximum channels indicates the greatest number of variables that can be handled by the device (information extracted from the Schneider Electric specification for the message).

| **Device** | **Data Type** | **Direction** | **Modbus message** | **Max Chan.** |
|---|---|---|---|---|
| Server_RC | BOOL | output | Read Coils (fct 1) | 2000 |
| Server_RID | BOOL | output | Read Input Discretes (fct 2) | 2000 |
| Server_R_INT | INT | output | Respond to RMR (Fct 3) or RIR (Fct 4) | 125 |
| Server_R_UINT | UINT | output | Respond to RMR (Fct 3) or RIR (Fct 4) | 125 |
| Server_R_DINT | DINT | output | Respond to RMR (Fct 3) or RIR (Fct 4) | 62 |

| | | | | |
|---|---|---|---|---|
| Server_R_UDINT | UDINT | output | Respond to RMR (Fct 3) or RIR (Fct 4) | 62 |
| Server_R_REAL | REAL | output | Respond to RMR (Fct 3) or RIR (Fct 4) | 62 |
| Server_WC | BOOL | input | Write Coil (fct 5) | 1 |
| Server_WSR_INT | INT | input | Write Single Register (fct 6) | 1 |
| Server_WSR_UINT | UINT | input | Write Single Register (fct 6) | 1 |
| Server_FMC | BOOL | input | Force Multiple Coils (fct 15) | 800 |
| Server_WMR_INT | INT | input | Write Multiple Registers (fct 16) | 100 |
| Server_WMR_UINT | UINT | input | Write Multiple Registers (fct 16) | 100 |
| Server_WMR_DINT | DINT | input | Write Multiple Registers (fct 16) | 50 |
| Server_WMR_UDINT | UDINT | input | Write Multiple Registers (fct 16) | 50 |
| Server_WMR_REAL | REAL | input | Write Multiple Registers (fct 16) | 50 |
| Server_RW_INT | INT | output | Respond to Fct 3 or Fct 4 or Fct 16 | 100 |
| Server_RW_UINT | UINT | output | Respond to Fct 3 or Fct 4 or Fct 16 | 100 |
| Server_RW_DINT | DINT | output | Respond to Fct 3 or Fct 4 or Fct 16 | 50 |
| Server_RW_UDINT | UDINT | output | Respond to Fct 3 or Fct 4 or Fct 16 | 50 |
| Server_RW_REAL | REAL | output | Respond to Fct 3 or Fct 4 or Fct 16 | 50 |
| ServerStatus | SERVER_STAT | input | n/a | 256 |

The Read/Write (RW) drivers are bidirectional meaning that these are assigned to output free variables and can respond to a read request (message function 3 or 4) or to a write request (message function 16). Their parameters are the same as an output device.

The ServerStatus device has no parameters and update it's values at each **ISaGRAF** target's cycle.

Because many different devices can respond to the same Modbus message, use different addresses to differentiate them. For example, you can have a Server_R_INT device responding to address 100 and a Server_RUINT device responding to address 500. Failing to do this produces erroneous results, a Server_WMR_REAL device could respond to a Client_WMR_DINT device.

If a request has an invalid range, for example, asking more registers than there are channels hooked on a device or asking for an address outside the range of the server, a Modbus exception is returned.

If two devices have an address overlap, only the first device instance responds to the requests. For example, a Server_R_INT covering the address range 20000-20100 and a Server_R_INT covering the address range 20050-20200, the requests in the overlap range 20050-20100 are responded to by Server_R-_INT if the device index is less (instanced before) than Server_R-_INT.

Two devices of different data types can have an address overlap. In the above example, if the first device is a Server_R_INT and the second is a Server_R_UINT, both devices respond.

More than one client can send requests to a single server while there can be a maximum of 256 connected clients.

The ServerStatus device updates the values of the SERVER_STAT structure. One of these structures exists for each device currently instantiated in your project.

```
struct SERVER_STAT

{

  char  ClientCount;// Current number of client(s) connected

  uint32 MessageRxCount;// Client requests received

  uint32 ExceptionTxCount;// Exception message sent
```

```
uint32 TxErrorCount;// Transmit failures

uint32 RxErrorCount;// Read failures

int16  LastError;// Socket last error code
```

## Target Preparation

The ModbusTcpServer.DLL file must be copied in the directory where the executable file for the target is located.

You start the target by executing the **ISaGRAF** process, located in the Cmds sub-directory for the target.

The isagraf.ini initialization file can be used to specify some target parameters. Refer to the *Startup Parameters Configuration* section in the **ISaGRAF** Development Kit Guide documentation.

## Importation of Drivers in the Workbench

To enable access to the Modbus/TCP server driver, you need to import the definitions of the Modbus devices into **ISaGRAF**, defined in the following file:

Windows_ModbusTcpServer.txt

Enter your project functionality and variables. Follow the instructions as described in the manual to instantiate a driver. Then, connect the desired variables to the corresponding Modbus device. The workbench only allows connecting a variable whose type matches the data type of the device.

**To import the Plc definition file for Modbus devices**

1. From the Solution Explorer, right-click the project, point to **Import**, and then click **Import Target Definitions**.

2. In the Open dialog box, browse to locate the *Windows_ModbusTcpServer.txt* Plc Definition file, then click **Open**.

## Properties of Modbus/TCP Server Devices

When performing I/O Wiring, you can define the device properties. You access device properties by selecting a device from the list of available devices. You modify the properties for the selected device using the Properties window. The following properties apply to devices depending on their type:

| | |
|---|---|
| StartAddress | The register offset of the data in the server. The format of this property is WordHexa; its value ranges from 0 to 65535; its default value is 0. |
| TimeOut | The time period in which to wait for a response, in milliseconds. The format of this property is Word; its value ranges from 1 to 65535; its default value is 2000. |
| DataIsBigEndian | Applies only to devices other than BOOL type. The indication that the data read from the server is interpreted as Big-endian. Possible values are TRUE or FALSE. FALSE indicates that the data read is interpreted as Little-endian. The default value is TRUE. |
| UseTCP | The indication that the communication method for the transport layer is TCP. The format of this property is BOOL; possible values are TRUE or FALSE. FALSE indicates that the communication method for the transport layer is UDP (not currently implemented). The default value is TRUE. |
| PortNumber | The transport layer port number to use. The format of this property is WordHexa; its value ranges from 0 to 65535; its default value is 502. |

The ServerStatus device has no properties. A project has only one instance of this driver and you need to hook as many SERVER_STAT type channels as the largest ModbusTCP device index in your project. For example, when a project has five ModbusTCP devices along with other driver devices where the largest ModbusTCP device index is twenty-five, you need to hook twenty-five SERVER_STAT channels to a ServerStatus device in order to monitor all five ModbusTCP devices. Those twenty-five SERVER_STAT channels can be made of discrete variables or an array.

## Modbus/TCP Prefixes

Each Modbus/TCP message contains a seven-byte prefix:

Byte 0: transaction identifier - copied by server - usually 0

Byte 1: transaction identifier - copied by server - usually 0

Byte 2: protocol identifier = 0

Byte 3: protocol identifier = 0

Byte 4: length field (upper byte) = 0 (since all messages are smaller than 256)

Byte 5: length field (lower byte) = number of bytes following

Byte 6: unit identifier (previously ‘slave address’)

An example transaction ‘read 1 register at offset 4 from UI 9’ returning a value of 5 is

request: 00 00 00 00 00 06 09 03 00 04 00 01

response: 00 00 00 00 00 05 09 03 02 00 05

The MODBUS ‘slave address’ field is replaced by a single byte ‘Unit Identifier’ which may be used to communicate via devices such as bridges and gateways which use a single IP address to support multiple independent end units.

The transaction identifier will be a word variable that increments at each message sent. The server shall respond with it in it’s response prefix.

# Modbus/TCP Message Descriptions

## Read coils (FC 1)

**Request**

Byte 0: FC = 01

Byte 1-2: Reference number

Byte 3-4: Bit count (1-2000)

**Response**

Byte 0: FC = 01

Byte 1: Byte count of response (B=(bit count+7)/8)

Byte 2-(B+1): Bit values (least significant bit is first coil!)

**Exceptions**

Byte 0: FC = 81 (hex)

Byte 1: exception code = 01 or 02

**Example**

Read 1 coil at reference 0 (00001 in Modicon 984) resulting in value 1

01 00 00 00 01 => 01 01 01

The format of the return data is not consistent with a Big-Endian architecture. Also, this request can be very computation-intensive on the slave if the request calls for multiple words and these are not aligned on 16-bit boundaries.

## Read input discretes (FC 2)

**Request**

Byte 0: FC = 02

Byte 1-2: Reference number

Byte 3-4: Bit count (1-2000)

**Response**

Byte 0: FC = 02

Byte 1: Byte count of response (B=(bit count+7)/8)

Byte 2-(B+1): Bit values (least significant bit is first coil!)

**Exceptions**

Byte 0: FC = 82 (hex)

Byte 1: exception code = 01 or 02

**Example**

Read 1 discrete input at reference 0 (10001 in Modicon 984) resulting in value 1

02 00 00 00 01 => 02 01 01

The format of the return data is not consistent with a big-endian architecture. Also, this request can be very computation-intensive on the slave if the request calls for multiple words and these are not aligned on 16-bit boundaries.

### Read multiple registers (FC 3)

**Request**

Byte 0: FC = 03

Byte 1-2: Reference number

Byte 3-4: Word count (1-125)

**Response**

Byte 0: FC = 03

Byte 1: Byte count of response (B=2 x word count)

Byte 2-(B+1): Register values

**Exceptions**

Byte 0: FC = 83 (hex)

Byte 1: exception code = 01 or 02

**Example**

Read 1 register at reference 0 (40001 in Modicon 984) resulting in value 1234 hex

03 00 00 00 01 => 03 02 12 34

## Read input registers (FC 4)

**Request**

Byte 0: FC = 04

Byte 1-2: Reference number

Byte 3-4: Word count (1-125)

**Response**

Byte 0: FC = 04

Byte 1: Byte count of response (B=2 x word count)

Byte 2-(B+1): Register values

**Exceptions**

Byte 0: FC = 84 (hex)

Byte 1: exception code = 01 or 02

**Example**

Read one input register at reference 0 (30001 in Modicon 984) resulting in value 1234 hex

04 00 00 00 01 => 04 02 12 34

### Write coil (FC 5)

**Request**

Byte 0: FC = 05

Byte 1-2: Reference number

Byte 3: = FF to turn coil ON, =00 to turn coil OFF

Byte 4: = 00

**Response**

Byte 0: FC = 05

Byte 1-2: Reference number

Byte 3: = FF to turn coil ON, =00 to turn coil OFF (echoed)

Byte 4: = 00

**Exceptions**

Byte 0: FC = 85 (hex)

Byte 1: exception code = 01 or 02

**Example**

Write one coil at reference 0 (00001 in Modicon 984) to the value 1

05 00 00 FF 00 => 05 00 00 FF 00

### Write single register (FC 6)

**Request**

Byte 0: FC = 06

Byte 1-2: Reference number

Byte 3-4: Register value

**Response**

Byte 0: FC = 06

Byte 1-2: Reference number

Byte 3-4: Register value

**Exceptions**

Byte 0: FC = 86 (hex)

Byte 1: exception code = 01 or 0

**Example**

Write one register at reference 0 (40001 in Modicon 984) of value 1234 hex

06 00 00 12 34 => 06 00 00 12 34

### Force multiple coils (FC 15)

**Request**

Byte 0: FC = 0F (hex)

Byte 1-2: Reference number

Byte 3-4: Bit count (1-800)

Byte 5: Byte count (B = (bit count + 7)/8)

Byte 6-(B+5):Data to be written (least significant bit = first coil)

**Response**

Byte 0: FC = 0F (hex)

Byte 1-2: Reference number

Byte 3-4: Bit count

**Exceptions**

Byte 0: FC = 8F (hex)

Byte 1: exception code = 01 or 02

**Example**

Write 3 coils at reference 0 (00001 in Modicon 984) to values 0,0,1

0F 00 00 00 03 01 04 => 0F 00 00 00 03

The format of the input data is not consistent with a Big-Endian architecture. Also, this request can be very computation-intensive on the slave if the request calls for multiple words and these are not aligned on 16-bit boundaries.

### Write multiple registers (FC 16)

**Request**

Byte 0: FC = 10 (hex)

Byte 1-2: Reference number

Byte 3-4: Word count (1-100)

Byte 5: Byte count (B=2 x word count)

Byte 6-(B+5): Register values

**Response**

Byte 0: FC = 10 (hex)

Byte 1-2: Reference number

Byte 3-4: Word count

**Exceptions**

Byte 0: FC = 90 (hex)

Byte 1: exception code = 01 or 02

**Example**

Write one register at reference 0 (40001 in Modicon 984) of value 1234 hex

10 00 00 00 01 02 12 34 => 10 00 00 00 01

## Modbus/TCP Exception Codes

Slaves return a defined set of exception codes in the event of problems. Masters may send out commands ‘speculatively’ and use the success or exception codes received to determine which MODBUS commands the device is willing to respond to and to determine the size of the various data regions available on the slave.

This is the description of the exceptions that **ISaGRAF**’s Modbus/TCP server driver can send. For a description of the other exceptions, please refer to the Schneider Electric V1.0 specification document.

All exceptions are signaled by adding 0x80 to the function code of the request and following this byte by a single reason byte for example as follows:

```
03  12  34  00  01  =>  83  02
```

request read one register at index 0x1234 response exception type 2 - ‘illegal data address’

The list of exceptions follows:

01 ILLEGAL FUNCTlON

The function code received in the query is not an allowable action for the slave. This may be because the function code is only applicable to newer controllers, and was not implemented in the unit selected. It could also indicate that the slave is in the wrong state to process a request of this type, for example because it is not configured and is being asked to return register values.

02 ILLEGAL DATA ADDRESS

The data address received in the query is not an allowable address for the slave. More specifically, the combination of reference number and transfer length is invalid. For a controller having 100 registers, a request with offset 96 and length 4 would succeed, a request with offset 96 and length 5 generates exception 02.

03 ILLEGAL DATA VALUE

A value contained in the query data field is not an allowable value for the slave. This indicates a fault in the structure of the remainder of a complex request, such as that the implied length is incorrect. It specifically does NOT mean that a data item submitted for storage in a register has a value outside the expectation of the application program, since the MODBUS protocol is unaware of the significance of any particular value of any particular register.

# Defining Ports Usage

When using **ISaGRAF**, you may need to define ports for various usage.

**Ports usage specific to ISaGRAF**

ETCP: 1131(CRU - Channel Replacement Unit) and 1113 (VRU - Variable Replacement Unit)

HabDts: 5005 and 6001

**General ports usage**

RPC (Remote Procedure Call) : 111

Telnet: 23, 24

FTP (File Transfer Protocol): 20, 21

# Error Messages

You can search for error messages relating to the following modules:

- Events Logger
- ISaGRAF Target

For the **ISaGRAF** target, you can also refine your search to the sub-module level:

| | |
|---|---|
| Configuration Manager | Kernel |
| Kernel Warning | System Layer |
| I/Os | Host System Driver Binding |
| eXchange Dispatcher (IXD) | eXchange Layer (IXL) |
| ETCP Task | ETCP Binding |
| ISaRSI Task | Common Errors |
| ISaGRAF 3 Communication | |

**Events Logger**

| Code | Description | Probable Cause | Diagnostic | #define |
|---|---|---|---|---|
| 0x00100001 | Cannot read network parameters of the device | Project not compiled | Build project | ISA_ER_EL_NOCONFIG |
| 0x00100002 | Not implemented on the target | Old target | Use ISaGRAF 4.20 Target | ISA_ER_EL_NOTIMPLEMENTED |
| 0x00100003 | Disk full, logging interrupted | Not enough space on disk | Clear space on disk | ISA_ER_EL_DISKFULL |

| Code | Description | Probable Cause | Diagnostic | #define |
|---|---|---|---|---|
| 0x00100004 | Cannot connect to the target | Cannot connect to the target | Verify target and network parameters | ISA_ER_EL_CONNECTION |
| 0x00100005 | Cannot communicate with the target | Cannot communicate with the target | Verify target | ISA_ER_EL_COMMUNICATION |
| 0x00100006 | Cannot retrieve events from the target | Error while retrieving events | Verify target and communication | ISA_ER_EL_GETEVENTS |
| 0x00100007 | Device has an invalid password | Password for the device is invalid | Verify password | ISA_ER_EL_PASSWDINVALID |

**ISaGRAF Target**

| Configuration Manager | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000300 | CMG: Cannot start new kernel | The configuration manager is requesting to start more resources than supported | Increase the value of the "Multi Resources Max Quantity" property (found in TDBuild) and regenerate the dsys0tgt.h file | ISA_ER_CMG_KER_START |
| 0x20000301 | CMG: Kernel is already running | A request has been made to start an already existing resource or there is a problem in the task management | Review the implementation | ISA_ER_CMG_KER_ALREADYRUNNING |

| **Configuration Manager** | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000302 | CMG: Kernel is not running | A request has been made to access a task that does not exist or there is problem in the task management | Review the implementation | ISA_ER_CMG_KER_NOTRUNNING |
| 0x20000310 | CMG: Cannot start task | The configuration manager is requesting to start more miscellaneous tasks than supported | Increase the value of ISA_CMG_MISNBR | ISA_ER_CMG_MIS_START |

| **Kernel** | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x1201000AUL | Driver for the consumer is in error state | Error during data consumption in the binding mechanism | Verify if the producer is disconnected or review the driver implementation | ISA_RC_DKER_KVB_CONSUME |
| 0x22010001UL | Private resources not found or not initialized | Memory Data block corrupted | Compare with workbench files | ISA_RC_DKER_INIT_PRIV_BLOCK |
| 0x22010002UL | Kernel data allocation failed | Kernel Resource Data Loading error | Verify the system layer implementation | ISA_RC_DKER_INIT_ALLOC |

| Kernel | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x22010003UL | Standard 'C function not initialized | Changes made outside the system layer | Compare with the original PRDK | ISA_RC_DKER_INIT_STD_C |
| 0x22010004UL | User 'C' function not initialized | Incorrect implementation | Review the implementation | ISA_RC_DKER_INIT_USR_C |
| 0x22010005UL | Standard function block not initialized | Changes made outside the system layer | Compare with the original PRDK | ISA_RC_DKER_INIT_STD_FBL |
| 0x22010006UL | User function block not initialized | Incorrect implementation | Review the implementation | ISA_RC_DKER_INIT_USR_FBL |
| 0x22010007UL | 'C' conversions functions not initialized | Incorrect implementation | Review the implementation | ISA_RC_DKER_INIT_CONV_C |
| 0x22010008UL | IOs not initialized | Incorrect implementation | Review the implementation | ISA_RC_DKER_INIT_IOS |
| 0x22010009UL | Driver for the bindings failed to initialize | Creation of the binding memory space or initialization of the binding mechanism failed | Review the implementation | ISA_RC_DKER_INIT_KVB |
| 0x2201000BUL | Initialize Step by step debugging management | File corrupted | Compare with workbench files | ISA_RC_DKER_INIT_DBG |

| Kernel | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000100 | RDCC: Target name mismatch. | Incorrect target type selected in the workbench | Compare workbench target type and real target type | ISA_ER_RDCC_BADTGTNAME |
| 0x20000101 | RDCC: Version of generated code & conf mismatch | Version mismatch between the application downloaded on the run-time and the run-time itself | Determine if the proper workbench has been used to develop the application (an old workbench cannot download an application on a newer run-time) | ISA_ER_RDCC_BADRDCCVERS |
| 0x20000102 | RDCC: Data base CRC mismatch. | File corrupted | Compare with workbench files | ISA_ER_RDCC_BADRDBCRC |
| 0x20000103 | RDCC: Module name mismatch. | File corrupted | Compare with workbench files | ISA_ER_RDCC_BADMODNAME |
| 0x20000104 | RDCC: Resource name mismatch. | File corrupted | Compare with workbench files | ISA_ER_RDCC_BADRESNAME |
| 0x20000105 | RDCC: Corrupted module. | File corrupted | Compare with workbench files | ISA_ER_RDCC_CORRUPTMODULE |
| 0x20000110 | K_LDG: Target segmentation mismatch. | Application too large | Reduce the application resource | ISA_ER_LDG_TGTNONSGMTD |

| Kernel | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000111 | K_LDG: Too many blocks of memory to allocate. | File corrupted | Compare with workbench files | ISA_ER_LDG_TOOMANYBLK |
| 0x20000112 | K_LDG: System variables overlap | File corrupted | Compare with workbench files | ISA_ER_LDG_SYSVAOVERLAP |
| 0x20000120 | KVB: Memory allocated is too short | File corrupted | Compare with workbench files | ISA_ER_KVB_MEMTOOSHORT |
| 0x20000121 | KVB: Cannot load driver | Communication driver error | Review the implementation | ISA_ER_KVB_DRIVERLOAD |
| 0x20000122 | KVB: Driver is not loaded | Not in use | Not applicable | ISA_ER_KVB_DRVNOTLOADED |
| 0x20000123 | KVB: Invalid driver | Communication driver error | Review the implementation | ISA_ER_KVB_DRVINVALID |
| 0x20000130 | K_MDF: Online modification not initialized | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_MDF_NOTINIT |
| 0x20000131 | K_MDF: The “C” function that allocates new data space memory has received from it’s input parameter a size equal to zero. | File corrupted | Compare with workbench files | ISA_ER_MDF_ZERODATA |

| Kernel | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000132 | K_MDF: Not enough memory for online modifications | Space reserved for on-line modification is not enough | Increase the Space reserved for on-line modification in the workbench application | ISA_ER_MDF_MEMTOOSHORT |
| 0x20000133 | K_MDF: No new modifications to update | File corrupted | Compare with workbench files | ISA_ER_MDF_NOMODIF |
| 0x20000134 | K_MDF: Cannot update POU (new objects within it) | File corrupted | Compare with workbench files | ISA_ER_MDF_CHKPOUOBJ |
| 0x20000135 | K_MDF: Cannot save modifications, code is not saved | File corrupted | Compare with workbench files | ISA_ER_MDF_SAVENOCODE |
| 0x20000136 | SFCFBL: Changes are not allowed | Required function not implemented (example: accepting IO on-line change) | Review the implementation | ISA_ER_MDF_NOTALLOWED |
| 0x20000140 | SFCFBL: Error when initializing SFC function block, space is present | Incorrect implementation | Review the implementation | ISA_ER_SFCFBL_SPC_PRESENT |
| 0x20000141 | SFCFBL: Error when initializing SFC function block, space allocation failed | Unable to create memory space | Verify the system layer implementation | ISA_ER_SFCFBL_SPC_ALLOC |

| **Kernel** | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000142 | SFCFBL: Table is corrupted | File corrupted | Compare with workbench files | ISA_ER_SFCFBL_TBL_CORRUPTED |
| 0x20000180 | KER: Slave number not allowed | Incorrect resource number | Change the resource number in the workbench application | ISA_ER_KER_BADSLAVENUM |
| 0x20000181 | KER: Kernel is not in appropriate state | No conf module available | Compare with workbench files | ISA_ER_KER_BADSTATE |
| 0x20000182 | KER: Bad parameters in request | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_KMP_BADPARAM |
| 0x20000200 | SRV: Cannot allocate memory for server. | Unable to create memory space | Verify the system layer implementation | ISA_ER_SRV_MEMORY |
| 0x20000201 | SRV: Cannot create message queue for connection to server | Unable to create message queue | Verify the system layer implementation | ISA_ER_SRV_MSGQ |
| 0x20000202 | SRV: Size of server buffer is smaller than connection message | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_SRV_LENGTHBUFFER |
| 0x20000203 | SRV: Cannot remove connection from server | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_SRV_DELCNX |
| 0x20000204 | SRV: No more connections available | CnxNbr value not enough | Change CnxNbr parameter value | ISA_ER_SRV_FULLCONNECT |

| Kernel | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000205 | SRV: Cannot link with client's message queue | Unable to open message queue | Verify the system layer implementation | ISA_ER_SRV_LINKMSGQ |
| 0x20000206 | SRV: Invalid connection identifier, attempted to read a message from an invalid connection | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_SRV_BADCNX |
| 0x20000207 | SRV: The question that the server read is larger than its buffer. The question is discarded | Message queue corrupted | Verify the system layer implementation | ISA_ER_SRV_MSGDISCARDED |
| 0x20000208 | SRV: Time out in received message | Not in use | Not applicable | ISA_ER_SRV_RCVTIMEOUT |
| 0x20000209 | SRV: Server replied with a bad TRC | Not in use | Not applicable | ISA_ER_SRV_TRCERROR |

| Kernel Warning | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x0001 | Startup error | Initialize Kernel core System error | Verify the system layer implementation | ISA_KWNG_STARTUP |
| 0x0002 | Server communication exchange: Accept error | Not in use | Not applicable | ISA_KWNG_SRVACC |

| Kernel Warning | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x0003 | Resource Restore error | Unable to restore saved resource | Verify the system layer implementation | ISA_KWNG_RESTORE |
| 0x0004 | Kernel Retain: Init error. | Unable to initialize retain | Verify the system layer implementation | ISA_KWNG_RTNINIT |
| 0x0005 | Kernel Retain: Bad memory description | Memory description too long | Change memory description in the workbench application | ISA_KWNG_RTNMEM |
| 0x0006 | Kernel Retain: CRC error | File corrupted | Verify the system layer implementation | ISA_KWNG_RTNCRC |
| 0x0007 | Kernel Retain: Read error | Unable to read | Verify the system layer implementation | ISA_KWNG_RTNREAD |
| 0x0008 | Kernel Retain: Write error | Unable to write | Verify the system layer implementation | ISA_KWNG_RTNWRITE |
| 0x0009 | Resource Data Allocation error | File corrupted | Compare with workbench files | ISA_KWNG_DATAALLOC |
| 0x000A | Resource Start Report | Resource started workbench | Verify the workbench state | ISA_KWNG_RSTART |
| 0x000B | Resource Stop Report | Resource stopped by the workbench | Verify the workbench state | ISA_KWNG_RSTOP |
| 0x000C | Standard function not implemented | Changes made outside the system layer | Compare with the original PRDK | ISA_KWNG_USFSTDCALL_NOTIMPLEM |

| Kernel Warning | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x000D | Standard function block instance init not implemented | Changes made outside the system layer | Compare with the original PRDK | ISA_KWNG_FBLSTDIINIT_NOTIMPLEM |
| 0x000E | Standard function block instance exit not implemented | Changes made outside the system layer | Compare with the original PRDK | ISA_KWNG_FBLSTDIEXIT_NOTIMPLEM |
| 0x000F | Standard function block call not implemented | Changes made outside the system layer | Compare with the original PRDK | ISA_KWNG_FBLSTDCALL_NOTIMPLEM |
| 0x0010 | Function not implemented | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_USF_FCTNOTFOUND |
| 0x0011 | Function block instance init required but not implemented | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_FBLIINIT_NOTIMPLEM |
| 0x0012 | Function block instance exit required but not implemented | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_FBLIEXIT_NOTIMPLEM |
| 0x0013 | Function block call not implemented | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_FBLCALL_NOTIMPLEM |
| 0x0014 | Function not implemented | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_CNV_FCTNOTFOUND |

| Kernel Warning | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x0015 | Initialize IO management error | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_IOSINIT |
| 0x0016 | Kernel IOs: Device Open/Close fct(s) not found | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_IOSDRV_FCTNOTFOUND |
| 0x0017 | Kernel IOs: Device Open/Close fct(s) not found | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_IOSDVC_FCTNOTFOUND |
| 0x0018 | Kernel IOs: Device read Function not found | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_IOSDVCR_FCTNOTFOUND |
| 0x0019 | Device write Function no found | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_IOSDVCW_FCTNOTFOUND |
| 0x001A | Kernel IOs: Device Control fct not found | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_IOSDVCCTL_FCTNOTFOUND |
| 0x001B | Kernel IOs: Driver Init fct failure | Incorrect implementation | Review the implementation | ISA_KWNG_IOSDRV_INITFAIL |
| 0x001C | Kernel IOs: Device Open function failure | Incorrect implementation | Review the implementation | ISA_KWNG_IOSDVC_OPENFAIL |

| Kernel Warning | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x001D | Kernel Binding: Driver not found | A binding driver configured in an application cannot be found on the run-time | Review the implementation. Due to a mistmach between the TDB file used by the workbench and the definition of the run-time. | ISA_KWNG_KVBDRVNOTFOUND |
| 0x001E | Kernel Binding: Init error | Creation of the binding memory space or initialization of the binding mechanism failed | Review the implementation | ISA_KWNG_KVBINIT |
| 0x001F | Kernel TIC: Unknown tic code | File corrupted | Compare with workbench files | ISA_KWNG_TICDEC |
| 0x0020 | Unknown data type on conversion | File corrupted | Compare with workbench files | ISA_KWNG_TICCNV |
| 0x0021 | TIC Boundary check check error | Access out of range in the variable array | Review the workbench application | ISA_KWNG_TICBNDCHK |
| 0x0022 | Kernel TIC: SINT divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICSINTDIVZ |

| Kernel Warning | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x0023 | Kernel TIC: DINT divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICDINTDIVZ |
| 0x0024 | Kernel TIC: REAL divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICREALDIVZ |
| 0x0025 | Dynamic SFC behaviour: Behaviour processing error | Incorrect implementation | Review the workbench application | ISA_KWNG_SFCEVO |
| 0x0026 | Dynamic SFC behaviour: Action Execution error | Changes made outside the system layer | Compare with the original PRDK | ISA_KWNG_SFCACT |
| 0x0027 | Cycle Time Overflow | Cycle time too low | Increase the cycle time in the workbench application | ISA_KWNG_TCYOVERFLOW |
| 0x0028 | Dynamic SFC behaviour: Initialisation error | Declared in the workbench but not exist in the target | Review the implementation | ISA_KWNG_SFCINIT |
| 0x0029 | Kernel TIC: INT divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICINTDIVZ |
| 0x0030 | Kernel TIC: LINT divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICLINTDIVZ |

| Kernel Warning | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x0031 | Kernel TIC: USINT divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICUSINTDIVZ |
| 0x0032 | Kernel TIC: UINT divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICUINTDIVZ |
| 0x0033 | Kernel TIC: UDINT divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICUDINTDIVZ |
| 0x0034 | Kernel TIC: ULINT divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICULINTDIVZ |
| 0x0035 | Kernel TIC: LREAL divided by zero | Divided by zero | Review the workbench application | ISA_KWNG_TICLREALDIVZ |
| 0x0036 | Kernel TIC: Call stack overflow | The running application requires a call stack depth higher than what is supported by the run-time | Review the application | ISA_KWNG_TICCALLSTKOVERFLOW |
| 0x0037 | Kernel TIC: Soft watch dog called | The execution cycle is higher than the limit defined in the run-time | Review run-time implementation or application | ISA_KWNG_TICSOFTWDOG |

| System Layer | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000001 | SYS: Too many inits have been done | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_SYS_TOOMANYINIT |
| 0x20000002 | SYS: Bad owner number (generaly to high) | Incorrect implementation | Review the implementation | ISA_ER_SYS_OWNERNUM |
| 0x20000003 | SYS: Bad user number (generaly to high) | Incorrect implementation | Review the implementation | ISA_ER_SYS_USERNUM |
| 0x20000004 | SYS: Bad object number (generaly to high) | Incorrect implementation | Review the implementation | ISA_ER_SYS_OBJNUM |
| 0x20000010 | SPC: Invalid space identifier | Incorrect implementation | Review the implementation | ISA_ER_SPC_ID |
| 0x20000011 | SPC: Owner number is not available | Incorrect implementation | Review the implementation | ISA_ER_0x0011 |
| 0x20000012 | SPC: Cannot create memory block | Incorrect implementation | Review the implementation | ISA_ER_SPC_CREATE |
| 0x20000013 | SPC: Cannot create memory block when already exists | Incorrect implementation | Review the implementation | ISA_ER_SPC_CREATE_ALREADYEXIST |
| 0x20000014 | SPC: Cannot delete memory block | Incorrect implementation | Review the implementation | ISA_ER_SPC_DELETE |

| System Layer | | | | |
| --- | --- | --- | --- | --- |
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000015 | SPC: Cannot link with memory block. The memory block has been deleted or does not exist. | Incorrect implementation | Review the implementation | ISA_ER_SPC_LINK |
| 0x20000016 | SPC: Cannot unlink with memory block | Incorrect implementation | Review the implementation | ISA_ER_SPC_UNLINK |
| 0x20000017 | SPC: Cannot save space | Incorrect implementation | Review the implementation | ISA_ER_SPC_SAVE |
| 0x20000018 | SPC: Cannot load space into memory block | Incorrect implementation | Review the implementation | ISA_ER_SPC_LOAD |
| 0x20000019 | SPC: Cannot load space, space does not exist | Incorrect implementation | Review the implementation | ISA_ER_SPC_LOAD_NOTEXIST |
| 0x2000001A | SPC: Cannot remove saved space | Incorrect implementation | Review the implementation | ISA_ER_SPC_BKUP_REMOVE |
| 0x20000020 | SEM: Invalid semaphore identifier | Incorrect implementation | Review the implementation | ISA_ER_SEM_ID |
| 0x20000021 | SEM: Owner number is not available | Incorrect implementation | Review the implementation | ISA_ER_SEM_0x0021 |

| System Layer | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000022 | SEM: Cannot create semaphore | Incorrect implementation | Review the implementation | ISA_ER_SEM_CREATE |
| 0x20000023 | SEM: Semaphore already exists, semaphore already exists | Incorrect implementation | Review the implementation | ISA_ER_SEM_CREATE_ALREADYEXIST |
| 0x20000024 | SEM: Cannot delete semaphore | Incorrect implementation | Review the implementation | ISA_ER_SEM_DELETE |
| 0x20000025 | SEM: Cannot link with semaphore. The sempahore has been deleted or does not exist. | Incorrect implementation | Review the implementation | ISA_ER_SEM_OPEN |
| 0x20000026 | SEM: Cannot close semaphore | Incorrect implementation | Review the implementation | ISA_ER_SEM_CLOSE |
| 0x20000027 | SEM: Cannot take semaphore | Incorrect implementation | Review the implementation | ISA_ER_SEM_TAKE |
| 0x20000028 | SEM: Time out is reached taking semaphore | Incorrect implementation | Review the implementation | ISA_ER_SEM_TAKETIMEOUT |
| 0x20000029 | SEM: Error releasing semaphore | Incorrect implementation | Review the implementation | ISA_ER_SEM_GIVE |

| System Layer | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000030 | MSGQ: Invalid message queue identifier | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_ID |
| 0x20000031 | MSGQ: Cannot create message queue | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_CREATE |
| 0x20000032 | MSGQ: Cannot create message queue, message queue already exists | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_CREATE_ALREADYEXIST |
| 0x20000033 | MSGQ: Cannot create message queue, the size of message queue is too large | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_CREATE_SIZE |
| 0x20000034 | MSGQ: Cannot create message queue, the length of messages is too large | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_CREATE_MSGTOOLONG |
| 0x20000035 | MSGQ: Cannot delete message queue | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_DEL |

| System Layer | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000036 | MSGQ: Cannot link with message queue. The message queue has been deleted or does not exist | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_OPEN |
| 0x20000037 | MSGQ: Cannot close message queue | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_CLOSE |
| 0x20000038 | MSGQ: Cannot send message to message queue | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_SEND |
| 0x20000039 | MSGQ: Cannot send message, time out reached | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_SEND_TIMEOUT |
| 0x2000003A | MSGQ: Cannot send message, message is too large | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_SEND_TOOLONG |
| 0x2000003B | MSGQ: Priority parameter is incorrect, message priority is unknown | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_SEND_PRIORITY |

| System Layer | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x2000003C | MSGQ: Cannot read message from message queue | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_RCV |
| 0x2000003D | MSGQ: Time out is reached receiving message | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_RCV_TIMEOUT |
| 0x2000003E | MSGQ: The message is discarded. The buffer is too small. | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_DISCARDED |
| 0x2000003F | MSGQ: Cannot send message, no message available from pool | Incorrect implementation | Review the implementation | ISA_ER_MSGQ_SEND_OVERFLOW |
| 0x20000050 | NTF: Cannot install handler routine | Incorrect implementation | Review the implementation | ISA_ER_NTF_INSTALL |
| 0x20000051 | NTF: Cannot open notification | Incorrect implementation | Review the implementation | ISA_ER_NTF_OPEN |
| 0x20000052 | NTF: Cannot send notification, invalid notif signal identifier | Incorrect implementation | Review the implementation | ISA_ER_NTF_SIGNAL |
| 0x20000060 | DSA: Invalid name | Incorrect implementation | Review the implementation | ISA_ER_DSA_NAME |

| System Layer | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x02030061 | DSA: Cannot open DSA | Incorrect implementation | Review the implementation | ISA_ER_DSA_OPEN |
| 0x20000062 | DSA: Cannot remove DSA | Incorrect implementation | Review the implementation | ISA_ER_DSA_REMOVE |
| 0x20000063 | DSA: Cannot create DSA | Incorrect implementation | Review the implementation | ISA_ER_DSA_CREATE |
| 0x20000064 | DSA: Cannot write DSA | Incorrect implementation | Review the implementation | ISA_ER_DSA_WRITE |
| 0x20000065 | DSA: Cannot read DSA | Incorrect implementation | Review the implementation | ISA_ER_DSA_READ |
| 0x20000066 | DSA: DSA does not exist | Incorrect implementation | Review the implementation | ISA_ER_DSA_NOTEXIST |
| 0x20000067 | DSA: DSA does not exist | Incorrect implementation | Review the implementation | ISA_ER_DSA_INIT |
| 0x20000068 | DSA: Error in reading DSA | Incorrect implementation | Review the implementation | ISA_ER_DSA_SEEK |
| 0x20000070 | TSK: Task is not running | Incorrect implementation | Review the implementation | ISA_ER_TSK_NOTRUNNING |
| 0x20000071 | TSK: Cannot create task. | Incorrect implementation | Review the implementation | ISA_ER_TSK_CREATE |
| 0x20000072 | TSK: Cannot terminate task. | Incorrect implementation | Review the implementation | ISA_ER_TSK_TERMINATE |
| 0x20000073 | TSK: Cannot create thread | Incorrect implementation | Review the implementation | ISA_ER_THR_CREATE |

| System Layer | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000074 | TSK: Cannot terminate thread | Incorrect implementation | Review the implementation | ISA_ER_THR_TERMINATE |
| 0x20000075 | TSK: Cannot restart a task | Incorrect implementation | Review the implementation | ISA_ER_TSK_RESTART |
| 0x20000080 | SOC: Socket initialization failed | Incorrect implementation | Review the implementation | ISA_ER_SOC_INIT |
| 0x20000081 | SOC: Cannot create socket | Incorrect implementation | Review the implementation | ISA_ER_SOC_CREATE |
| 0x20000082 | SOC: Cannot bind socket | Incorrect implementation | Review the implementation | ISA_ER_SOC_BIND |
| 0x20000083 | SOC: Cannot listen to socket | Incorrect implementation | Review the implementation | ISA_ER_SOC_LISTEN |
| 0x20000084 | SOC: Cannot accept a socket, connection failed | Incorrect implementation | Review the implementation | ISA_ER_SOC_ACCEPT |
| 0x20000085 | SOC: Invalid address | Incorrect implementation | Review the implementation | ISA_ER_SOC_ADDRESS |
| 0x20000086 | SOC: Cannot connect a socket | Incorrect implementation | Review the implementation | ISA_ER_SOC_CONNECT |
| 0x20000087 | SOC: Connection is broken | Incorrect implementation | Review the implementation | ISA_ER_SOC_BROKEN |
| 0x20000088 | SOC: Error receiving data from socket | Incorrect implementation | Review the implementation | ISA_ER_SOC_RECEIVE |

| System Layer | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000089 | SOC: Error sending data on socket | Incorrect implementation | Review the implementation | ISA_ER_SOC_SEND |
| 0x2000008A | SOC: Change option has failed | Incorrect implementation | Review the implementation | ISA_ER_SOC_OPTION |
| 0x2000008B | SOC: Command not implemented | Incorrect implementation | Review the implementation | ISA_ER_SOC_NOTIMPLEMENTED |

| I/Os | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x02050001UL | Invalid number of errors | Changes made outside the system layer | Compare with the original PRDK | ISA_RC_DSYS_NBROFERR |
| 0x02050002UL | Semaphore can't be created | Incorrect implementation | Review the implementation | ISA_RC_DSYS_SEMCREATE |
| 0x02050003UL | Open semaphore failed | Incorrect implementation | Review the implementation | ISA_RC_DSYS_SEMOPEN |
| 0x02050004UL | Take semaphore failed | Incorrect implementation | Review the implementation | ISA_RC_DSYS_SEMTAKE |
| 0x02050005UL | Space can be created | Incorrect implementation | Review the implementation | ISA_RC_DSYS_SPCCREATE |
| 0x02050006UL | Space cannot be deleted | Incorrect implementation | Review the implementation | ISA_RC_DSYS_SPCDELETE |

| I/Os | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x02050007UL | Link to space failed | Incorrect implementation | Review the implementation | ISA_RC_DSYS_SPCLINK |
| 0x02050008UL | Unlink to space failed | Incorrect implementation | Review the implementation | ISA_RC_DSYS_SPCUNLINK |
| 0x02050009UL | Error if the space failed | Incorrect implementation | Review the implementation | ISA_RC_DSYS_WNGSET |
| 0x0205000AUL | Warning stack is empty | Incorrect implementation | Review the implementation | ISA_RC_DSYS_SPCEMPTY |

| Host System Driver Binding | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000630 | Host System Driver Binding: Incompatible version of binding table (Bad CRC) | CRC mismatch in the data exchanged through the binding | Review the implementation | ISA_ER_HSD_KVB_CRC |
| 0x20000631 | Host System Driver Binding: Produced variables are not refresh since the maximum time allowed | Timeout occured during binding communication over the HSD | Producer has been switched off | ISA_ER_HSD_KVB_TIMEOUT |

| Host System Driver Binding | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000632 | Host System Driver Binding: There is no producer | The producing resource has been stopped or the connection properties are incorrect | Restart the producing resource or review the connection properties | ISA_ER_HSD_KVB_KERNELSTOP |
| 0x20000633 | Host System Driver Binding: Service not implemented | Service not implemented | Review the implementation | ISA_ER_HSD_KVB_SERVICE |

| eXchange Dispatcher (IXD) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x22070001 | IXD: Initialization of IXL failed | Changes made outside the system layer | Compare with the original PRDK | ISA_RC_DIXD_IXLINIT |
| 0x22070002 | IXD: Initialization of IXS failed | Changes made outside the system layer | Compare with the original PRDK | ISA_RC_DIXD_IXSINIT |
| 0x20000400 | IXD: Allocation error. | Incorrect implementation | Review the implementation | ISA_ER_IXD_ALLOC |
| 0x20000401 | IXD: Trying to connect to unknown resource | Not in use | Not applicable | ISA_ER_IXD_RES_NOT_FOUND |

| eXchange Dispatcher (IXD) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000402 | IXD: Network configuration not loaded | Not in use | Not applicable | ISA_ER_IXD_CFG_NOT_LOAD |
| 0x20000403 | IXD: Network configuration not found | Not in use | Not applicable | ISA_ER_IXD_CFG_NOT_FOUND |
| 0x20000404 | IXD: Operation fails due to system error | Dialog not yet established | Call StartDialog before sending requests over IXL | ISA_ER_IXD_SYSTEM |
| 0x20000405 | IXD: Received data for a closed connection | Not in use | Not applicable | ISA_ER_IXD_DATA_TO_CLOSED_CX |
| 0x20000406 | IXD: No more connections are available | No connection available | Increase the value of the IXD_DEFAULT_CHANNELNBR or ISA_CNXNBR define | ISA_ER_IXD_NO_CNX_AVAILABLE |
| 0x20000407 | IXD: Bad connection identifier | Bad connection identifier used over the communication | Disconnect and reconnect to the run-time | ISA_ER_IXD_BAD_CNX_ID |

| eXchange Dispatcher (IXD) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000408 | IXD: Too many pending message at a time | Received too many messages at once | Increase the value of the IXD_DEFAULT_MSGBYCHANNEL define | ISA_ER_IXD_TOOMANY_MSG_ATATIME |
| 0x20000409 | IXD: IXD buffer is too short. | Communication buffer is too small | Increase the value of the IXD_DEFAULT_MSGPROCSZ define | ISA_ER_IXD_MSGOVERFLOW |
| 0x2000040A | IXD: Connection timeout | Communication timeout | Disconnect and reconnect to the run-time | ISA_ER_IXD_CNXTIMEOUT |
| 0x2000040B | IXD: Request timeout | Communication timeout | Disconnect and reconnect to the run-time | ISA_ER_IXD_NIDTIMEOUT |
| 0x02070408 | IXD: Too many pending message at a time. | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXD_TOOMANY_MSG_ATATIME |

| eXchange Layer (IXL) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000500 | ISXL: Memory block allocated for device is too short | File corrupted | Compare with workbench files | ISA_ER_ISXL_CONFIG |
| 0x20000501 | ISXL: Cannot establish connection | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_CONNECT |
| 0x20000502 | ISXL: Cannot remove connection | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_DISCONNECT |
| 0x20000503 | ISXL: Cannot read variables | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_READ |
| 0x20000504 | ISXL: Too late too change device (connection maybe already established) | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_CFG_TOOLATE |
| 0x20000505 | ISXL: Cannot set device parameters | File corrupted | Compare with workbench files | ISA_ER_ISXL_CONFIGPARAM |
| 0x20000506 | ISXL: The memory block allocated for connection is too short | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_CONNECTMEM |
| 0x20000507 | ISXL: Time out | Incorrect implementation | Review the implementation | ISA_ER_ISXL_TIMEOUT |

| eXchange Layer (IXL) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000508 | ISXL: An error occurs during the transport of message. | Incorrect implementation | Review the implementation | ISA_ER_ISXL_TRANSPORTFAILED |
| 0x20000509 | ISXL: The RQ code not corresponding | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_RQ |
| 0x2000050A | ISXL: The maximum capacity of the buffer is reached. | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_OVERFLOW |
| 0x2000050B | ISXL: The notification identifier is wrong | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_NOTIFID |
| 0x2000050C | ISXL: Bad return check or error during the transport | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_RC |
| 0x2000050D | ISXL: Cannot remove connection | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_DELCNX |
| 0x2000050E | ISXL: This function required a header for the buffer. | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_HEADERTOOSMALL |
| 0x2000050F | ISXL: Unknown type | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_UNKNOWNTYPE |

| eXchange Layer (IXL) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000510 | ISXL: Bad index number | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_BADVANUMBER |
| 0x20000511 | ISXL: Start dialog is not allowed (maybe dialog is already established) | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_STRTDLG |
| 0x20000512 | ISXL: Stop dialog is not allowed | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_STPDLG |
| 0x20000513 | ISXL: Start dialog has failed | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_STRTDLGFAILED |
| 0x20000514 | ISXL: An error has occurred during the stop dialog procedure. | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_STPDLGFAILED |
| 0x20000515 | ISXL: Start not in progress | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_STRTINPROG |
| 0x20000516 | ISXL: Stop not in progress | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_STPINPROG |
| 0x20000518 | ISXL: Dialog is not established | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_DLG_NOT_STARTED |

| eXchange Layer (IXL) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000519 | ISXL: Error in variable description | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_VARDESC |
| 0x2000051A | ISXL: Method not provided by the driver or invalid method | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_MTH_NO_AVBLE |
| 0x2000051B | ISXL: Service not provided by the driver or invalid service | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_SERVICE |
| 0x2000051C | ISXL: Size allowed for this variable is too short | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_DATAOVERFLOW |
| 0x2000051D | ISXL: Only one connection by resource and by method allowed | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_CNX_1BYRESOURCE |
| 0x2000051F | ISXL: Bad extra parameters for connection | Incorrect implementation | Review the implementation | ISA_ER_ISXL_CNX_EXTRAPARAM |
| 0x20000520 | ISXL: Request cannot be proceeded, retry later | Incorrect implementation | Review the implementation | ISA_ER_ISXL_NEEDWAIT |
| 0x20000521 | ISXS: Cannot establish connection, no more free IXS connections available | Incorrect implementation | Review the implementation | ISA_ER_ISXS_NO_FREE_CNX |

| eXchange Layer (IXL) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000522 | ISXL: Routing feature is not implemented | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_RTG_NOTIMPLEM |
| 0x20000523 | ISXL: Cannot write variables | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_ISXL_WRITE |
| 0x20000530 | IXL: Invalid IXL identifier | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_ID |
| 0x20000531 | IXL: Too many calls to IXL init. | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_TOOMANYINIT |
| 0x20000532 | IXL: Clients cannot have the same number | Incorrect implementation | Review the implementation | ISA_ER_IXL_INITCLIENTNUM |
| 0x20000533 | IXL: The configuration completed successfully. | All IXL drivers have been initialized properly | | ISA_ER_IXL_REGISTRATIONOK |
| 0x20000534 | IXL: Cannot register the driver, its name is invalid | Incorrect implementation | Review the implementation | ISA_ER_IXL_REGISTERNAME |
| 0x20000535 | IXL: Cannot register the driver, a parameter is NULL | Incorrect implementation | Review the implementation | ISA_ER_IXL_REGISTER |

| eXchange Layer (IXL) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000536 | IXL: Cannot register driver, maximum driver is reached | Incorrect implementation | Review the implementation | ISA_ER_IXL_MAXDRV |
| 0x20000537 | IXL: Cannot configure all drivers | Incorrect implementation. The RSI driver in the target may not be configured. The RSI driver (serial driver ) is not mandatory for an target. | Review the implementation | ISA_ER_IXL_CONFIGDRIVER |
| 0x20000538 | IXL: Invalid connection identifier | Incorrect implementation | Review the implementation | ISA_ER_IXL_BADCNXID |
| 0x20000539 | IXL: Cannot establish connection, maximum connection is reached | Incorrect implementation | Review the implementation | ISA_ER_IXL_MAXCNX |
| 0x2000053A | IXL: Cannot establish connection, driver is unknown | Incorrect implementation | Review the implementation | ISA_ER_IXL_UNKNOWNDRV |
| 0x2000053B | IXL: IXL buffer is too short | Incorrect implementation | Review the implementation | ISA_ER_IXL_MSGOVERFLOW |
| 0x2000053C | IXL: This capability is not implemented. | Incorrect implementation | Review the implementation | ISA_ER_IXL_BADCAPS |

| eXchange Layer (IXL) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x2000053D | IXL: Parameters are bad | Incorrect implementation | Review the implementation | ISA_ER_IXL_PARAM |
| 0x2000053E | IXL: Bad RQ | Incorrect implementation | Review the implementation | ISA_ER_IXL_RQ |
| 0x2000053F | IXL: Kernel problem in executing request | Incorrect implementation | Review the implementation | ISA_ER_IXL_RC |
| 0x20000540 | IXL: Symbol table is not loaded | Incorrect implementation | Review the implementation | ISA_ER_IXL_SYM_NOTLOADED |
| 0x20000541 | IXL: Maximum iteration is reached in symbol management | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_ITERATIONMAX |
| 0x20000542 | IXL: Variable is unknown | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_VAR_UNKNOWN |
| 0x20000543 | IXL: Type or Sub-type is unknown | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_TYP_UNKNOWN |
| 0x20000544 | IXL: Symbols mismatch | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_MISMATCH |
| 0x20000545 | IXL: Symbols mismatch, bad CRC | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_CRC |

| eXchange Layer (IXL) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000546 | IXL: Symbols mismatch, bad resource name | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_RESNAME |
| 0x20000547 | IXL: End of symbols is reached or stop is required | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_END |
| 0x20000548 | IXL: Symbols are corrupted | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_CORRUPTED |
| 0x20000549 | IXL: Symbols are already loaded | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_ALREADYLOADED |
| 0x2000054A | IXL: Symbols are currently loading | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_LOADING |
| 0x2000054B | IXL: Both IXL versions cannot coexist | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_BADVERSION |
| 0x2000054C | IXL: Device is unknown. | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_IXL_SYM_DEV_UNKNOWN |

| eXchange Layer (IXL) | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x2000054D | IXL: Syntax error. | File corrupted | Compare with workbench files | ISA_ER_IXL_SYM_BADSYNTAX |
| 0x2000054E | IXL: Symbols table is incomplete, it is reduced one | File corrupted | Compare with workbench files | ISA_ER_IXL_SYM_NOTCOMPLETE |

| ETCP Binding | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000730 | ETCP-KVB: Host address not resolved | Cannot resolve the IP address provided in the binding configuration | Verify the device connection to the network and review the state of the IP stack | ISA_ER_ETCP_KVB_ADDRESS |
| 0x20000731 | ETCP-KVB: No remote resource found | The resource to be connected is local on the device | Review the application to use the HSD driver for local resources. The ETCP driver is for use with remote resources. | ISA_ER_ETCP_KVB_RES_LOCAL |
| 0x20000732 | ETCP-KVB: ETCP Server is not running | Cannot connect to the ETCP task | Verify that the ETCP task is running | ISA_ER_ETCP_KVB_NO_SERVER |

| ETCP Binding | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000733 | ETCP-KVB: This resource has no variables to bind. | No variables to consume or produce | Review the application | ISA_ER_ETCP_KVB_NO_VARIABLES |
| 0x20000734 | ETCP-KVB: Service not implemented | Service not implemented | Review the implementation | ISA_ER_ETCP_KVB_SERVICE |
| 0x20000735 | ETCP-KVB: This variable is not bound by the ETCP driver. | The variable address cannot be found | Review the application, then rebuild and redownload it | ISA_ER_ETCP_KVB_VA_NOT_FOUND |
| 0x20000736 | ETCP-KVB: This resource is not bound by the ETCP driver. | The resource to bind with cannot be found | Review the application, then rebuild and redownload it | ISA_ER_ETCP_KVB_RES_NOT_FOUND |

| ETCP Task | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000410 | CRU: Bad channel identifier | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_CRU_BAD_CH_ID |
| 0x20000411 | CRU: Channel table full | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_CRU_CH_TABLE_FULL |

| ETCP Task | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000412 | CRU: Connection refused | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_CRU_CX_REFUSED |
| 0x20000413 | CRU: Operation has no effect | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_CRU_NO_EFFECT |
| 0x20000414 | CRU: Attempt to access closed socket | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_CRU_CLOSED_CX |
| 0x20000415 | CRU: No IXD to accept connection | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_CRU_NO_IXD |
| 0x20000416 | CRU: Connection refused (out of space) | Changes made outside the system layer | Compare with the original PRDK | ISA_ER_CRU_OUT_OF_SPACE |
| 0x20000417 | CRU: Bad parameters related to channel operation | Incorrect implementation | Review the implementation | ISA_ER_CRU_BAD_PARAM |
| 0x20000418 | CRU: Server is overloaded: Retry later | Incorrect implementation | Review the implementation | ISA_ER_CRU_OVERLOADED |

| ETCP Task | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000420 | VRU: Error during binding: common Data eXchange Space full | Not enough memory to manage connections | Increase either of the following defines: ETCP_DEF_ESCT_NB_ENTRIES, ETCP_DEF_ISCT_NB_ENTRIES, ETCP_DEF_RBIT_NB_ENTRIES, or ETCP_DEF_RCT_NB_ENTRIES | ISA_ER_VRU_DXS_FULL |
| 0x20000421 | VRU: Cannot link to producer | Incorrect implementation | Review the implementation | ISA_ER_VRU_BIND_FAIL |
| 0x20000422 | VRU: Bad binding parameter | Not in use | Not applicable | ISA_ER_VRU_BAD_PARAM |
| 0x20000701 | TAL: Fail to close socket. | Incorrect implementation | Review the implementation | ISA_ER_TAL_CLOSE |
| 0x20000702 | TAL: Fails to launch ISaGRAF server | Incorrect implementation | Review the implementation | ISA_ER_TAL_MAKESERVER |
| 0x20000703 | TAL: Fail to connect to remote node | Incorrect implementation | Review the implementation | ISA_ER_TAL_CONNECT |
| 0x20000704 | TAL: Can't read in socket | Incorrect implementation | Review the implementation | ISA_ER_TAL_READ |
| 0x20000705 | TAL: Error during remote client connection | Incorrect implementation | Review the implementation | ISA_ER_TAL_ACCEPT |

| ETCP Task | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000706 | TAL: FAIL TO INITIALIZE THE TCP/IP STACK | Incorrect implementation | Review the implementation | ISA_ER_TAL_INITFAIL |
| 0x20000707 | TAL: Fail to change socket status | Incorrect implementation | Review the implementation | ISA_ER_TAL_CHGBLOCKING |
| 0x20000708 | TAL: Broken connection | Incorrect implementation | Review the implementation | ISA_ER_TAL_BROKEN_CX |
| 0x20000709 | TAL: Error during socket write | Incorrect implementation | Review the implementation | ISA_ER_TAL_WRITE |
| 0x2000070A | TAL: Received data are not coherent | Incorrect implementation | Review the implementation | ISA_ER_TAL_BAD_ADR |
| 0x2000070B | TAL: Remote ETCP connection fails | Incorrect implementation | Review the implementation | ISA_ER_TAL_REP_CNX_ERR |
| 0x20000710 | ETCP: ETCP is in Timeout mode | Incorrect implementation | Review the implementation | ISA_ER_ETCP_TIMEOUT |
| 0x20000711 | ETCP: The ETCP server is already connected to a default queue. | Incorrect implementation | Review the implementation | ISA_ER_ETCP_DQ_ALREADY_CONNECTED |
| 0x20000712 | ETCP: The ETCP server is full. | Incorrect implementation | Review the implementation | ISA_ER_ETCP_NO_CNX_AVALAIBLE |

| ISaRSI Task | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000640 | RSI: Cannot initialize serial device | Incorrect implementation | Review the implementation | ISA_ER_RSI_INIT |
| 0x20000641 | RSI: Cannot open serial device | Incorrect implementation | Review the implementation | ISA_ER_RSI_OPEN |
| 0x20000642 | RSI: Cannot read serial device | Incorrect implementation | Review the implementation | ISA_ER_RSI_READ |
| 0x20000643 | RSI: Cannot write serial device | Incorrect implementation | Review the implementation | ISA_ER_RSI_WRITE |
| 0x20000644 | RSI: Bad parameters | Incorrect implementation | Review the implementation | ISA_ER_RSI_BADPARAM |

| Common Errors | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000430 | NET-CFG: There is no network device table. | Incorrect implementation | Review the implementation | ISA_ER_NET_CFG_NOT_LOADED |
| 0x20000431 | NET: Resource not found | Incorrect implementation | Review the implementation | ISA_ER_NET_RES_NOT_FOUND |
| 0x20000432 | NET: Variable conversion failed. | Incorrect implementation | Review the implementation | ISA_ER_NET_CONVFAILED |
| 0x20000440 | IPC: IPC is in Timeout mode. | Incorrect implementation | Review the implementation | ISA_ER_IPC_TIMEOUT |

| Common Errors | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000441 | IPC: The IPC server is already connected to a default queue. | Incorrect implementation | Review the implementation | ISA_ER_IPC_DQ_ALREADY_CONNECTED |
| 0x20000442 | IPC: The IPC server is full. | Incorrect implementation | Review the implementation | ISA_ER_IPC_NO_CNX_AVALAIBLE |
| 0x20008000 | NT IOs: Call to device for direct port access failed | Incorrect implementation | Review the implementation | ISA_WNGWNT_IOSPORTACC |
| 0x20008001 | NT IPL: Package not up to date, it may have hazardous behavior | Incorrect implementation | Review the implementation | ISA_WNGWNT_IPL_GETTGTNM_NOTFOUND |
| 0x20008002 | NT IPL: Package ignored, memory model mismatch | Incorrect implementation | Review the implementation | ISA_WNGWNT_IPL_MODELMISMATCH |
| 0x20008003 | NT IPL: Package ignored, version not compatible | Windows specific version mismatch between the run-time and the loading .dll | Verify the .dll was compiled with the state of the ITGTDEF_NEW_ARRAY_AND_FB define as the run-time loading the .dll | ISA_WNGWNT_IPL_GETTGTVERSION |
| 0x20008004 | NT IPL: Package ignored, Windows cannot load the dll | Could not load DLL | Internal error returned by Windows | ISA_WNGWNT_IPL_DLL_NOTLOADED |

| ISaGRAF 3 Communication | | | | |
|---|---|---|---|---|
| **Code** | **Description** | **Probable Cause** | **Diagnostic** | **#define** |
| 0x20000740 | ISA3: No application or application inactive | Not in use | Not applicable | ISA_ER_ISA3_NO_APPL |
| 0x20000741 | ISA3: Media is busy, retry later | Not in use | Not applicable | ISA_ER_ISA3_MEDIA_BUSY |

Back to top

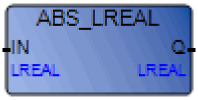
# Functions

The workbench supports the following functions and function blocks:

| | | |
|---|---|---|
| **Arithmetic Operations** | ABS_LREAL | Absolute value of a long real value |
| | EXPT_LREAL, POW_LREAL | Exponent, power calculation of long real values |
| | LOG_LREAL | Logarithm of a long real value |
| | SQRT_LREAL | Square root of a long real value |
| | TRUNC_LREAL | Truncate decimal part of a long real value |
| | ACOS_LREAL, ASIN_LREAL, ATAN_LREAL | Arc cosine, Arc sine, Arc tangent of a long real value |
| | COS_LREAL, SIN_LREAL, TAN_LREAL | Cosine, Sine, Tangent of a long real value |
| **Binary Operations** | AND_MASK_BYTE | BYTE bit-to-bit AND mask |
| | AND_MASK_WORD | WORD bit-to-bit AND mask |
| | AND_MASK_DWORD | DWORD bit-to-bit AND mask |
| | AND_MASK_LWORD | LWORD bit-to-bit AND mask |
| | OR_MASK_BYTE | BYTE bit-to-bit OR mask |
| | OR_MASK_WORD | WORD bit-to-bit OR mask |
| | OR_MASK_DWORD | DWORD bit-to-bit OR mask |
| | OR_MASK_LWORD | LWORD bit-to-bit OR mask |
| | XOR_MASK_BYTE | BYTE bit-to-bit Exclusive OR mask |
| | XOR_MASK_WORD | WORD bit-to-bit Exclusive OR mask |
| | XOR_MASK_DWORD | DWORD bit-to-bit Exclusive OR mask |
| | XOR_MASK_LWORD | LWORD bit-to-bit Exclusive OR mask |
| | NOT_MASK_BYTE | BYTE bit-to-bit negation |
| | NOT_MASK_WORD | WORD bit-to-bit negation |

| | | |
|---|---|---|
| | NOT_MASK_DWORD | DWORD bit-to-bit negation |
| | NOT_MASK_LWORD | LWORD bit-to-bit negation |
| | ROL_BYTE, ROR_BYTE | Rotate Left, Rotate Right a BYTE value |
| | ROL_WORD, ROR_WORD | Rotate Left, Rotate Right a WORD value |
| | ROL_DWORD, ROR_DWORD | Rotate Left, Rotate Right a DWORD value |
| | ROL_LWORD, ROR_LWORD | Rotate Left, Rotate Right an LWORD value |
| | SHL_BYTE, SHR_BYTE | Shift Left, Shift Right a BYTE value |
| | SHL_WORD, SHR_WORD | Shift Left, Shift Right a WORD value |
| | SHL_DWORD, SHR_DWORD | Shift Left, Shift Right a DWORD value |
| | SHL_LWORD, SHR_LWORD | Shift Left, Shift Right an LWORD value |
| **Process Control** | SET_PRIORITY | Set virtual machine priority |
| **Serial Communications** | ISA_SERIAL_CLOSE | Closes the communication port |
| | ISA_SERIAL_CONNECT | Performs a serial connection with an RS-232 or TCP-IP link |
| | ISA_SERIAL_DISCONNECT | Disconnects the communication link |
| | ISA_SERIAL_OPEN | Opens a communication link |
| | ISA_SERIAL_RECEIVE | Receives data from the communication link |
| | ISA_SERIAL_SEND | Sends data on the communication link |
| | ISA_SERIAL_SET | Sets the parameters of an open communication link |
| | ISA_SERIAL_STATUS | Returns a series of communication statuses |
| **String Manipulation** | GET_TIME_STRING | String representing the current time |

# ABS_LREAL




Arguments:

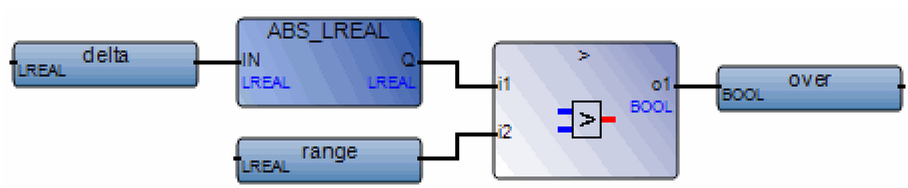
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Any signed long real value |
| ABS_LREAL | Q | LREAL | Absolute long real value (always positive) |

Description:

Gives the absolute (positive) value of a long real value.

## Example

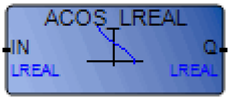
(* FBD Program using "ABS_LREAL" Function *)




(* ST Equivalence: *)

```
over := (ABS_LREAL (delta) > range);
```

# ACOS_LREAL




Arguments:

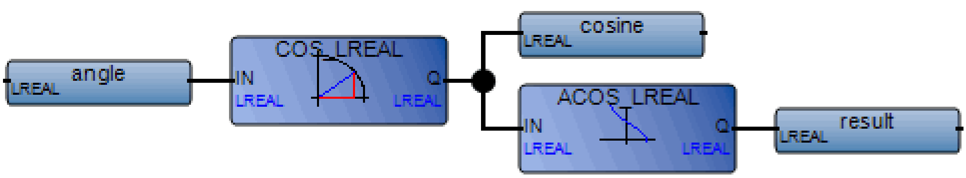
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Must be in set [-1.0 .. +1.0] |
| ACOS_LREAL | Q | LREAL | Arc-cosine of the input value (in set [0.0 .. PI])<br>= 0.0 for invalid input |

Description:

Calculates the Arc cosine of a long real value.

## Example

(* FBD Program using "COS_LREAL" and "ACOS_LREAL" Functions *)




(* ST Equivalence: *)

```
cosine := COS_LREAL (angle);

result := ACOS_LREAL (cosine); (* result is equal to angle *)
```

# AND_MASK_BYTE


Arguments:

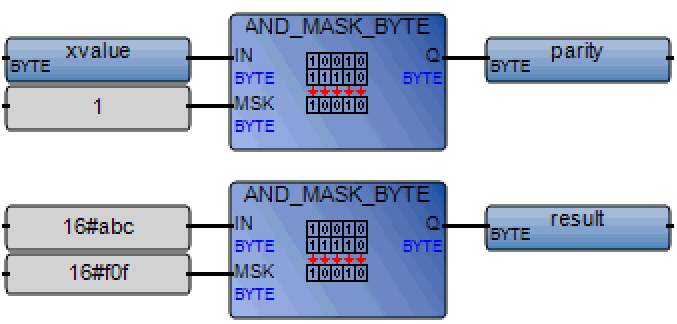
| | | | |
|---|---|---|---|
| IN | IN | BYTE | Must have BYTE format |
| MSK | MSK | BYTE | Must have BYTE format |
| AND_MASK_BYTE | Q | BYTE | Bit-to-bit logical **AND** between IN and MSK |

Description:

BYTE AND bit-to-bit mask.

### Example

(* FBD example with AND_MASK_BYTE Operators *)




(* ST Equivalence: *)

```
parity := AND_MASK_BYTE (xvalue, 1); (* 1 if xvalue is odd *)
result := AND_MASK_BYTE (16#abc, 16#f0f); (* equals 16#a0c *)
```

## AND_MASK_DWORD


Arguments:

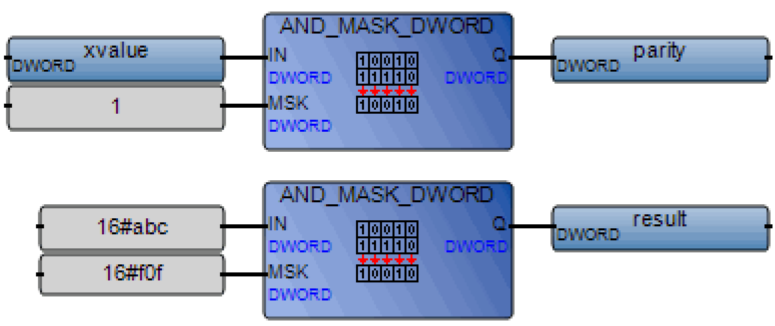
| | | | |
|---|---|---|---|
| IN | IN | DWORD | Must have DWORD format |
| MSK | MSK | DWORD | Must have DWORD format |
| AND_MASK_DWORD | Q | DWORD | Bit-to-bit logical **AND** between IN and MSK |

Description:

DWORD AND bit-to-bit mask.

### Example

(* FBD example with AND_MASK_DWORD Operators *)




(* ST Equivalence: *)

```
parity := AND_MASK_DWORD (xvalue, 1); (* 1 if xvalue is odd *)
result := AND_MASK_DWORD (16#abc, 16#f0f); (* equals 16#a0c *)
```

# AND_MASK_LWORD


Arguments:

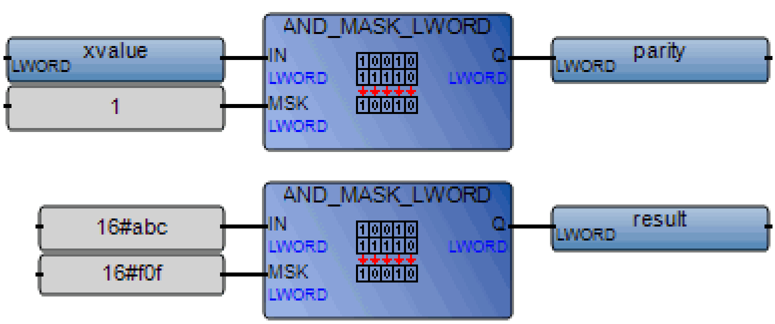
| | | | |
|---|---|---|---|
| IN | IN | LWORD | Must have LWORD format |
| MSK | MSK | LWORD | Must have LWORD format |
| AND_MASK_LWORD | Q | LWORD | Bit-to-bit logical **AND** between IN and MSK |

Description:

LWORD AND bit-to-bit mask.

### Example

(* FBD example with AND_MASK_LWORD Operators *)




(* ST Equivalence: *)

```
parity := AND_MASK_LWORD (xvalue, 1); (* 1 if xvalue is odd *)
result := AND_MASK_LWORD (16#abc, 16#f0f); (* equals 16#a0c *)
```

# AND_MASK_WORD


Arguments:

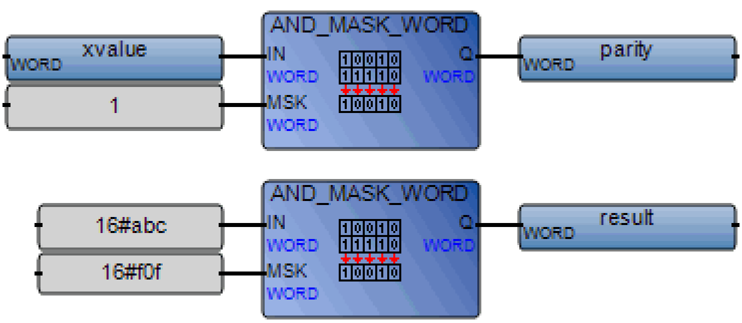
| | | | |
|---|---|---|---|
| IN | IN | WORD | Must have WORD format |
| MSK | MSK | WORD | Must have WORD format |
| AND_MASK_WORD | Q | WORD | Bit-to-bit logical **AND** between IN and MSK |

Description:

WORD AND bit-to-bit mask.

### Example

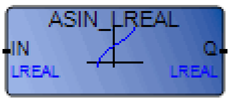
(* FBD example with AND_MASK_WORD Operators *)




(* ST Equivalence: *)

```
parity := AND_MASK_WORD (xvalue, 1); (* 1 if xvalue is odd *)
result := AND_MASK_WORD (16#abc, 16#f0f); (* equals 16#a0c *)
```

# ASIN_LREAL




Arguments:

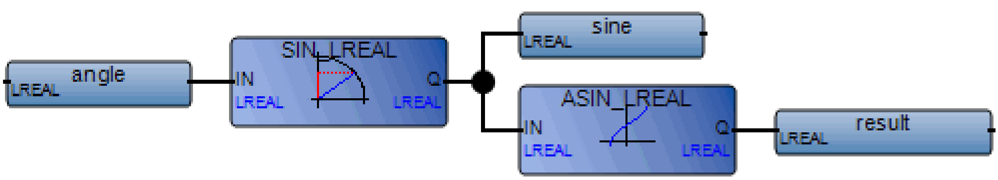
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Must be in set [-1.0 .. +1.0] |
| ASIN_LREAL | Q | LREAL | Arc-sine of the input value (in set [-PI/2 .. +PI/2])<br>= 0.0 for invalid input |

Description:

Calculates the Arc sine of a long real value.

## Example

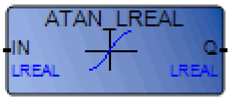
(* FBD Program using "SIN_LREAL" and "ASIN_LREAL" Functions *)




(* ST Equivalence: *)

```
sine := SIN_LREAL (angle);
result := ASIN_LREAL (sine); (* result is equal to angle *)
```

# ATAN_LREAL




Arguments:

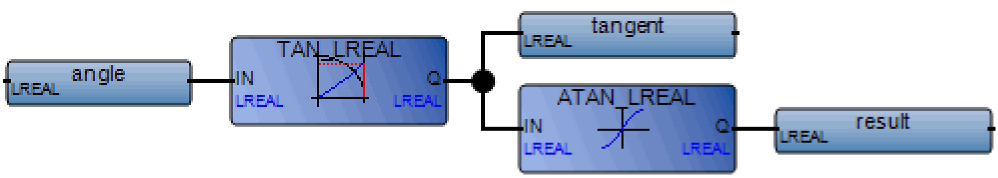
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Any LREAL value |
| ATAN_LREAL | Q | LREAL | Arc-tangent of the input value (in set [-PI/2 .. +PI/2])<br>= 0.0 for invalid input |

Description:

Calculates the arc tangent of a long real value.

### Example

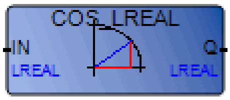
(* FBD Program using "TAN_LREAL" and "ATAN_LREAL" Function *)




(* ST Equivalence: *)

```
tangent := TAN_LREAL (angle);

result := ATAN_LREAL (tangent); (* result is equal to angle*)
```

# COS_LREAL




Arguments:

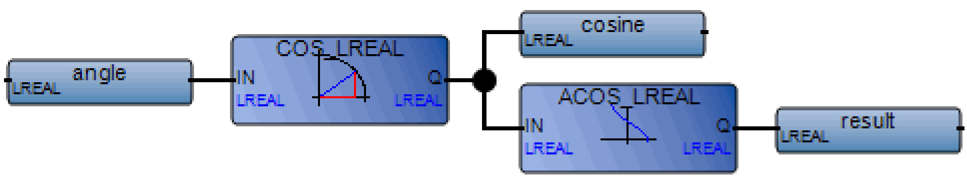
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Any LREAL value |
| COS_LREAL | Q | LREAL | Cosine of the input value (in set [-1.0 .. +1.0]) |

Description:

Calculates the cosine of a long real value.

## Example

(* FBD Program using "COS_LREAL" and "ACOS_LREAL" Functions *)




(* ST Equivalence: *)

```
cosine := COS_LREAL (angle);
result := ACOS_LREAL (cosine); (* result is equal to angle *)
```

# EXPT_LREAL


Arguments:

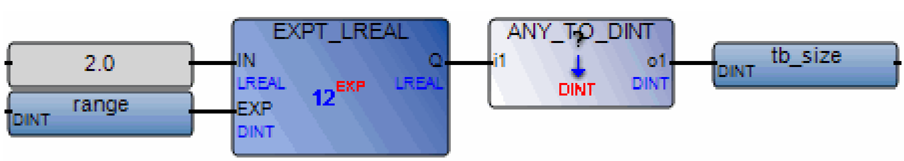
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Any signed long real value |
| EXP | EXP | DINT | Integer exponent |
| EXPT_LREAL | Q | LREAL | $(IN^{EXP})$ |

Description:

Gives the long real result of the operation: (base $^{exponent}$) 'base' being the first argument and 'exponent' the second one.

## Example

(* FBD Program using "EXPT_LREAL" Function *)




(* ST Equivalence: *)

```
tb_size := ANY_TO_DINT (EXPT_LREAL (2.0, range) );
```

# GET_TIME_STRING


Arguments:

| | | | |
|---|---|---|---|
| SEC | SEC | DINT | Number of seconds since 1970/01/01 00:00:00:000 |
| NSEC | NSEC | DINT | Number of nanoseconds from the beginning of the second indicated by SEC |
| GET_TIME_STRING | Q | STRING | Date, in the YYYY/MM/DD HH:MM:SS:MMM format |

Description:

Transforms a date given in seconds to a text format and adjusts the time to match the time zone settings on your computer. The GET_TIME_STRUCT and NOW function blocks also perform time-related operations.

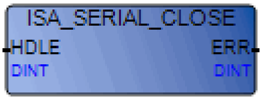
The date

### Example

(* ST equivalence: NOW1 is an instance of the NOW block. *)

```
NOW1();

number_seconds := NOW1.SEC;

number_nanos := NOW1.NSEC;

cur_date := GET_TIME_STRING(number_seconds, number_nanos);
```

# ISA_SERIAL_CLOSE




**Note:** The failover mechanism does not support the ISA_SERIAL functions.

Arguments:

| | | | |
|---|---|---|---|
| HANDLE | HDLE | DINT | handle of the communication link |
| ISA_SERIAL_CLOSE | ERR | DINT | status of the operation:<br>0 = operation succeeded<br>-1 = operation failed |

Description:

Closes the communication port, causing the PCP_SER administrator to terminate or the PCP_IP administrator and data sockets to close.

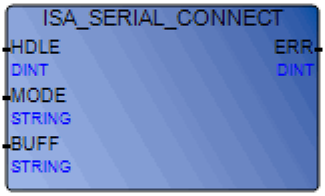
The workbench simulator does not support this function.

### Example

To close the communication port for the `HDLE` communication link:

```
ISA_SERIAL_CLOSE(HDLE);
```

# ISA_SERIAL_CONNECT




**Note:** The failover mechanism does not support the ISA_SERIAL functions.

Arguments:

| | | | |
|---|---|---|---|
| HANDLE | HDLE | DINT | handle of the communication link |
| MODE | MODE | STRING [6] | connection mode: 'SERVER' or 'CLIENT' |

<table>
<tr><td>BUFFER</td><td>BUFF</td><td>STRING [252]</td><td>information required for the connection. This information varies depending on the protocol and connection mode. Four cases can occur:
<table>
<tr><td>PCP_SER</td><td>CLIENT</td><td>RTS/DTR signals are asserted. To perform a connection by modem, enter the required commands and the self-dial telephone number in BUFF. To perform an immediate connection (NULL MODEM), put an empty string in BUFF.</td></tr>
<tr><td>PCP_SER</td><td>SERVER</td><td>RTS/DTR signals are asserted. To indicate a valid connection, you can use either the CTS/DSR signal (by putting an empty string in BUFF) or the modem’s DCD signal (by putting any string in BUFF).</td></tr>
<tr><td>PCP_IP</td><td>CLIENT</td><td>You have to insert the server’s host name in BUFF. A connection is established with the server, using the host name (hosts).</td></tr>
<tr><td>PCP_IP</td><td>SERVER</td><td>You need to insert an empty string in BUFF. The host name for the server is defined in the hosts file.</td></tr>
</table></td></tr>
</table>

| | | |
|---|---|---|
| ISA_SERIAL_CONNECT ERR | DINT | status of the operation:<br>0 = operation succeeded<br>-1 = operation failed |

Description:

Performs a serial connection with an RS-232 or TCP-IP link.

The workbench simulator does not support this function.

## Example

To make on a valid connection in the SERVER connection mode:

```
error := ISA_SERIAL_CONNECT(handle, 'SERVER', '');
errorBool := LOG_MSG('ErrLog', 'Connect: '+ ANY_TO_STRING (error));
IF error = 0 THEN
(* No error: Proceed with the next step*)
END_IF;
```

To make a valid connection in the CLIENT connection mode:

```
error := ISA_SERIAL_CONNECT(handle, 'CLIENT', 'hostname');
errorBool := LOG_MSG('ErrLog', 'Connect: '+ ANY_TO_STRING (error));
IF error = 0 THEN
(* No error: Proceed with the next step*)
ELSE
error := ISA_SERIAL_CLOSE(handle);
END_IF;
```

# ISA_SERIAL_DISCONNECT


**Note:** The failover mechanism does not support the ISA_SERIAL functions.

Arguments:

| | | | |
|---|---|---|---|
| HANDLE | HDLE | DINT | handle of the communication link |
| FLUSH | FLSH | STRING[5] | indicates whether the data transmission must be completed before stopping the communication:<br>'FLUSH' complete the transmission<br>' ' disregard the completion of the transmission. Any value having a maximum of five characters. |
| ISA_SERIAL_DISCONNECT | ERR | DINT | status of the operation:<br>0 = operation succeeded<br>-1 = operation failed |

Description:

Disconnects the communication link.

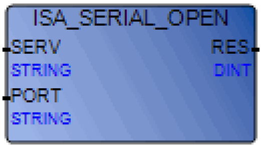
The workbench simulator does not support this function.

### Examples

To complete the transmission before stopping the HDLE communication link and place the status of the operation in the ERR variable:
ERR:= ISA_SERIAL_DISCONNECT(HDLE, 'FLUSH');

To immediately disconnect the HDLE communication link disregarding the completion of the transmission:
ISA_SERIAL_DISCONNECT(HDLE, '');

# ISA_SERIAL_OPEN




**Note:** The failover mechanism does not support the ISA_SERIAL functions.

Arguments:

| | | | |
|---|---|---|---|
| SERVER | SERV | STRING[1] | administrator used: 'PCP_SER' or 'PCP_IP' |
| PORT | PORT | STRING[252] | varies depending on the administrator used in SERVER:<br>PCP_SER, enter the name of the serial device<br>PCP_IP, enter the IP port number of the server |
| ISA_SERIAL_OPEN | RES | DINT | handle of the communication link |

Description:

**Warning:** This function uses the Malloc dynamic memory allocation at run time.

Opens a communication link. You can start an RS-232 (PCP_SER) or TCP/IP (PCP_IP) link. Each time a communication link is opened, a communication administrator is started.

The workbench simulator does not support this function.

### Examples

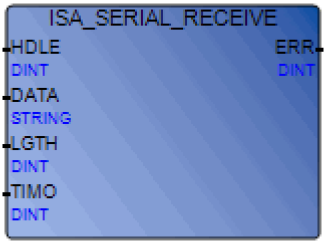
To open a communication link using the PCP_SER protocol:

```
handle := ISA_SERIAL_OPEN('PCP_SER','/dev/ser1');
errorBool := LOG_MSG('ErrLog', 'Open: '+ ANY_TO_STRING (handle));
IF handle > 0 THEN
(* No error: Proceed with the next step*)
END_IF;
```

To open a communication link using the PCP_IP protocol:

```
handle := ISA_SERIAL_OPEN('PCP_IP', '7500');
errorBool := LOG_MSG('ErrLog', 'Open: ' + ANY_TO_STRING (error));
IF handle > 0 THEN
(* No error: Proceed with the next step*)
END_IF;
```

# ISA_SERIAL_RECEIVE




**Note:** The failover mechanism does not support the ISA_SERIAL functions.

Arguments:

| | | | |
|---|---|---|---|
| HANDLE | HDLE | DINT | handle of the communication link |
| DATA | DATA | STRING[252] | received information |
| LENGTH | LGTH | DINT | length of the data, in bytes. The maximum length is 252 bytes. |
| TIMEOUT | TIMO | DINT | maximum number of seconds during which a receive block occurs |
| ISA_SERIAL_RECEIVE | ERR | DINT | status of the operation:<br>0 = operation succeeded<br>-1 = operation failed |

Description:

**Warning:** This function uses the Malloc dynamic memory allocation at run time.

Receives data from the communication link. Reception stops when either the specified number of bytes or the time-out is reached. If data contains a character string that will be used as such, you must make sure that it finishes with a null terminator.

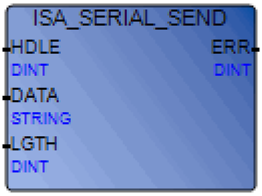
The workbench simulator does not support this function.

### Examples

To receive data using a communication link:

```
error := ISA_SERIAL_RECEIVE(handle, data, 11, 0);
errorBool := LOG_MSG('ErrLog', 'Received data: '+ data);
IF error = -1 THEN
error := ISA_SERIAL_STATUS(handle, SocketError, stat1, stat2, stat3);
errorBool := LOG_MSG('ErrLog', 'Received error: '+ ANY_TO_STRING
(SocketError));
END_IF;
```

# ISA_SERIAL_SEND




**Note:** The failover mechanism does not support the ISA_SERIAL functions.

Arguments:

| | | | |
|---|---|---|---|
| HANDLE | HDLE | DINT | handle of the communication link |
| DATA | DATA | STRING[252] | information to be transmitted |
| LENGTH | LGTH | DINT | length of the data, in bytes. The maximum length is 252 bytes. |
| ISA_SERIAL_SEND | ERR | DINT | status of the operation:<br>0 = operation succeeded<br>-1 = operation failed |

Description:

**Warning:** This function uses the Malloc dynamic memory allocation at run time.

Sends data on the communication link.

The workbench simulator does not support this function.

### Example

To send the string `'Hello world'` on a communication link:

```
error := ISA_SERIAL_SEND(handle, 'Hello world', 11);
IF error = -1 THEN
error := ISA_SERIAL_STATUS(handle, SocketError, stat1, stat2, stat3);
errorBool := LOG_MSG('ErrLog', 'Sent error: '+
ANY_TO_STRING(SocketError));
ELSE
errorBool := LOG_MSG('ErrLog', 'Data Sent: Hello World');
END_IF;
```

# ISA_SERIAL_SET


**Note:** The failover mechanism does not support the ISA_SERIAL functions.

Arguments:

| | | | |
|---|---|---|---|
| HANDLE | HDLE | DINT | handle of the communication link |
| ARG1 | ARG1 | DINT | content varies depending on the protocol used:<br>For PCP_SER, handshake, echo, and trace *<br>For PCP_IP, trace<br>OFF = 0<br>ON = 1 |
| ARG2 | ARG2 | DINT | baud rate. Only used with PCP_SER protocol. |
| ARG3 | ARG3 | DINT | number of stop bits (1 or 2). Only used with PCP_SER protocol. |
| ARG4 | ARG4 | STRING[8] | parity: even, odd, none. Only used with PCP_SER protocol. |
| ISA_SERIAL_SET | ERR | DINT | status of the operation:<br>0 = operation succeeded<br>-1 = operation failed |

Description:

Sets the parameters of an open communication link. These parameters vary according to the protocol and the serial communication standard.

The workbench simulator does not support this function.

* For PCP_SER, the value of **ARG1** varies according to the serial communications standard. If RS-232 is used, **ARG1** holds the state of the trace: 1 = ON, 0 = OFF. On the other hand, with RS-485, **ARG1** holds the composite states of handshake, echo, and trace:

| Handshake | Echo | Trace | ARG1 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

### Examples

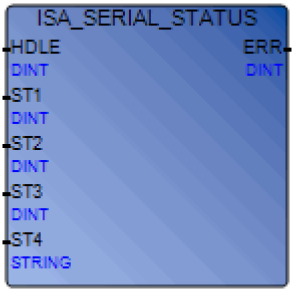
To set the parameters of the `HDLE` communication link using the PCP_SER protocol and the RS-232 serial communication standard without trace, having a baud rate of 9600, 8 bits, and even parity:

```
ERR:= ISA_SERIAL_SET(HDLE, 0, 9600, 8, 'even');
```

To set the parameters of the HDLE communication link using the PCP_IP protocol with a trace:

```
ERR:= ISA_SERIAL_SET(HDLE, 1, 0, 0, '');
```

# ISA_SERIAL_STATUS




**Note:** The failover mechanism does not support the ISA_SERIAL functions.

Arguments:

| | | | |
|---|---|---|---|
| HANDLE | HDLE | DINT | handle of the communication link |
| STA1 | STA1 | DINT | error number. Refer to the target operating system's errno.h file. |
| STA2 | STA2 | DINT | varies depending on the protocol used:<br>For PCP_SER, number of received characters<br>For PCP_IP, port number of the client if in server mode, or port number of the server if in client mode |
| STA3 | STA3 | DINT | CD control bit. Only used for the PCP_SER protocol. |
| STA4 | STA4 | STRING [252] | address of the client if in server mode, or address of the server if in client mode. Only used for the PCP_IP protocol. |
| ISA_SERIAL_STATUS | ERR | DINT | status of the operation:<br>0 = operation succeeded<br>-1 = operation failed |

Description:

Returns a series of communication statuses. These statuses vary depending on the protocol.

The workbench simulator does not support this function.

### Examples

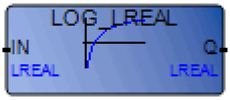
To get the communication statuses of the `HDLE` communication link using the PCP_SER protocol and place them in their respective variables:

```
ERR:= ISA_SERIAL_STATUS(HDLE, STA1, STA2, STA3, STA4);
```

```
To get the communication statuses of the HDLE communication link using
the PCP_IP protocol and place them in their respective variables:
```

```
ERR:= ISA_SERIAL_STATUS(HDLE, STA1, STA2, STA3, STA4);
```

# LOG_LREAL




Arguments:

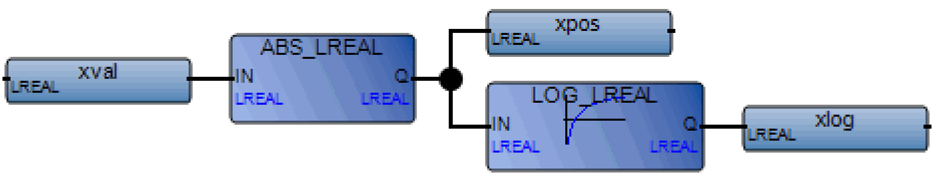
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Must be greater than zero |
| LOG_LREAL | Q | LREAL | Logarithm (base 10) of the input value |

Description:

Calculates the logarithm (base 10) of a long real value.

## Example

(* FBD Program using "LOG_LREAL" Function *)




(* ST Equivalence: *)

```
xpos := ABS_LREAL (xval);
xlog := LOG_LREAL (xpos);
```

# NOT_MASK_BYTE


Arguments:

| | | | |
|---|---|---|---|
| IN | IN | BYTE | Must have BYTE format |
| NOT_MASK_BYTE | Q | BYTE | Bit-to-bit negation on 8 bits of IN |

Description:

BYTE bit-to-bit negation mask.

## Example

(* FBD example with NOT_MASK_BYTE Operators *)


(*ST equivalence: *)

```
result := NOT_MASK_BYTE (16#1234);
(* result is 16#FFFF_EDCB *)
```

# NOT_MASK_DWORD


Arguments:

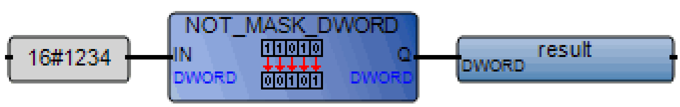
| | | | |
|---|---|---|---|
| IN | IN | DWORD | Must have DWORD format |
| NOT_MASK_DWORD | Q | DWORD | Bit-to-bit negation on 32 bits of IN |

Description:

DWORD bit-to-bit negation mask.

### Example

(* FBD example with NOT_MASK_DWORD Operators *)




(*ST equivalence: *)

```
result := NOT_MASK_DWORD (16#1234);
(* result is 16#FFFF_EDCB *)
```

# NOT_MASK_LWORD


Arguments:

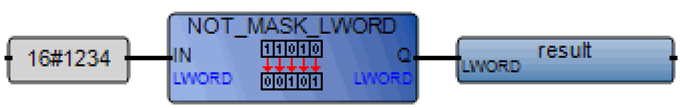
| | | | |
|---|---|---|---|
| IN | IN | LWORD | Must have LWORD format |
| NOT_MASK_LWORD | Q | LWORD | Bit-to-bit negation on 64 bits of IN |

Description:

LWORD bit-to-bit negation mask.

## Example

(* FBD example with NOT_MASK_LWORD Operators *)




(*ST equivalence: *)

```
result := NOT_MASK_LWORD (16#1234);
(* result is 16#FFFF_EDCB *)
```

# NOT_MASK_WORD


Arguments:

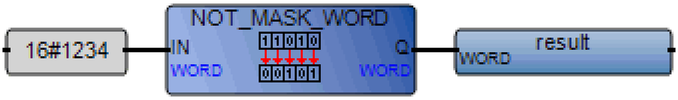
| | | | |
|---|---|---|---|
| IN | IN | WORD | Must have WORD format |
| NOT_MASK_WORD | Q | WORD | Bit-to-bit negation on 16 bits of IN |

Description:

WORD bit-to-bit negation mask.

## Example

(* FBD example with NOT_MASK_WORD Operators *)




(*ST equivalence: *)

```
result := NOT_MASK_WORD (16#1234);
(* result is 16#FFFF_EDCB *)
```

# OR_MASK_BYTE


Arguments:

| | | | |
|---|---|---|---|
| IN | IN | BYTE | Must have BYTE format |
| MSK | MSK | BYTE | Must have BYTE format |
| OR_MASK_BYTE | Q | BYTE | Bit-to-bit logical **OR** between IN and MSK |

Description:

BYTE OR bit-to-bit mask.

## Example

(* FBD example with OR_MASK_BYTE Operators *)



(* ST Equivalence: *)

```
parity := OR_MASK_BYTE (xvalue, 1); (* makes value always odd *)
result := OR_MASK_BYTE (16#abc, 16#f0f); (* equals 16#fbf *)
```

# OR_MASK_DWORD


Arguments:

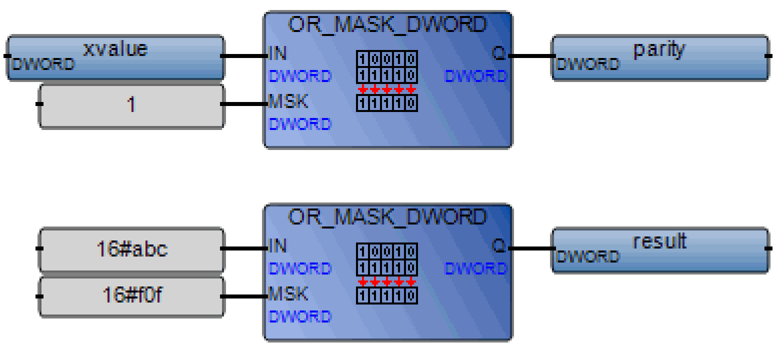
| | | | |
|---|---|---|---|
| IN | IN | DWORD | Must have DWORD format |
| MSK | MSK | DWORD | Must have DWORD format |
| OR_MASK_DWORD | Q | DWORD | Bit-to-bit logical **OR** between IN and MSK |

Description:

DWORD OR bit-to-bit mask.

### Example

(* FBD example with OR_MASK_DWORD Operators *)




(* ST Equivalence: *)

```
parity := OR_MASK_DWORD (xvalue, 1); (* makes value always odd *)
result := OR_MASK_DWORD (16#abc, 16#f0f); (* equals 16#fbf *)
```

# OR_MASK_LWORD


Arguments:

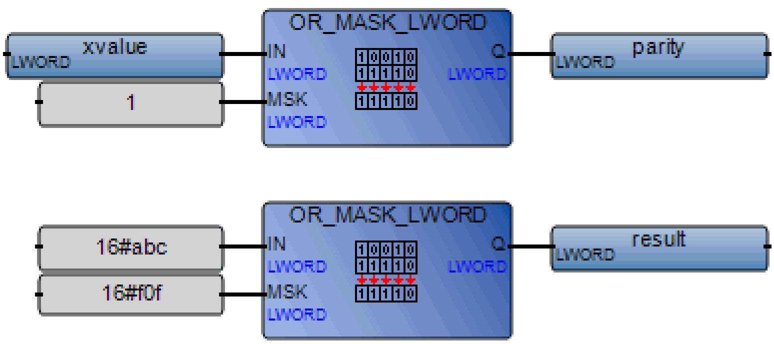
| | | | |
|---|---|---|---|
| IN | IN | LWORD | Must have LWORD format |
| MSK | MSK | LWORD | Must have LWORD format |
| OR_MASK_LWORD | Q | LWORD | Bit-to-bit logical **OR** between IN and MSK |

Description:

LWORD OR bit-to-bit mask.

### Example

(* FBD example with OR_MASK_LWORD Operators *)




(* ST Equivalence: *)

```
parity := OR_MASK_LWORD (xvalue, 1); (* makes value always odd *)
result := OR_MASK_LWORD (16#abc, 16#f0f); (* equals 16#fbf *)
```

# OR_MASK_WORD


Arguments:

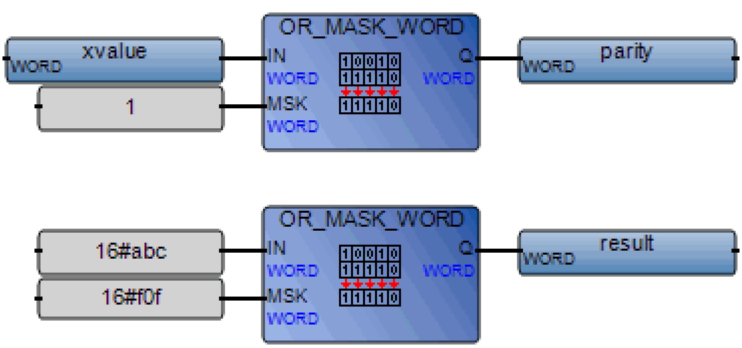
| | | | |
|---|---|---|---|
| IN | IN | WORD | Must have WORD format |
| MSK | MSK | WORD | Must have WORD format |
| OR_MASK_WORD | Q | WORD | Bit-to-bit logical **OR** between IN and MSK |

Description:

WORD OR bit-to-bit mask.

### Example

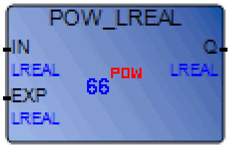
(* FBD example with OR_MASK_WORD Operators *)




(* ST Equivalence: *)

```
parity := OR_MASK_WORD (xvalue, 1); (* makes value always odd *)
result := OR_MASK_WORD (16#abc, 16#f0f); (* equals 16#fbf *)
```

# POW_LREAL




Arguments:

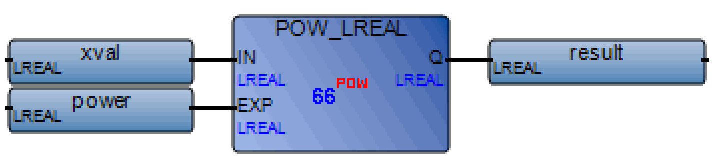
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Long real number to be raised |
| EXP | EXP | LREAL | Power (exponent) |
| POW_LREAL | Q | LREAL | (IN $^{EXP}$)<br>1.0 if IN is not 0.0 and EXP is 0.0<br>0.0 if IN is 0.0 and EXP is negative<br>0.0 if both IN and EXP are 0.0<br>0.0 if IN is negative and EXP does not correspond to an integer |

Description:

Gives the long real result of the operation: (base $^{exponent}$) 'base' being the first argument and 'exponent' the second one. The exponent is a long real value.

### Example

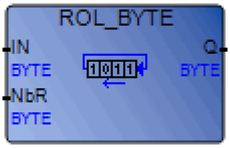
(* FBD Program using "POW_LREAL" Function *)




(* ST Equivalence: *)

```
result := POW_LREAL (xval, power);
```

# ROL_BYTE

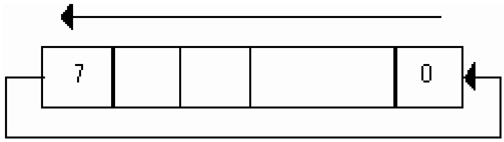



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | BYTE | Any BYTE value |
| NbR | NbR | BYTE | Number of 1 bit rotations (in set [1..7]) |
| ROL_BYTE | Q | BYTE | Left rotated value |

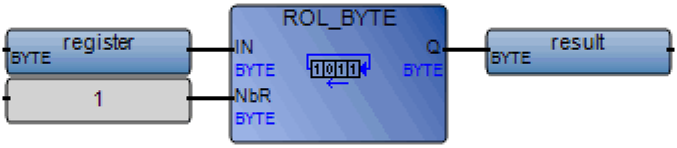
Description:

Make the bits of an BYTE rotate to the left. Rotation is made on 8 bits:




### Example

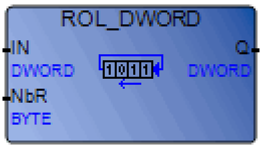
(* FBD Program using "ROL_BYTE" Function *)




(* ST Equivalence: *)

```
result := ROL_BYTE (register, 1);
(* register = 2#1011_0101*)
(* result = 2#0110_1011*)
```

# ROL_DWORD

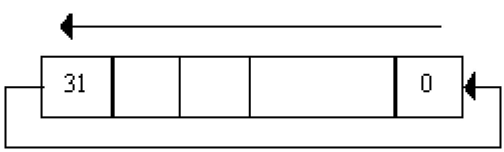



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DWORD | Any DWORD value |
| NbR | NbR | DWORD | Number of 1 bit rotations (in set [1..31]) |
| ROL_DWORD | Q | DWORD | Left rotated value |

Description:

Make the bits of a DWORD rotate to the left. Rotation is made on 32 bits:



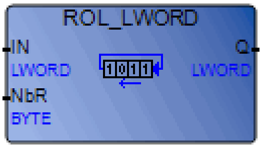

### Example

(* FBD Program using "ROL_DWORD" Function *)


(* ST Equivalence: *)

```
result := ROL_DWORD (register, 1);
(* register = 2#1100_0110_0111_0100_1101_0011_0101_0000*)
(* result = 2#1000_1100_1110_1001_1010_0110_1010_0001*)
```

## ROL_LWORD

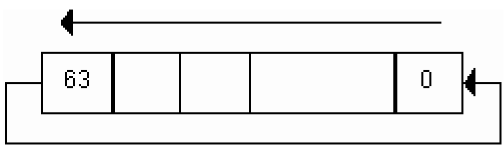



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | LWORD | Any LWORD value |
| NbR | NbR | LWORD | Number of 1 bit rotations (in set [1..63]) |
| ROL_LWORD | Q | LWORD | Left rotated value |

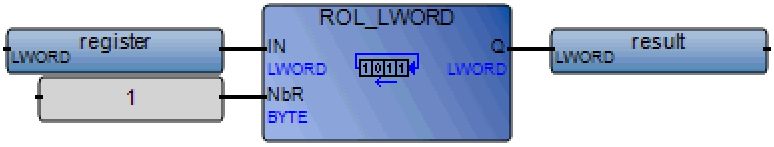
Description:

Make the bits of a LWORD rotate to the left. Rotation is made on 64 bits:




### Example

(* FBD Program using "ROL_LWORD" Function *)




(* ST Equivalence: *)

```
result := ROL_LWORD (register, 1);
(* register = 2#1100_0110_0111_..._1101_0011_0101*)
(* result = 2#1000_1100_1110_..._1010_0110_1011*)
```

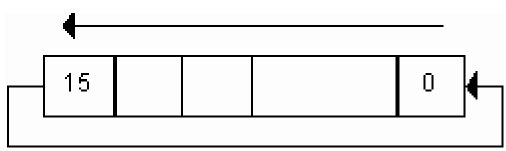
# ROL_WORD


Arguments:

| | | | |
|---|---|---|---|
| IN | IN | WORD | Any WORD value |
| NbR | NbR | WORD | Number of 1 bit rotations (in set [1..15]) |
| ROL_WORD | Q | WORD | Left rotated value |

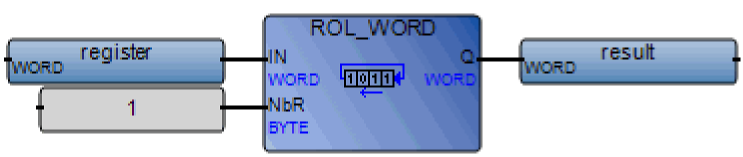
Description:

Make the bits of a WORD rotate to the left. Rotation is made on 16 bits:




## Example

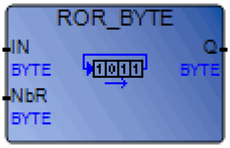
(* FBD Program using "ROL_WORD" Function *)




(* ST Equivalence: *)

```
result := ROL_WORD (register, 1);
(* register = 2#0100_1101_0011_0101*)
(* result = 2#1001_1010_0110_1010*)
```

# ROR_BYTE

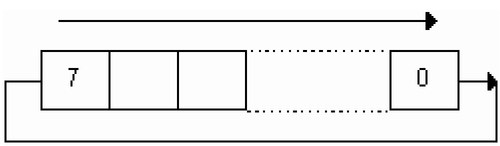



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | BYTE | Any BYTE value |
| NbR | NbR | BYTE | Number of 1 bit rotations (in set [1..7]) |
| ROR_BYTE | Q | BYTE | Right rotated value |

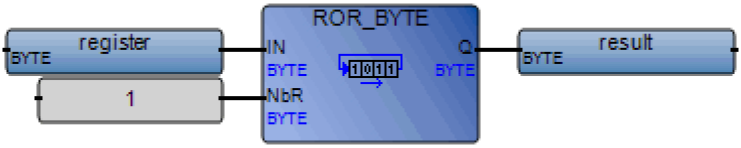
Description:

Make the bits of a BYTE rotate to the right. Rotation is made on 8 bits:




### Example

(* FBD Program using "ROR_BYTE" Function *)




(* ST Equivalence: *)

```
result := ROR_BYTE (register, 1);
(* register = 2#0011_0101 *)
(* result = 2#1001_1010 *)
```

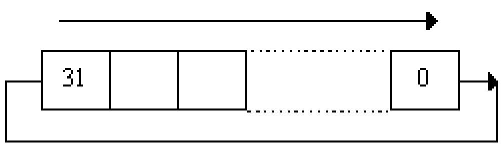
# ROR_DWORD


Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DWORD | Any DWORD value |
| NbR | NbR | DWORD | Number of 1 bit rotations (in set [1..31]) |
| ROR_DWORD | Q | DWORD | Right rotated value |

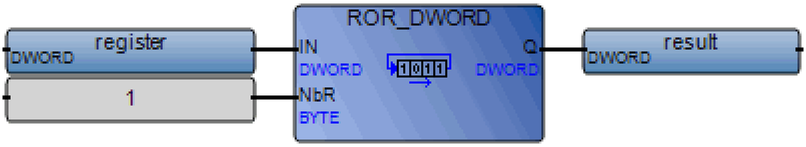
Description:

Make the bits of a DWORD rotate to the right. Rotation is made on 32 bits:




### Example

(* FBD Program using "ROR_DWORD" Function *)




(* ST Equivalence: *)

```
result := ROR_DWORD (register, 1);
(* register = 2#0111_0101_1100_0001_0100_1101_0011_0101 *)
(* result = 2#1011_1010_1110_0000_1010_0110_1001_1010 *)
```

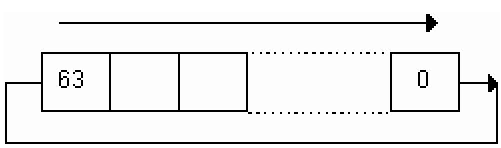
# ROR_LWORD


Arguments:

| | | | |
|---|---|---|---|
| IN | IN | LWORD | Any LWORD value |
| NbR | NbR | LWORD | Number of 1 bit rotations (in set [1..63]) |
| ROR_LWORD | Q | LWORD | Right rotated value |

Description:

Make the bits of an LWORD rotate to the right. Rotation is made on 64 bits:




## Example

(* FBD Program using "ROR_LWORD" Function *)


(* ST Equivalence: *)

```
result := ROR_LWORD (register, 1);
(* register = 2#0111_0101_1100_0001_..._1101_0011_0101 *)
(* result = 2#1011_1010_1110_0000_..._0110_1001_1010 *)
```

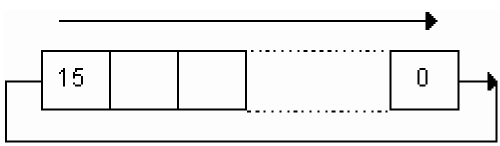
# ROR_WORD


Arguments:

| | | | |
|---|---|---|---|
| IN | IN | WORD | Any WORD value |
| NbR | NbR | WORD | Number of 1 bit rotations (in set [1..15]) |
| ROR_WORD | Q | WORD | Right rotated value |

Description:

Make the bits of a WORD rotate to the right. Rotation is made on 16 bits:



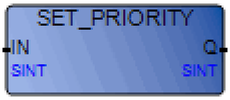

### Example

(* FBD Program using "ROR_WORD" Function *)


(* ST Equivalence: *)

```
result := ROR_WORD (register, 1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1010_0110_1001_1010 *)
```

# SET_PRIORITY




Arguments:

| | | | |
|---|---|---|---|
| INPUT | IN | SINT | New priority for the virtual machine. Possible values are:<br>0: SET_PRIORITY() returns the current virtual machine priority (no change)<br>1-29: new priority for the virtual machine |
| SET_PRIORITY | Q | SINT | priority of the virtual machine before SET_PRIORITY was called |

Description:

Changes the priority of a virtual machine in the target operating system.

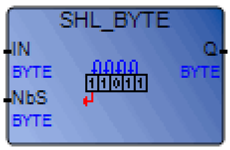
The workbench simulator does not support this function.

### Example

(* ST *)

```
old_priority := SET_PRIORITY(26);
```

# SHL_BYTE

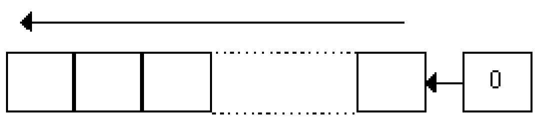



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | BYTE | Any BYTE value |
| NbS | NbS | BYTE | Number of 1 bit shifts (in set [1..7]) |
| SHL_BYTE | Q | BYTE | Left shifted value |

Description:

Make the bits of a BYTE shift to the left. Shift is made on 8 bits:



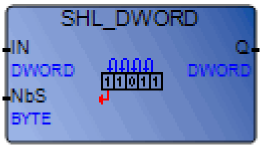

## Example

(* FBD Program using "SHL_BYTE" Function *)


(* ST Equivalence: *)

```
result := SHL_BYTE (register,1);
(* register = 2#0100_1101 *)
(* result = 2#1001_1010 *)
```

# SHL_DWORD

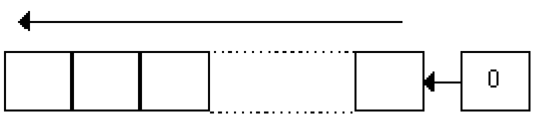



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DWORD | Any DWORD value |
| NbS | NbS | DWORD | Number of 1 bit shifts (in set [1..31]) |
| SHL_DWORD | Q | DWORD | Left shifted value |

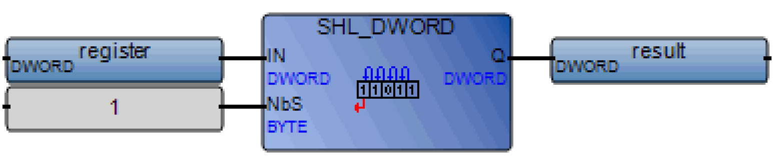
Description:

Make the bits of a DWORD shift to the left. Shift is made on 32 bits:




## Example

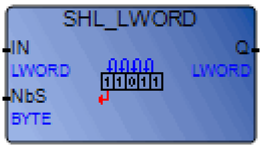
(* FBD Program using "SHL_DWORD" Function *)




(* ST Equivalence: *)

```
result := SHL_DWORD (register,1);
(* register = 2#1010_1100_0011_1010_0100_1101_0011_0101 *)
(* result = 2#0101_1000_0111_0100_1001_1010_0110_1010 *)
```

# SHL_LWORD

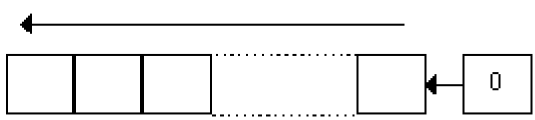



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | LWORD | Any LWORD value |
| NbS | NbS | LWORD | Number of 1 bit shifts (in set [1..63]) |
| SHL_LWORD | Q | LWORD | Left shifted value |

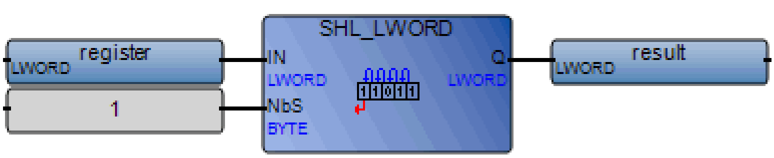
Description:

Make the bits of an LWORD shift to the left. Shift is made on 64 bits:




### Example

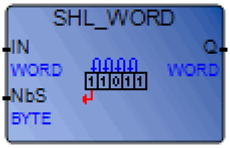
(* FBD Program using "SHL_LWORD" Function *)




(* ST Equivalence: *)

```
result := SHL_LWORD (register,1);
(* register = 2#1010_1100_0011_1010_..._1101_0011_0101 *)
(* result = 2#0101_1000_0111_0100_..._1010_0110_1010 *)
```

# SHL_WORD

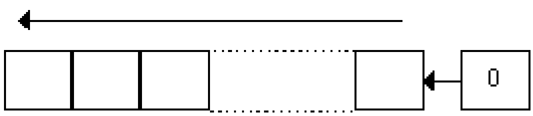



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | WORD | Any WORD value |
| NbS | NbS | WORD | Number of 1 bit shifts (in set [1..15]) |
| SHL_WORD | Q | WORD | Left shifted value |

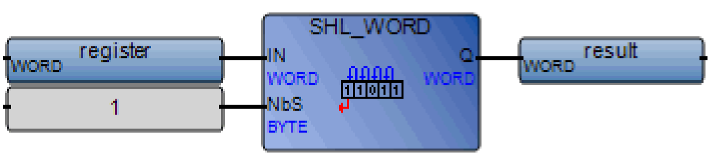
Description:

Make the bits of a WORD shift to the left. Shift is made on 16 bits:




### Example

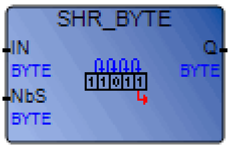
(* FBD Program using "SHL_WORD" Function *)




(* ST Equivalence: *)

```
result := SHL_WORD (register,1);
(* register = 2#0100_1101_0011_0101 *)
(* result = 2#1001_1010_0110_1010 *)
```

# SHR_BYTE

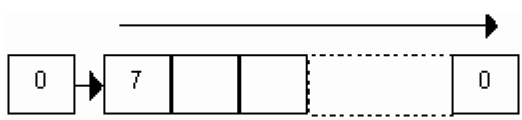



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | BYTE | Any BYTE value |
| NbS | NbS | BYTE | Number of 1 bit shifts (in set [1..7]) |
| SHR_BYTE | Q | BYTE | Right shifted value |

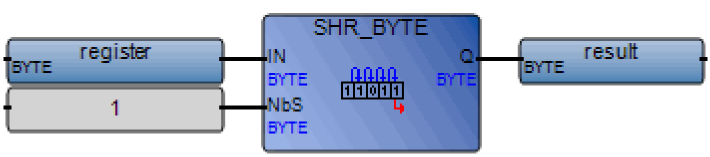
Description:

Make the bits of a BYTE shift to the right. Shift is made on 8 bits:




## Example

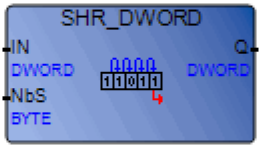
(* FBD Program using "SHR_BYTE"Function *)




(* ST Equivalence: *)

```
result := SHR_BYTE (register,1);
(* register = 2#1100_1101_0011_0101 *)
(* result = 2#0110_0110_1001_1010 *)
```

# SHR_DWORD

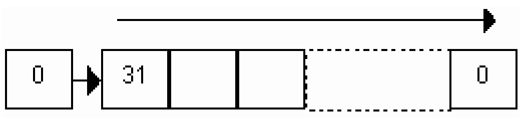



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | DWORD | Any DWORD value |
| NbS | NbS | DWORD | Number of 1 bit shifts (in set [1..31]) |
| SHR_DWORD | Q | DWORD | Right shifted value |

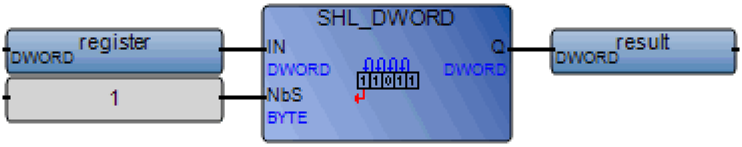
Description:

Make the bits of a DWORD shift to the right. Shift is made on 32 bits:




### Example

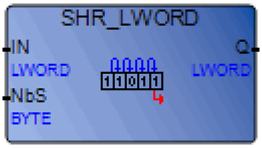
(* FBD Program using "SHR_DWORD"Function *)




(* ST Equivalence: *)

```
result := SHR_DWORD (register,1);
(* register = 2#1010_1100_0001_0101_1100_1101_0011_0101 *)
(* result = 2#0101_0110_0000_1010_1110_0110_1001_1010 *)
```

# SHR_LWORD

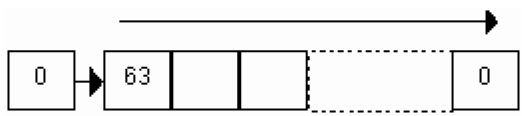



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | LWORD | Any LWORD value |
| NbS | NbS | LWORD | Number of 1 bit shifts (in set [1..63]) |
| SHR_LWORD | Q | LWORD | Right shifted value |

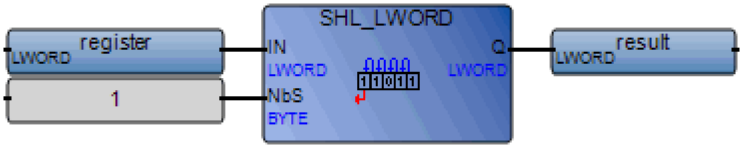
Description:

Make the bits of an LWORD shift to the right. Shift is made on 64 bits:




### Example

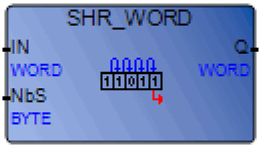
(* FBD Program using "SHR_LWORD"Function *)




(* ST Equivalence: *)

```
result := SHR_LWORD (register,1);
(* register = 2#1010_1100_0001_0101_..._1101_0011_0101 *)
(* result = 2#0101_0110_0000_1010_..._0110_1001_1010 *)
```

# SHR_WORD

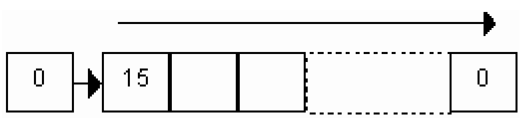



Arguments:

| | | | |
|---|---|---|---|
| IN | IN | WORD | Any WORD value |
| NbS | NbS | WORD | Number of 1 bit shifts (in set [1..15]) |
| SHR_WORD | Q | WORD | Right shifted value |

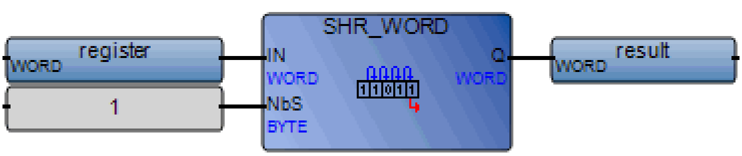
Description:

Make the bits of a WORD shift to the right. Shift is made on 16 bits:




### Example

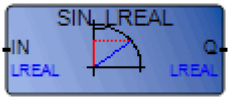
(* FBD Program using "SHR_WORD"Function *)




(* ST Equivalence: *)

```
result := SHR_WORD (register,1);
(* register = 2#1100_1101_0011_0101 *)
(* result = 2#0110_0110_1001_1010 *)
```

# SIN_LREAL




Arguments:

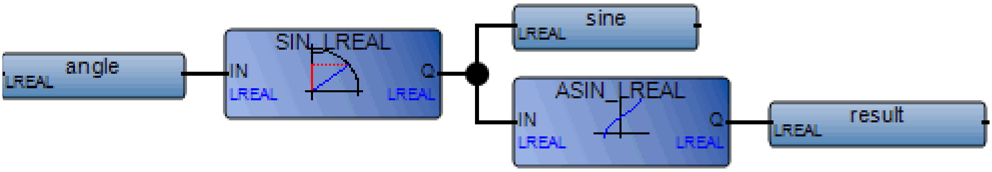
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Any LREAL value |
| SIN_LREAL | Q | LREAL | Sine of the input value (in set [-1.0 .. +1.0]) |

Description:

Calculates the Sine of a long real value.

## Example

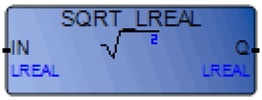
(* FBD Program using "SIN_LREAL" and "ASIN_LREAL" Functions *)




(* ST Equivalence: *)

```
sine := SIN_LREAL (angle);
result := ASIN_LREAL (sine); (* result is equal to angle *)
```

## SQRT_LREAL




Arguments:

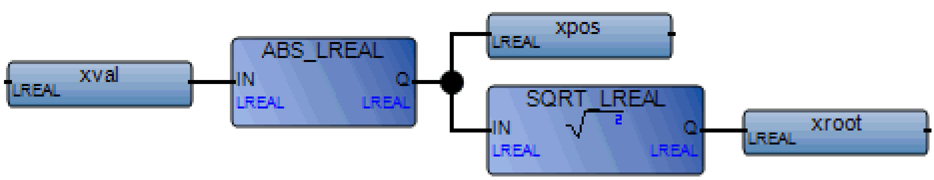
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Must be greater than or equal to zero |
| SQRT_LREAL | Q | LREAL | Square root of the input value |

Description:

Calculates the square root of a long real value.

### Example

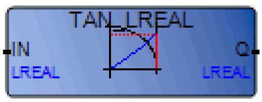
(* FBD Program using "SQRT_LREAL" Function *)




(* ST Equivalence: *)

```
xpos := ABS_LREAL (xval);
xroot := SQRT_LREAL (xpos);
```

# TAN_LREAL




Arguments:

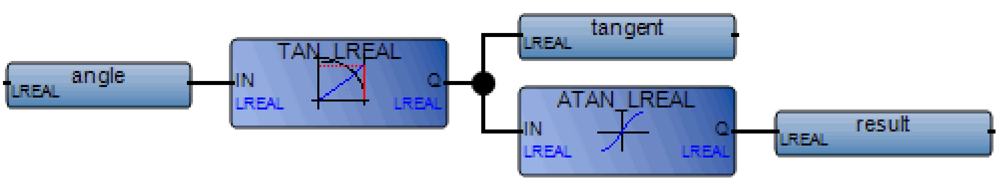
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Cannot be equal to PI/2 modulo PI |
| TAN_LREAL | Q | LREAL | Tangent of the input value<br>= 1E+38 for invalid input |

Description:

Calculates the Tangent of a long real value.

### Example

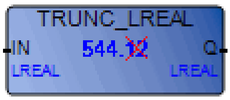
(* FBD Program using "TAN_LREAL" and "ATAN_LREAL" Functions *)




(* ST Equivalence: *)

```
tangent := TAN_LREAL (angle);

result := ATAN_LREAL (tangent); (* result is equal to angle*)
```

# TRUNC_LREAL




Arguments:

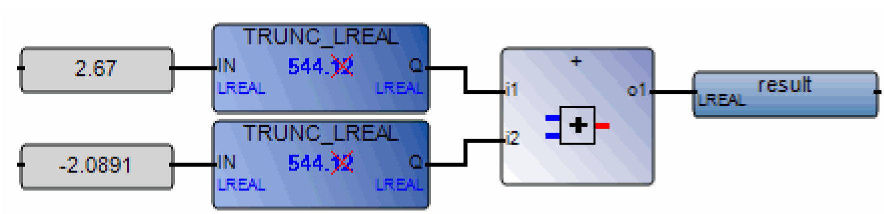
| | | | |
|---|---|---|---|
| IN | IN | LREAL | Any LREAL value |
| TRUNC_LREAL | Q | LREAL | If IN>0, biggest integer less or equal to the input<br>If IN<0, least integer greater or equal to the input |

Description:

Truncates a long real value to have just the integer part.

## Example

(* FBD Program using "TRUNC_LREAL" Function *)




(* ST Equivalence: *)

```
result := TRUNC_LREAL (+2.67) + TRUNC_LREAL (-2.0891);
(* means: result := 2.0 + (-2.0) := 0.0; *)
```

# XOR_MASK_BYTE


Arguments:

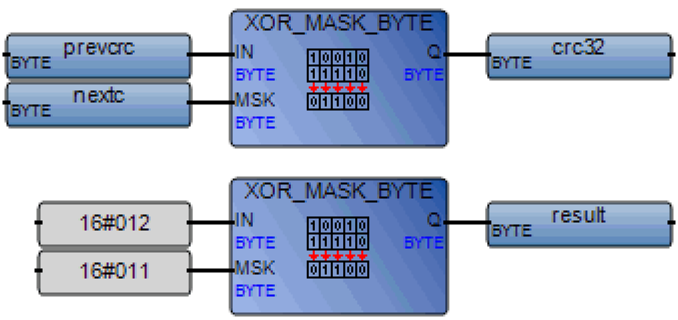
| | | | |
|---|---|---|---|
| IN | IN | BYTE | Must have BYTE format |
| MSK | MSK | BYTE | Must have BYTE format |
| XOR_MASK_BYTE | Q | BYTE | Bit-to-bit logical **Exclusive OR** between IN and MSK |

Description:

BYTE exclusive OR bit-to-bit mask.

### Example

(* FBD example with **XOR_MASK_BYTE** Operators *)




(* ST Equivalence: *)

```
crc32  := XOR_MASK_BYTE (prevcrc, nextc);
result := XOR_MASK_BYTE (16#012, 16#011); (* equals 16#003 *)
```

# XOR_MASK_DWORD


Arguments:

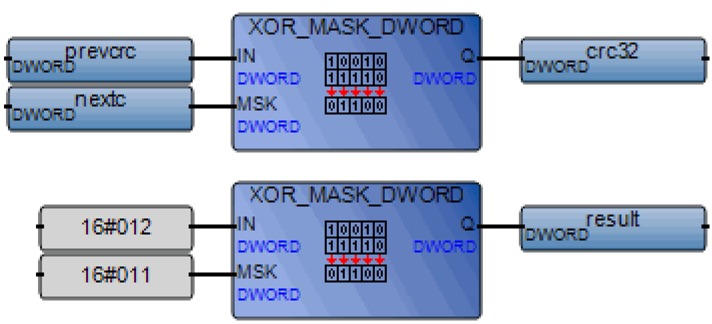
| | | | |
|---|---|---|---|
| IN | IN | DWORD | Must have DWORD format |
| MSK | MSK | DWORD | Must have DWORD format |
| XOR_MASK_DWORD | Q | DWORD | Bit-to-bit logical **Exclusive OR** between IN and MSK |

Description:

DWORD exclusive OR bit-to-bit mask.

### Example

(* FBD example with **XOR_MASK_DWORD** Operators *)




(* ST Equivalence: *)

```
crc32 := XOR_MASK_DWORD (prevcrc, nextc);

result := XOR_MASK_DWORD (16#012, 16#011); (* equals 16#003 *)
```

# XOR_MASK_LWORD


Arguments:

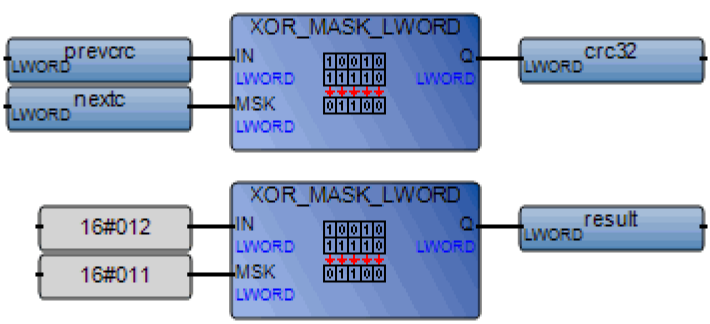
| | | | |
|---|---|---|---|
| IN | IN | LWORD | Must have LWORD format |
| MSK | MSK | LWORD | Must have LWORD format |
| XOR_MASK_LWORD | Q | LWORD | Bit-to-bit logical **Exclusive OR** between IN and MSK |

Description:

LWORD exclusive OR bit-to-bit mask.

### Example

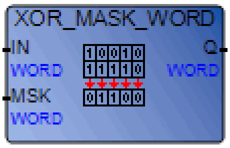
(* FBD example with **XOR_MASK_LWORD** Operators *)




(* ST Equivalence: *)

```
crc32 := XOR_MASK_LWORD (prevcrc, nextc);
result := XOR_MASK_LWORD (16#012, 16#011); (* equals 16#003 *)
```

# XOR_MASK_WORD




Arguments:

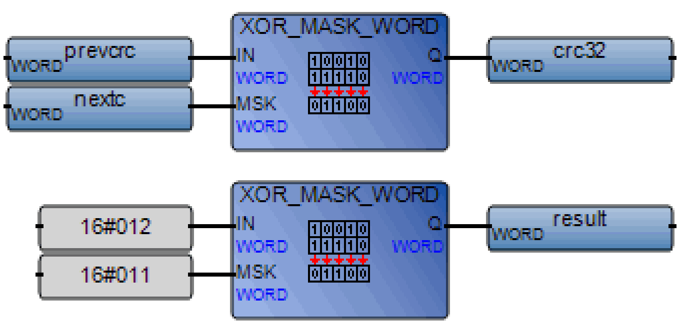
| | | | |
|---|---|---|---|
| IN | IN | WORD | Must have WORD format |
| MSK | MSK | WORD | Must have WORD format |
| XOR_MASK_WORD | Q | WORD | Bit-to-bit logical **Exclusive OR** between IN and MSK |

Description:

WORD exclusive OR bit-to-bit mask.

### Example

(* FBD example with **XOR_MASK_WORD** Operators *)




(* ST Equivalence: *)

```
crc32 := XOR_MASK_WORD (prevcrc, nextc);

result := XOR_MASK_WORD (16#012, 16#011); (* equals 16#003 *)
```

# Function Blocks

**ISaGRAF** supports many types of function blocks:

- Basic Operations
- Advanced Control
- Matrix2 Operations
- Matrix Operations

## Basic Operations

Basic function blocks perform various basic operations:

| | | |
|---|---|---|
| **Time Operations** | GET_TIME_STRUCT | Current time, in the date's parts |
| | NOW | Current time, in seconds |

## Advanced Control

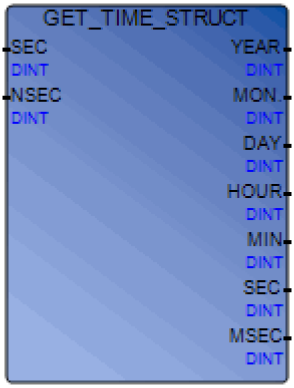
Advanced Control function blocks perform various process control operations:

| | | |
|---|---|---|
| **Alarms Management** | ANALOGALARM | Provides alarm conditions for an analog input |
| | DIGITALALARM | Provides alarm conditions for a digital input |
| **Boolean Operations** | FLIPFLOP | Provides a flip-flop function |
| **Comparator Operations** | COMPARATOR | Compares an input signal with a value and indicates when the value is exceeded |
| **Process Control** | BATCHSWITCH | Eliminates overshoot during startup conditions when using the IPIDCONTROLLER function block |

| | |
|---|---|
| BATCHTOTALIZER | Integrates an analog input with alarms on presets and provides a pulse output to drive a remote counter |
| BIAS | Provides a means to bias a signal, such as the setpoint in an external set application |
| BIASCALIBRATION | Calibrates a BIAS value while tracking an input signal |
| CHARACTERIZER | Provides segments that can characterize an input signal |
| IPIDCONTROLLER | An interacting PID controller |
| LEADLAGCONTROLLER | A lead/lag controller |
| LEADLAGBACONTROLLER | A lead/lag bilinear approximation controller |
| LIMITER | Limits an input value to a range between a low and high limit |
| PID_AL | *To be defined* |
| RATELIMITER | Limits the rate of change for an input signal |
| RATIO | Provides a means of setting a ratio in an external setpoint application |
| RATIOCALIBRATION | Calibrates RATIO by tracking an input signal |
| RETENTIVEONTIMER | Performs an on-delay timing function with output states determined by input values used to start and enable a timer |
| SCALER | Scales an input value according to an output range |

| | |
|---|---|
| SETPOINT | Multi-action setpoint command having six different settings and adjustments of setpoint for controller |
| SIGNALSELECTOR | Selects either the highest or lowest signal value from three input signals |
| TRACKANDHOLD | Holds an initial value transferred to an output on first scan then either tracks the input signal or holds the last output value |
| TRANSFERSWITCH | Selects a signal between two input signals |

# GET_TIME_STRUCT




Arguments:

| | | | |
|---|---|---|---|
| SEC | SEC | DINT | Number of seconds since 1970/01/01 00:00:00:000 |
| NSEC | NSEC | DINT | Number of nanoseconds from the beginning of the second indicated by SEC |
| YEAR | YEAR | DINT | Year of the date, in a four-digit format |
| MONTH | MON. | DINT | Month of the date (1-12) |
| DAY | DAY | DINT | Day of the date (1-31) |
| HOUR | HOUR | DINT | Hour of the date (0-23) |
| MINUTE | MIN | DINT | Minute of the date (0-59) |
| SECOND | SEC | DINT | Second of the date (0-59) |
| MSEC | MSEC | DINT | Millisecond of the date, from the beginning of SECOND (0-999) |

Description:

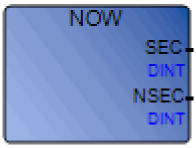
Converts a date into a series of DINT values representing the date's parts. GET_TIME_STRUCT adjusts the time to match the time zone settings on your computer. The GET_TIME_STRING function and NOW function block also perform time-related operations.

### Example

(* ST equivalence: NOW1 is an instance of the NOW block; GET_TIME_STRUCT1 is an instance of the GET_TIME_STRUCT block. *)

```
NOW1();

number_seconds := NOW1.SEC;

number_nanos := NOW1.NSEC;

GET_TIME_STRUCT1(number_seconds, number_nanos);

cur_year := GET_TIME_STRUCT1.YEAR;

cur_month := GET_TIME_STRUCT1.MONTH;

cur_day := GET_TIME_STRUCT1.DAY;

cur_hour := GET_TIME_STRUCT1.HOUR;

cur_minute := GET_TIME_STRUCT1.MINUTE;

cur_second := GET_TIME_STRUCT1.SECOND;

cur_msec := GET_TIME_STRUCT1.MSEC;
```

# NOW




Arguments:

| | | | |
|---|---|---|---|
| SEC | SEC | DINT | Number of seconds since 1970/01/01 00:00:00:000 |
| NSEC | NSEC | DINT | Number of nanoseconds from the beginning of the second indicated by SEC |

Description:

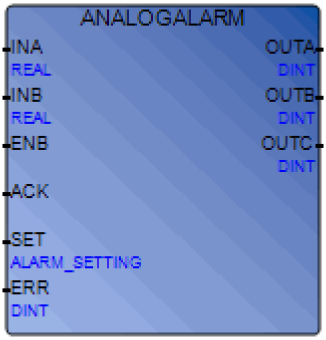
Gets the current time since 1970/01/01 00:00:00:000, in seconds. The GET_TIME_STRING function and GET_TIME_STRUCT function block also perform time-related operations. The ANY_TO_DATE function enables the conversion of NSEC to a date format.

### Example

(* ST equivalence: NOW1 is an instance of the NOW block. *)

```
NOW1();

number_seconds := NOW1.SEC;

number_nanos := NOW1.NSEC;
```

# ANALOGALARM




Arguments:

| | | | |
|---|---|---|---|
| InputA | INA | REAL | Input signal A |
| InputB | INB | REAL | Input signal B for deviation alarms calculation |
| OutputEnable | ENB | BOOL[0..2] | OutputEnable. For each entry, possible values are True or False:<br>0 High/Low Limit, High/Low Alarm, High/Low Warning<br>1 Deviation High and Deviation Low<br>2 Rate of Change Up and Rate of Change Down |
| Acknowledge | ACK | BOOL[0..2] | Acknowledge. For each entry, possible values are True or False:<br>0 High/Low Limit, High/Low Alarm, High/Low Warning<br>1 Deviation High and Deviation Low<br>2 Rate of Change Up and Rate of Change Down |
| AlarmSetting | SET | ALARMSETTING | AlarmSetting.<br>See ALARMSETTING structure |

| | | | |
|---|---|---|---|
| ErrorMode | ERR | DINT | ErrorMode. Mode used to handle errors of the different types:<br><br>RateOfChangePeriod <= 0.0. Possible values are:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets RateOfChangeUpEnable and RateOfChangeDownEnable to FALSE<br>RateOfChangeUp <= 0.0. Possible values are:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets RateOfChangeUpEnable to FALSE<br>RateOfChangeDown <= 0.0. Possible values are:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets RateOfChangeDownEnable to FALSE<br>HighDeviation < 0.0. Possible values are:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets HighDeviationEnable to FALSE<br>LowDeviation < 0.0. Possible values are:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets LowDeviationEnable to FALSE |

| | | | |
|---|---|---|---|
| OutputA | OUTA | DINT | (OutputA) Output for High/Low Limit, High/Low Alarm, High/Low Warning alarms |
| OutputB | OUTB | DINT | (OutputB) Output for Deviation High and Deviation Low alarms |
| OutputC | OUTC | DINT | (OutputC) Output for Rate of Change Up and Rate of Change Down alarms |

Description:

Provides 10 alarm conditions for an analog input. There are three outputs, one for each alarm category: High/Low alarms, deviation alarms, and rate of change alarms.

ALARMSETTING structure:

| | | |
|---|---|---|
| HighLimit | REAL | Value for which InputA exceeds the maximum range |
| HighAlarm | REAL | Value above which InputA is in high alarm condition |
| HighWarning | REAL | Value above which InputA is in warning alarm condition |
| LowWarning | REAL | Value below which InputA is in warning alarm condition |
| LowAlarm | REAL | Value below which InputA is in high alarm condition |
| LowLimit | REAL | Value for which InputA exceed is out of minimum range |
| DeadBand | REAL | Value for which InputA must be changed to get out of alarm condition |
| HighDeviation | REAL | Maximum acceptable difference in value from InputA to InputB |
| LowDeviation | REAL | Maximum acceptable difference in value from InputB to InputA |
| RateOfChangePeriod | REAL | Time interval used to calculate RateOfChange alarms, in seconds |
| RateOfChangeUp | REAL | Maximum increase in value of InputA during the RateOfChangePeriod triggering a rate of change up alarm |

| | | |
|---|---|---|
| RateOfChangeDown | REAL | Minimum decrease in value of InputA during the RateOfChangePeriod triggering a rate of change down alarm |
| DelayInTime | REAL | Minimum period of time, in seconds, during which a condition is present before activating alarms (High and Low Limit, Alarm, Warning, and Deviation) |
| DelayOutTime | REAL | Minimum period of time, in seconds, during which a condition is absent before deactivating alarms (High and Low Limit, Alarm, Warning and Deviation) |
| HighLimitEnable | BOOL | Bit enabling HighLimit alarm check |
| HighAlarmEnable | BOOL | Bit enabling HighAlarm alarm check |
| HighWarningEnable | BOOL | Bit enabling HighWarning alarm check |
| LowWarning Enable | BOOL | Bit enabling LowWarning alarm check |
| LowAlarmEnable | BOOL | Bit enabling LowAlarm alarm check |
| LowLimitEnable | BOOL | Bit enabling LowLimit alarm check |
| HighDeviationEnable | BOOL | Bit enabling HighDeviation alarm check |
| LowDeviationEnable | BOOL | Bit enabling LowDeviation alarm check |
| RateOfChangeUpEnable | BOOL | Bit enabling RateOfChangeUp alarm check |
| RateOfChangeDown Enable | BOOL | Bit enabling RateOfChangeDown alarm check |
| RingBack | BOOL | Bit enabling the Not-Present Not-Acknowledge state when a condition alarm goes out |

High/Low Alarms

| Conditions making high/low alarms switch from not present to present | |
|---|---|
| High Limit | HighLimitEnable is TRUE and InputA has been greater than HighLimit for a period of time greater than DelayInTime |
| High Alarm | HighAlarmEnable is TRUE and InputA has been greater than HighAlarm and lower than HighLimit for a period of time greater than DelayInTime |
| High Warning | HighWarningEnable is TRUE and InputA has been greater than HighWarning and lower than HighAlarm for a period of time greater than DelayInTime |

| Conditions making high/low alarms switch from not present to present | |
|---|---|
| Low Warning | LowWarningEnable is TRUE and InputA has been lower than LowWarning and higher than LowAlarm for a period of time greater than DelayInTime |
| Low Alarm | LowAlarmEnable is TRUE and InputA has been lower than LowAlarm and higher than LowLimit for a period of time greater than DelayInTime |
| Low Limit | LowLimitEnable is TRUE and InputA has been lower than LowLimit for a period of time greater than DelayInTime |

| Conditions making high/low alarms switch from present to not present | |
|---|---|
| High Limit | HighLimitEnable is TRUE and InputA has been lower than HighLimit minus DeadBand for a period of time greater than DelayOutTime |
| High Alarm | HighAlarmEnable is TRUE and InputA has been lower then HighAlarm minus DeadBand for a period of time greater than DelayOutTime |
| High Warning | HighWarningEnable is TRUE and InputA has been lower than HighWarning minus DeadBand for a period of time greater than DelayOutTime |
| Low Warning | LowWarningEnable is TRUE and InputA has been greater than LowWarning plus DeadBand for a period of time greater than DelayOutTime |
| Low Alarm | LowAlarmEnable is TRUE and InputA has been greater than LowAlarm plus DeadBand for a period of time greater than DelayOutTime |
| Low Limit | LowLimitEnable is TRUE and InputA has been greater than LowLimit plus DeadBand for a period of time greater than DelayOutTime |

OutputA Values:

| State | Value | | |
|---|---|---|---|
| **No Alarm** | 0 | | |
| **Present** | yes | yes | no(1) |

| State | Value | | |
|---|---|---|---|
| **Acknowledged** | no | yes | no(1) |
| **HighLimit** | 1 | 11 | 21 |
| **HighAlarm** | 2 | 12 | 22 |
| **HighWarning** | 3 | 13 | 23 |
| **LowWarning** | 4 | 14 | 24 |
| **LowAlarm** | 5 | 15 | 25 |
| **LowLimit** | 6 | 16 | 26 |

When OutputEnable[0] is FALSE, then the value of OutputA equals 0 (no alarm). The alarm is still processed but the value is kept internally.

If RingBack is TRUE, when an alarm state is Present-Acknowledge, the next step is Not-Present-Not-Acknowledge instead of no alarm. This causes a previously acknowledged alarm to require acknowledgment when the alarms clears.

Alarm Priority

| | |
|---|---|
| High Priority | High Limit – Low Limit<br>High Alarm – Low Alarm |
| Low Priority | High Warning – Low Warning |

If the condition of a higher priority alarm is met while the current alarm is not acknowledged, the value of Output will be changed to reflect the higher alarm state.

| Conditions making deviation alarms switch from not present to present | |
|---|---|
| DeviationHigh | HighDeviationEnable is TRUE and InputA has been greater than InputB plus HighDeviation for a period of time greater than DelayInTime |
| DeviationLow | LowDeviationEnable is TRUE and InputA has been lower than InputB minus LowDeviation for a period of time greater than DelayInTime |

| Conditions making deviation alarms switch from present to not present | |
|---|---|
| DeviationHigh | HighDeviationEnable is TRUE and InputA has been lower than InputB plus HighDeviation minus DeadBand for a period of time greater than DelayOutTime |
| DeviationLow | LowDeviationEnable is TRUE and InputA has been greater than InputB minus LowDeviation plus DeadBand for a period of time greater than DelayOutTime |

OutputB values:

| **State** | **Value** | | |
|---|---|---|---|
| **No Alarm** | 0 | | |
| **Present** | yes | yes | no(1) |
| **Acknowledged** | no | yes | no(1) |
| **DeviationHigh** | 1 | 11 | 21 |
| **DeviationLow** | 2 | 12 | 22 |

When OutputEnable[0] is FALSE, then the value of OutputB equals 0 (no alarm). The alarm is still processed but the value is kept internally.

(1) If RingBack is TRUE, when an alarm state is Present-Acknowledge the next step is Not-Present-Not-Acknowledge instead of no alarm. This causes a previously acknowledged alarm to require acknowledgment when the alarms clears.

Rate of Change Alarms

| Conditions making rate of change alarms switch from not present to present | |
|---|---|
| RateOfChangeUp | RateOfChangeUpEnable is TRUE and InputA increases more than the value of RateOfChangeUp during RateOfChangePeriod |
| RateOfChangeDown | RateOfChangeDownEnable is TRUE and InputA decreases more than the value of RateOfChangeDown during RateOfChangePeriod |

| Conditions making rate of change alarms switch from present to not present | |
|---|---|
| RateOfChangeUp | RateOfChangeUpEnable is TRUE and InputA up variation over a period of RateOfChangePeriod is lower then RateOfChangeUp |
| RateOfChangeDown | RateOfChangeDownEnable is TRUE and InputA down variation over a period of RateOfChangePeriod is lower then RateOfChangeDown |

OutputC Values:

| **State** | **Value** | | |
|---|---|---|---|
| **No Alarm** | 0 | | |
| **Present** | yes | yes | no(1) |
| **Acknowledged** | no | yes | no(1) |
| **RateOfChangeUp** | 1 | 11 | 21 |
| **RateOfChangeDown** | 2 | 12 | 22 |

When OutputEnable[0] is FALSE, then the value of OutputC equals 0 (no alarm). The alarm is still processed but the value is kept internally.

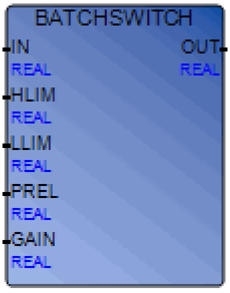
(1) If RingBack is TRUE, when an alarm state is Present-Acknowledge, the next step is Not-Present-Not-Acknowledge instead of no alarm. This will causes a previously acknowledged alarm to require acknowledgment when the alarms clears.

## Example

```
(* ST equivalence: ANALOGALARM1 is an instance of ANALOGALARM block *)
ANALOGALARM1( Signal_InA, Signal_InB, Enable, Ack, AlarmSetting, 0);
CASE ANALOGALARM.OutputA OF
  1: Message1 := 'Alarm High Limit for Signal_InA';
  2: Message1 := 'Alarm High Alarm for Signal_InA';
  3: Message1 := 'Alarm High Warning for Signal_InA';
  4: Message1 := 'Alarm Low Warning for Signal_InA';
  5: Message1 := 'Alarm Low Alarm for Signal_InA';
```

```
  6: Message1 := 'Alarm Low Limit for Signal_InA';
  11: Message1 := 'Alarm High Limit for Signal_InA Acknowledged';
  12: Message1 := 'Alarm High Alarm for Signal_InA Acknowledged';
  13: Message1 := 'Alarm High Warning for Signal_InA Acknowledged';
  14: Message1 := 'Alarm Low Warning for Signal_InA Acknowledged';
  15: Message1 := 'Alarm Low Alarm for Signal_InA Acknowledged';
  16: Message1 := 'Alarm Low Limit for Signal_InA Acknowledged';
  21: Message1 := 'Alarm High Limit for Signal_InA Done';
  22: Message1 := 'Alarm High Alarm for Signal_InA Done';
  23: Message1 := 'Alarm High Warning for Signal_InA Done';
  24: Message1 := 'Alarm Low Warning for Signal_InA Done';
  25: Message1 := 'Alarm Low Alarm for Signal_InA Done';
  26: Message1 := 'Alarm Low Limit for Signal_InA Done';
END_CASE;
```

# BATCHSWITCH




Arguments:

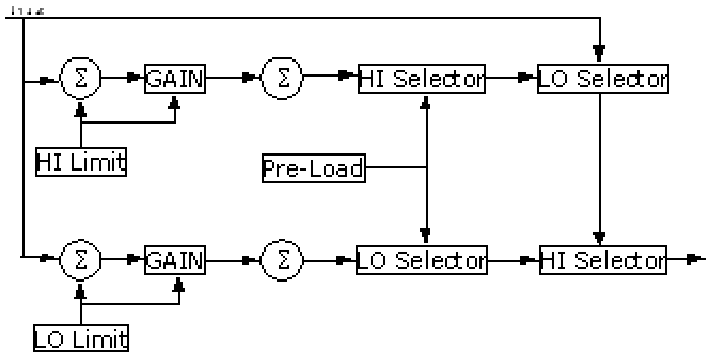
| | | | |
|---|---|---|---|
| Input | IN | REAL | Input signal |
| HighLimit | HLIM | REAL | High limit for input signal |
| LowLimit | LLIM | REAL | Low limit for input signal |
| PreLoad | PREL | REAL | (PreLoad) Limit on adjusting controller feedback signal |
| Gain | GAIN | REAL | Gain value |
| Output | OUT | REAL | Output signal |

Description:

Eliminates overshoot during startup conditions when using the IPIDCONTROLLER function block. When placed in the feedback path of the controller it causes the reset component of the controller to be reduced (if controller action is Reverse). Without the use of batch switch during startup, the controller output will equal full output since the reset will wind up. This requires the process to overshoot the setpoint in order to bring the controller output back down. With a batch switch in the feedback path, a lower reset value will be present when crossover occurs, thus reducing or eliminating overshoot.

As input equals or exceeds the high or low limit setting, the output of the batch switch will either be decreased (HighLimit) or increased (LowLimit), changing the feedback signal and therefore the controller reset signal. This maintains controller output at the batch switch limit setting and eliminates reset windup.

If a controller has a large proportional gain setting, the reset can be modified too much, such that the process may undershoot the setpoint during a startup condition. The PreLoad is adjusted to optimize the controller for startup conditions by limiting how much the batch switch to add additional compensation, very similar to derivative action, only during start up.




**Example**

```
(* ST equivalence: BATCHSWITCH1 is an instance of BATCHSWITCH block *)


BATCHSWITCH1(Feedback_Out_Process, 250.0, 0.0, 50.0, 2.0);

Feedback_In_Pid := BATCHSWITCH1.Output;
```

# BATCHTOTALIZER


Arguments:

| | | | |
|---|---|---|---|
| Input | IN | REAL | Input signal |
| InitialValue | INIT | REAL | Initial value |
| Preset1 | PRE1 | REAL | (Preset1) Value used to activate Alarm1 when Total equals Preset1 |
| Preset2 | PRE2 | REAL | (Preset2) Value used to activate Alarm2 when Total equals Preset2 |
| ZeroDropOut | ZERO | REAL | (ZeroDropOut) Small positive value used as zero point for Input to stop totalling |
| PulseScaling | PSCL | REAL | (PulseScaling) Value to scale Pulse output |

| | | | |
|---|---|---|---|
| TimeBase | TBAS | DINT | TimeBase. Possible values are:<br>1 second<br>2 minute<br>3 hour<br>4 day<br>5 week |
| Stop | STOP | BOOL | Stops totalling |
| Reset | RST | BOOL | (Reset) Reinitialize Total to InitialValue |
| DirectActing | DA | BOOL | (DirectActing) The indication of whether totalling is incremental or decremental:<br>TRUE totalling is incremental<br>FALSE totalling is decremental |
| ErrorMode | ERR | DINT | (ErrorMode) Mode used to handle errors of type TimeBase < 1 or TimeBase > 5. Possible values are:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets Total to 0.0, Alarm1 to TRUE, and Alarm2 to TRUE |
| Total | TOT | REAL | (Total) Batch total value |
| Alarm1 | ALM1 | BOOL | (Alarm1) TRUE when Preset1 is reached |
| Alarm2 | ALM2 | BOOL | (Alarm2) TRUE when Preset2 is reached |
| Pulse | PULS | BOOL | Pulse output integrates the input signal using TimeBase and output pulse at the rate determined by PulseScaling. The Pulse output operates on the absolute value of Input. |

Description:

Integrates an analog input with alarms on presets and provides a pulse output to drive a remote counter.

### Example

```
(* ST equivalence: BATCHTOTALIZER1 is an instance of BATCHTOTALIZER
block *)
```

```
BATCHTOTALIZER1(Signal_In,
       Init_Val,
       PreSet1,
       PreSet2,
       0.0,
       10.0,
       1,
       Stop_Batch,
       Reset_Batch,
       TRUE,
       0);
Pulse_Out := BATCHTOTALIZER1.Pulse ;
Batch_Tot := BATCHTOTALIZER1.Total ;
Done_1 := BATCHTOTALIZER1.Alarm1 ;
Done_2 := BATCHTOTALIZER1.Alarm2 ;
```

# BIAS


Arguments:

| | | | |
|---|---|---|---|
| InputA | INA | REAL | Input signal A |
| InputE | INE | REAL | Input signal E |
| Bias | BIAS | REAL | BIAS value |
| Output | OUT | REAL | Output value. Output = (BIAS) + InputA + InputE. |

Description:

Provides a means to bias a signal, such as the setpoint in an external set application. Input signal A and input signal E are summed and then added to the operator adjustable BIAS. The BIASCALIBRATION function block calibrates BIAS using a tracked input signal.

## Example

```
(* ST equivalence: BIAS1 is an instance of BIAS block and
BIASCALIBRATION1 is an instance of BIASCALIBRATION block *)


BIAS1(Signal_InA, Signal_InE, BIASCALIBRATION1.BIAS);

Out_Value := BIAS1.Output ;
```

# BIASCALIBRATION


Arguments:

| | | | |
|---|---|---|---|
| InputA | INA | REAL | Input signal A |
| InputE | INE | REAL | Input signal E |
| Initial | INIT | REAL | (Initial) BIAS value at first scan |
| High_Limit | HLIM | REAL | High Limit for RATIO |
| Low_Limit | LLIM | REAL | Low Limit for RATIO |
| TrackVariable | TV | REAL | (TrackVariable) Input Signal to track |
| TrackCommand | TC | BOOL | (TrackCommand) Indication of whether the value of TrackVariable is tracked:<br>TRUE TrackVariable's value is tracked<br>FALSE TrackVariable's value is not tracked |
| Bias | BIAS | REAL | BIAS value |
| TrackOutput | TO | REAL | (TrackOutput) Value of TrackOutput dependent on whether TrackCommand is initiated. When TrackCommand is FALSE, TrackOutput equals 0.0. When TrackCommand is TRUE, TrackOutput equals (TrackVariable) - (InputA + BIAS) |

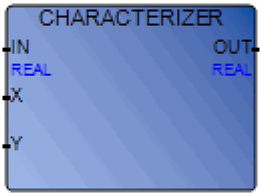
Description:

Calibrates BIAS using TrackVariable. When TrackCommand is FALSE, BIAS equals the last BIAS value and TrackOutput is 0.0. When TrackCommand is TRUE, BIAS = (TrackVariable) - (InputA + InputE); TrackOutput = (TrackVariable) - (InputA + BIAS) also BIAS will be limited by HighLimit and LowLimit. The BIAS function block provides a means to bias a signal such as the setpoint in an external set application.

### Example

```
(* ST equivalence: BIAS1 is an instance of BIAS block and
BIASCALIBRATION1 is an instance of BIASCALIBRATION block *)


BIASCALIBRATION1(Signal_InA,
       Signal_InE,
       0.2,
       300.0,
       10.0,
       Flow_Water,
       TK);
BIAS1(Signal_InA, Signal_InE, BIASCALIBRATION1.BIAS);
Out_Value := BIAS1.Output ;
```

# CHARACTERIZER

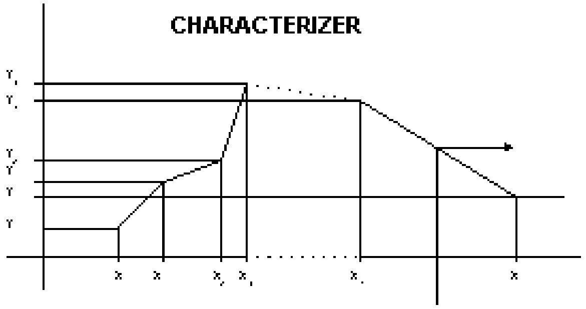



Arguments:

| | | | |
|---|---|---|---|
| Input | IN | REAL | Input X signal |
| X0_10 | X | REAL[0..10] | (X0_X10) Inputs coordinates segments |
| Y0_10 | Y | REAL[0..10] | (Y0_Y10) Outputs coordinates segments |
| Output | OUT | REAL | Output Y signal |

Description:

Provides 10 segments that can characterize the input signal. Segments are configured by entering the Xn, Yn, Xn+1, and Yn+1 points. All Xn+1 points must be greater than the Xn points.




## Example

```
(* ST equivalence: CHARACTERIZER1 is an instance of CHARACTERIZER
block, Table_X and Table_Y are defined as REAL with dimension [0..10]
in dictionary *)
```

```
CHARACTERIZER1( Signal_In, Table_X, Table_Y) ;
Characterized_Value := CHARACTERIZER1.Output ;
```

# COMPARATOR


Arguments:

| | | | |
|---|---|---|---|
| Input | IN | REAL | Input signal |
| LimitValue | LIM | REAL | Limit value |
| DeadBand | DB | REAL | Dead band value depending on setting of DirectActing. When DirectActing is TRUE, the Output switches from TRUE to FALSE when the input is lower than Limit – DeadBand. When DirectActing is FALSE, the Output switches from TRUE to FALSE when the input is more than Limit + DeadBand. |
| DirectActing | DIR | BOOL | (DirectActing) The indication of whether the function block operates in direct acting or reverse acting mode:<br>TRUE block is in direct acting mode and Output is TRUE when Input ≥ Limit<br>FALSE block is in reverse acting mode and Output is TRUE when Input ≤ Limit |
| Output | OUT | BOOL | Output signal |

Description:

Compares the input with a limit value and gives a TRUE output when the limit is exceeded.

### Example

```
(* ST equivalence: COMPARATOR1 is an instance of COMPARATOR block *)

COMPARATOR1(Signal_In , Limit, 5.0 , TRUE ) ;
Limit_Exceeded := COMPARATOR1.Output ;
```

# DIGITALALARM


Arguments:

| | | | |
|---|---|---|---|
| InputA | INA | BOOL | Input signal A |
| OutputEnable | ENB | BOOL | (OutputEnable) Enable alarm processing |
| Acknowledge | ACK | BOOL | Acknowledge signal when TRUE |

| | | | |
|---|---|---|---|
| Mode | MODE | DINT | The conditions triggering an alarm for Output. Possible values are:<br>0 Output goes in alarm when input signal A is TRUE (High state)<br>1 Output goes in alarm when input signal A is FALSE (Low state)<br>2 Output goes in alarm when input signal A changes from FALSE to TRUE (Rising edge)<br>3 Output goes in alarm when input signal A changes from TRUE to FALSE (Falling edge)<br>4 Output goes in alarm when input signal A changes from FALSE to TRUE or TRUE to FALSE (change of state)<br>5 Output goes in alarm when input signal A changes from FALSE to TRUE more than once during Period (Raising Rate Of Change)<br>6 Output go in alarm when input signal A changes from TRUE to FALSE more than once during Period (Falling Rate Of Change) |
| RingBack | RB | BOOL | (RingBack) Bit enabling the Not-Present Not-Acknowledge state when a condition alarm goes out |
| Period | PER | REAL | Period of time to calculate Rate Of Change alarms, in seconds |
| ErrorMode | ERR | DINT | (ErrorMode) Mode used to handle errors of type invalids Mode. Possible values are:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets Output to zero |
| Output | OUT | DINT | (Output) Alarm value = 0 when no alarm and 1 or 11 or 21 in alarm (see Output values below). |

Description:

Provides six alarm conditions for a digital input. Alarm conditions are High state, Low state, Rising edge, Falling edge, Change of state, Rising Rate of change, and Falling Rate of change.

Output values:

| State | Value | | |
|---|---|---|---|
| **No Alarm** | 0 | | |
| **Present** | yes | yes | no(1) |
| **Acknowledged** | no | yes | no(1) |
| **DIGITALALARM Output** | 1 | 11 | 21 |

When OutputEnable is FALSE, then Output equals 0 (no alarm). The alarm is still processed but the value is kept internally.

(1) If RingBack is TRUE, when an alarm state is Present-Acknowledge, the next step is Not-Present-Not-Acknowledge instead of no alarm. This causes a previously acknowledged alarm to require acknowledgment when the alarms clears.

### Example

```
(* ST equivalence: DIGITALALARM1 is an instance of DIGITALALARM block*)


DIGITALALARM1(Digit_InA, Enable, Ack, Mode, RingBack, 10, 0);


CASE Mode OF

  0:


  CASE DIGITALALARM1.Output OF

     1:Message2:= 'Alarm High State for Digit_InA';

     11:Message2:= 'Alarm High State for Digit_InA Acknowledged';

     21:Message2:= 'Alarm High State for Digit_InA Done';

  END_CASE;


  1:
```

```
CASE DIGITALALARM1.Output OF
   1:Message2:= 'Alarm Low State for Digit_InA';
   11:Message2:= 'Alarm Low State for Digit_InA Acknowledged';
   21:Message2:= 'Alarm Low State for Digit_InA Done';
END_CASE;

2:
CASE DIGITALALARM1.Output OF
   1:Message2:='Alarm Rising edge for Digit_InA';
   11:Message2:='Alarm Rising edge for Digit_InA Acknowledged';
   21:Message2:='Alarm Rising edge for Digit_InA Done';
END_CASE;

3:
CASE DIGITALALARM1.Output OF
   1:Message2:='Alarm Falling edge for Digit_InA';
   11:Message2:='Alarm Falling edge for Digit_InA Acknowledged';
   21:Message2:='Alarm Falling edge for Digit_InA Done';
END_CASE;

4:
CASE DIGITALALARM1.Output OF
   1:Message2:='Alarm C.O.S. for Digit_InA';
   11:Message2:='Alarm C.O.S. for Digit_InA Acknowledged';
   21:Message2:='Alarm C.O.S. for Digit_InA Done';
END_CASE;

5:
```

```
  CASE DIGITALALARM1.Output OF
     1:Message2:='Alarm Rising ROC for Digit_InA';
     11:Message2:='Alarm Rising ROC for Digit_InA Acknowledged';
     21:Message2:='Alarm Rising ROC for Digit_InA Done';
  END_CASE;

  6:
  CASE DIGITALALARM1.Output OF
     1:Message2:='Alarm Falling ROC for Digit_InA';
     11:Message2:='Alarm Falling ROC for Digit_InA Acknowledged';
     21:Message2:='Alarm Falling ROC for Digit_InA Done';
  END_CASE;

END_CASE;
```

# FLIPFLOP


Arguments:

| | | | |
|---|---|---|---|
| Set | SET | BOOL | Set input signal |
| Reset | RES | BOOL | Reset input signal |
| Output | OUT | BOOL | Output signal |

Description:

Provides a flip-flop function as detailed in the truth table below:

| R | S | LO | O |
|---|---|---|---|
| 1 | X | X | 0 |
| 0 | ¸ | 1 | 0 |
| 0 | ¸ | 0 | 1 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |

R = Reset input

S = Set input

X = any state

¸ = rising edge

### Example

```
(* ST equivalence: FLIPFLOP1 is an instance of FLIPFLOP block *)
FLIPFLOP1(Reset, Set) ;
```

```
Out_Value := FLIPFLOP1.Output ;
```

# IPIDCONTROLLER


Arguments:

| | | | |
|---|---|---|---|
| Process | P | REAL | Process value, measured from the output of the controller process |
| SetPoint | SP | REAL | Set point value |
| Feedback | FB | REAL | Feed Back signal, measured from control input to a process |
| Auto | AUTO | BOOL | The operation mode of the PID controller:<br>TRUE controller runs in normal mode<br>FALSE controller output value equals feedback value |
| Initialize | INIT | BOOL | A change in value (TRUE to FALSE or FALSE to TRUE) causes the controller to eliminate any proportional gain during that cycle. Also initializes autotune sequences. |
| Gains | GNS | GAIN_PID | Gains PID for IPIDCONTROLLER (see GAIN_PID structure) |

| | | | |
|---|---|---|---|
| AutoTune | ATUN | BOOL | Starts the Autotune sequence. See To perform an AutoTune sequence.<br>Autotune is available when using the IPID in direct acting systems. The current Autotune algorithm cannot calculate gains in reverse acting system.<br>Autotune is unable to calculate gains on slow-reaction or unstable systems. In such cases, Autotune ends in timeout. |
| ATParameters | ATPA | AT_Param | AutoTune Parameters (see AT_Param structure) |
| ErrorMode | ERR | DINT | Mode used to handle errors. Possible values are:<br>0 no error messages ErrLog file<br>1 prints error messages level 1 in ErrLog file<br>2 prints error messages level 1 and level 2 in ErrLog file |
| Output | OUT | REAL | Output value from controller |
| AbsoluteError | AERR | REAL | Absolute Error (Process – SETPOINT) from controller |
| ATWarning | ATW | DINT | Warning for Autotune sequence. Possible values are:<br>0 no autotune done<br>1 autotuning in progress<br>2 autotuning done<br>-1 ERROR 1 input Auto set to TRUE, no autotune possible<br>-2 ERROR 2 autotune error, ATDynaSet expired |
| OutGains | OGNS | GAIN_PID | Gains calculated after AutoTune sequences (see GAIN_PID structure) |

GAIN_PID structure:

| | | |
|---|---|---|
| DirectActing | BOOL | The type of acting:<br>TRUE direct acting<br>FALSE reverse acting |
| ProportionalGain | REAL | Proportional gain for PID (>= 0.0001) |
| TimeIntegral | REAL | Time integral value for PID (>= 0.0001) |
| TimeDerivative | REAL | Time derivative value for PID (> 0.0)<br>When setting TimeDerivative to 0.0, the IPIDCONTROLLER forces DerivativeGain to 1.0 then works as a PI controller. |
| DerivativeGain | REAL | Derivative gain for PID (> 0.0) |

AT_Param structure:

| | | |
|---|---|---|
| Load | REAL | Load parameter for auto tuning. This is the output value when starting AutoTune. |
| Deviation | REAL | Deviation for auto tuning. This is the standard deviation used to evaluate the noise band needed for AutoTune (noise band = 3*Deviation)[1]. |
| Step | REAL | Step value for AutoTune. Must be greater than noise band and less than ½ Load. |
| ATDynamSet | REAL | Time to wait for stabilization after the step test, in seconds. The AutoTune process stops when ATDynamSet time expires. |
| ATReset | BOOL | The indication of whether the Output value is reset to zero after an AutoTune sequence:<br>TRUE resets Output to zero<br>FALSE leaves Output at Load value |

[1]The application engineer can estimate the value of *Deviation* by observing the value of Process input. For example, in a project involving the control of temperature, if the temperature stabilizes around 22 °C, and a fluctuation of 21.7…22.5 °C is observed, the value of *Deviation* will be (22.5-21.7)/2=0.4.

Description:

The Interacting PID controller (IPIDCONTROLLER) is based on the following function block:

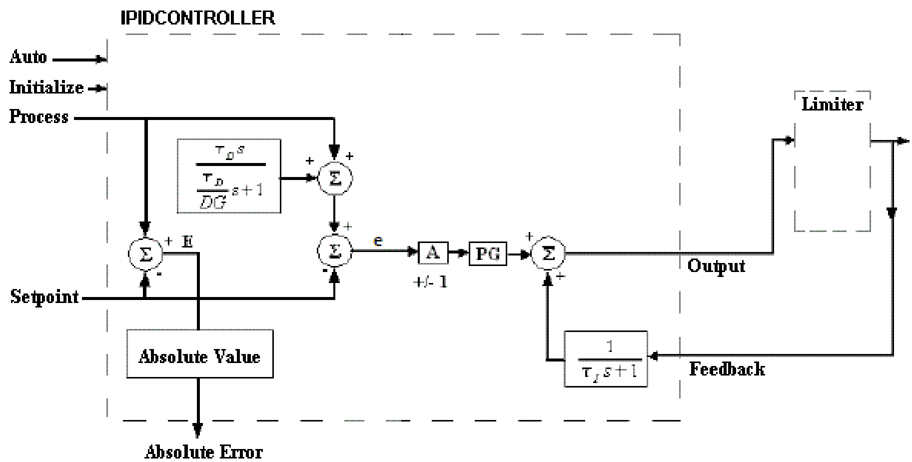
with A: Acting (+/- 1)

PG: Proportional Gain

DG: Derivative Gain

ã$_D$: Time Derivative

ã$_I$: Time Integral




In the HMI, the IPID faceplate is available for use with the IPIDCONTROLLER function block.

When *Auto* is TRUE, the IPIDCONTROLLER enables tracking and runs in normal auto mode. When *Auto* is FALSE, the controller output value equals the feedback value. This forces the IPIDCONTROLLER Output to track the feedback within the IPIDCONTROLLER limits and allows the controller to switch back to auto without bumping the Output.

For *Initialize*, changing from FALSE to TRUE or TRUE to FALSE when AutoTune is FALSE causes the IPIDCONTROLLER to eliminate any proportional gain action during that cycle (i.e Initialize). This can be used to prevent bumping the Output when changes are made to the SETPOINT using a switch function block.

**IPID Autotuning for First and Second Order Systems**

The IPIDCONTROLLER autotune is only functional on first and second order systems.

A first order system is a single independent energy storage element. A first order system can be written in a standard form such as $f(t) = \tau dy/dt + y(t)$, where $\tau$ is the system time constant, f is the forcing function, and y is the system state variable.

The cooling of a fluid tank, mentioned below in the table of first order system examples, can be modeled by the thermal capacitance (C) of the fluid and thermal resistance (R) of the tank walls. In this case, the system time constant is RC, the forcing function is the ambient temperature, and the system state variable is the fluid temperature.

(* Examples of first order systems *)

| First order system | Energy storage element |
|---|---|
| Cooling of a fluid tank | Heat energy |
| Flow of a fluid tank | Potential energy |
| Motor having constant torque driving a disk flywheel | Rotational kinetic energy |
| Electric RC lead network | Capacitive storage energy |

A second order system consists of two independent energy storage elements exchanging stored energy.

(* Examples of second order systems *)

| Second order system | Energy storage element |
|---|---|
| Motor driving a disk flywheel with the motor coupled to the flywheel via a shaft with torsional stiffness | Rotational kinetic energy and torsion spring energy |
| Electric circuit composed of a current source driving a series LR (inductor and resistor) with a shunt C (capacitor) | Inductive and capacitive storage energy |

Motor drive systems and heating systems can be typically modeled by the LR and C electric circuit.

### How Autotune Works

The autotune process begins when *Initialize* is set to FALSE (step 6). Once started, the control output increases by the *Step* value and the process waits until the *Process* value reaches or exceeds the "first peak".

The "first peak" is defined as:

*For Direct Operation: First peak = PV1 - (12*Deviation)*
*For Reverse Operation: First peak = PV1 + (12*Deviation)*
Where PV1 is the process value when *Initialize* is set to FALSE.

Once the process value reaches the first peak, the control output reduces by the *Step* value and waits for the process value to drop to the second peak.

The "second peak" is defined as:

*For Direct Operation: Second peak = PV1 - (3*Deviation)*
*For Reverse Operation: Second peak = PV1 + (3*Deviation)*
Once the process value reaches or falls below the second peak, calculations begin and a set of gains is generated to the *OutGains* parameter.

### IMPORTANT

User program scan time is important. The autotuning method needs to cause the oscillation of the control loop output. To identify the oscillation period, the IPIDCONTROLLER must be called at frequent intervals enabling adequate sampling of the oscillation. The scan time of the user program must be less than half the oscillation period. You must adhere to the Nyquist-Shannon sampling theorem.

In addition, it is important to execute the function block at relatively constant time intervals.

### Actions to Perform before Running AutoTune

Before running an Autotune sequence you need to perform the following:

- Verify the system is constant when there is no control. For example, the *Process* value should remain at room temperature for a temperature control system when there is no control output.

- Configure *SetPoint* to 0
- Set *Auto* to FALSE
- Set the *Gains* parameters to have the following values:

| Gains Parameters | Value |
|---|---|
| DirectActing | According to operation. For example, TRUE for cooling operations and FALSE for heating operations. |
| DerivativeGain | Typically set to 0.1 or 0.0 |
| ProportionalGain | 0.0001 |
| TimeIntegral | 0.0001 |
| TimeDerivative | 0.0 |

- Set the *ATParameters* parameters to have the following values:

| ATParameters Parameters | Recommendation |
|---|---|
| Load | Every *Load* parameter provides a saturated process value over a period of time. Adjust *Load* to the value required for the saturated process value.<br><br>**IMPORTANT**: If a load of 40 results in a process value of 30 ºC over a period of time, to tune your system to 30 ºC, set the load to 40. |
| Deviation | The *Deviation* parameter plays a significant role in the autotune process.<br>You are not required to set the *Deviation* parameter prior to autotuning. However you can set the deviation if you know the required value. |
| Step | The *Step* parameter value ranges between 3**Deviation* and ½ *load*. The *Step* parameter provides an offset for the *Load* during autotuning. The parameter should be set to a value high enough to create significant change in the *Process* value. |

| ATParameters Parameters | Recommendation |
|---|---|
| ATDynamSet | Set the *ATDynamSet* parameter value to a reasonably long time for the autotune process. Since every system is different, specify more time for systems having process values slower in reacting to change. |
| ATReset | Set the *ATReset* parameter to TRUE to reset the output to 0 after the completion of the autotune process.<br>Set the parameter to FALSE to keep the output at *Load* value after the completion of the autotune process. |

**To perform an AutoTune sequence**

1. Set *Initialize* to **TRUE**.

2. Set *AutoTune* to **TRUE**.

3. Wait for *Process* to stabilize or reach a steady state, then note the fluctuation of the *Process* value.

4. Calculate the *Deviation* value with regards to the fluctuation value. For example, if the temperature stabilizes around 22 ºC (72 ºF) with a fluctuation of 21.7 - 22.5 ºC (71 - 72.5 ºF), the *Deviation* value is:

| For ºC | For ºF |
|---|---|
| (22.5-21.7)/2=0.4 | (72.5-71)/2=0.75 |

5. Set the *Deviation* value.

6. Set *Initialize* to **FALSE**.

7. Wait until the *ATWarning* output value is 2, meaning the Autotune process has completed successfully.

8. Get the autotuned value displayed in the *OutGains* output.

**Troubleshooting the Autotune Process**

The sequences of control output enable knowing what is happening behind the autotune process. The following table displays some known sequences of control output, the autotune result, and what actions to perform if autotune fails:

(* For the following *Load* is equal to 50 and *Step* is equal to 20 *)

**Output Sequence 1: 50 -> 70 -> 30**

| Sequence Condition | Autotune Result | Action when Autotune Fails |
|---|---|---|
| Process value reached "first peak" and "second peak" in the required time | Likely successful | N/A |

**Output Sequence 2: 50 -> 70 -> 50**

| Sequence Condition | Autotune Result | Action when Autotune Fails |
|---|---|---|
| Process value unable to reach "first peak" | Likely unsuccessful | Reduce *Deviation* or increase *Step* value |

**Output Sequence 3: 50 -> 70 -> 30 -> 50**

| Sequence Condition | Autotune Result | Action when Autotune Fails |
|---|---|---|
| Process value unable to reach "second peak" | Likely unsuccessful | Increase *Deviation* or increase *Step* value |

**Output Sequence 4: 50 -> 70**

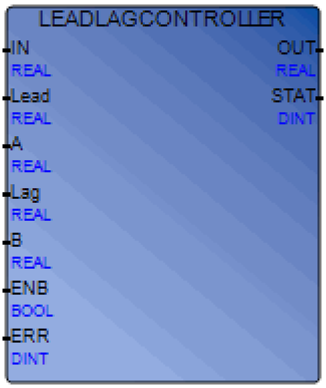
| Sequence Condition | Autotune Result | Action when Autotune Fails |
|---|---|---|
| Process value unable to reach "first peak" in the required time | Likely unsuccessful | Increase *ATDynamSet* value |

To finalize the tuning, some fine tuning may be needed depending on the processes and needs.

## Example

```
(* ST equivalence: IPIDCONTROLLER1 is an instance of IPIDCONTROLLER
block *)

IPIDCONTROLLER1(Proc,
        SP,
        FBK,
        Auto,
        Init,
        G_In,
        A_Tune,
        A_TunePar,
        Err );
Out_process := IPIDCONTROLLER1.Output;
A_Tune_Warn := IPIDCONTROLLER1.ATWarning;
Gain_Out := IPIDCONTROLLER1.OutGains;
```

# LEADLAGCONTROLLER




Arguments:

| | | | |
|---|---|---|---|
| Input | IN | REAL | Input signal |
| TimeLead | Lead | REAL | Time constant for lead controller, in seconds |
| A | A | REAL | Gain for lead controller (a > 1 and a x b = 1) |
| TimeLag | Lag | REAL | Time constant for lag controller, in seconds |
| B | B | REAL | Gain for lag controller (b < 1 and a x b = 1) |
| Enable | ENB | BOOL | Enables the LEADLAGCONTROLLER. If set to FALSE, Output = 0.0 |
| ErrorMode | ERR | DINT | Mode used to handle the various types of errors:<br>a < 1.0<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets a to 1.0001<br>b > 1.0<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets b to 0.9999<br>TimeLag < 0<br>1 prints message in ErrorLog, stops resource code execution, and sets Status output to 1<br>0 sets Status output to 1 |

| | | | |
|---|---|---|---|
| Output | OUT | REAL | LEADLAGCONTROLLER output |
| Status | STAT | DINT | Status for LEADLAGCONTROLLER:<br>0 OK<br>1 TimeLag < 0.0<br>2 Divided by zero<br>3 Square root error (negative argument) |

Description:

The LEADLAGCONTROLLER is based on the transfer function from Automatic control systems by Benjamin C.Kuo:

$$C(s) = \frac{(1 + aT_{lead}s)(1 + bT_{lag}s)}{(1 + T_{lead}s)(1 + T_{lag}s)}$$

$$a > 1$$

$$b < 1$$

$$a \times b = 1$$

The lead controller gain a must be greater than 1.0, the lag controller gain b must be less than 1.0, and a multiplied by b must equal 1.0. If a x b does not equal 1.0, the controller will use b = 1/a.

With TimeLead set to zero, the controller will act as a Lag controller.

For entry errors, ErrorMode gives you the possibility to stop the resource.

For error of type division by zero or square root with negative argument, the controller sets the Status output to 2 or 3 respectively. The Output for those cases will be 0.0.

**Discretization method: Zero-Order Hold**

The Zero-Order Hold method is used by the function block to provide a match between the continuous and discrete time systems in the time domain discretization.

The following steps illustrate the summary of the calculus:

**1.** Conversion: continuous time to discrete time:

$$C(z) = (1 - z^{-1})Z\left[\frac{C(S)}{S}\right] = (1 - z^{-1})Z\left[\frac{(1 + aT_{lead}S)(1 + bT_{lag}S)}{S(1 + T_{lead}S)(1 + T_{lead}S)}\right]$$

Where
z = the transform operator
s = the Laplace operator

**2.** Partial fraction decomposition:

The equation from step 1 could be written after development of denominator as:

$$C(z) = (1 - z^{-1})Z\left[\frac{(1+aT_{lead}S)(1+bT_{lag}S)}{S(T_{lead}T_{lag}S^2 + (T_{lead}+T_{lag})S+1)}\right] = (1 - z^{-1})Z\left[\frac{(1+aT_{lead}S)(1+bT_{lag}S)}{S(S-R)(S-Q)T_{lead}T_{lag}}\right]$$

Where R and Q are the solutions of the quadratic equation:

$$T_{lead}T_{lag}S^2 + (T_{lead} + T_{lag})S + 1$$

Then C(z) could be written as:

$$C(z) = (1 - z^{-1})Z\left[\frac{A}{S} \times \frac{1}{T_{lead}T_{lag}} + \frac{B}{S-R} \times \frac{1}{T_{lead}T_{lag}} + \frac{C}{S-Q} \times \frac{1}{T_{lead}T_{lag}}\right]$$

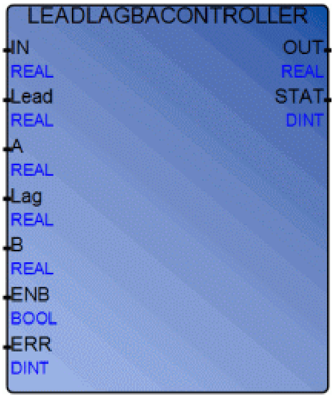
**3.** Factors A, B and C:

$$A = \left[\frac{(1 + aT_{lead}S)(1 + bT_{lag}S)}{S(S - R)(S - Q)T_{lead}T_{lag}}\right]_{S=0} = \frac{1}{QR} \times \frac{1}{T_{lead}T_{lag}}$$

$$B = \left[\frac{(1 + aT_{lead}S)(1 + bT_{lag}S)}{S(S - R)(S - Q)T_{lead}T_{lag}}\right]_{S=R} = \frac{(1 + aT_{lead}R)(1 + bT_{lag}R)}{R(R - Q)} \times \frac{1}{T_{lead}T_{lag}}$$

$$C = \left[\frac{(1 + aT_{lead}S)(1 + bT_{lag}S)}{S(S - R)(S - Q)T_{lead}T_{lag}}\right]_{S=Q} = \frac{(1 + aT_{lead}Q)(1 + bT_{lag}Q)}{Q(Q - R)} \times \frac{1}{T_{lead}T_{lag}}$$

# LEADLAGBACONTROLLER




Arguments:

| | | | |
|---|---|---|---|
| Input | IN | REAL | Input signal |
| TimeLead | Lead | REAL | Time constant for lead controller, in seconds |
| A | A | REAL | Gain for lead controller (a > 1 and a x b = 1) |
| TimeLag | Lag | REAL | Time constant for lag controller, in seconds |
| B | B | REAL | Gain for lag controller (b < 1 and a x b = 1) |
| Enable | ENB | BOOL | Enables the LEADLAGBACONTROLLER. If set to FALSE, Output = 0.0 |

| | | | |
|---|---|---|---|
| ErrorMode | ERR | DINT | Mode used to handle the various types of errors:<br>a < 1.0<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets *A* to 1.0001<br>b > 1.0<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets *B* to 0.9999<br>TimeLag < 0<br>1 prints message in ErrorLog, stops resource code execution, and sets Status output to 1<br>0 sets Status output to 1 |
| Output | OUT | REAL | LEADLAGBACONTROLLER output |
| Status | STAT | DINT | Status for LEADLAGBACONTROLLER:<br>0 OK<br>1 TimeLag < 0.0<br>2 Divided by zero<br>3 Square root error (negative argument) |

Description:

The LEADLAGBACONTROLLER is based on the transfer function from Automatic control systems by Benjamin C.Kuo:

$$C(s) = \frac{(1 + aT_{lead}s)(1 + bT_{lag}s)}{(1 + T_{lead}s)(1 + T_{lag}s)}$$

$$a > 1$$

$$b < 1$$

$$a \times b = 1$$

Where
s = Laplace transform complex variable
a = Lead compensator gain
b = Lag compensator gain
Tld = Lead compensator time constant, in seconds
Tlg = Lag compensator time constant, in seconds
C(s) = Output to input transfer function

The lead controller gain a must be greater than 1.0, the lag controller gain b must be less than 1.0, and a multiplied by b must equal 1.0. If a x b does not equal 1.0, the controller will use b = 1/a.

With TimeLead set to zero, the controller will act as a Lag controller.

For entry errors, ErrorMode gives you the possibility to stop the resource.

For error of type division by zero or square root with negative argument, the controller sets the Status output to 2 or 3 respectively. The Output for those cases will be 0.0.

**Discretization method: Bilinear Approximation (also called Tustin Approximation)**

The Bilinear Approximation method is used by the function block to provide a match between the continuous and discrete time systems in the time domain discretization.

To convert from the analog domain to the digital domain we apply a bilinear transform:

$$s = \frac{2}{T} \frac{(z-1)}{(z+1)}$$

where
s = Laplace transform complex variable
z = Z transform complex variable
T = Sampling period, in seconds

Substituting this "s" with C(s) and simplification results in Z transform of the Lead-Lag compensator:

$$C(z) = \frac{(T\,z + T + 2\,b\,Tlg\,z - 2\,b\,Tlg)\,(T\,z + T + 2\,a\,Tld\,z - 2\,a\,Tld}{(T\,z + T + 2\,Tlg\,z - 2\;Tlg)\,(T\,z + T + 2\;Tld\,z - 2\;Tld)}$$

Obviously we want the time domain filter to have the form:

y(n) = K1x(n) + K2x(n-1) + K3x(n-2) + …

Without going in calculation details the end results from Z transform to Time domain we get:

y(n) = 1/K7 [x(n) + 2 x(n–1) + K5 x(n) + x(n-2) – K5 x(n-2) + K6 x(n) + K4 x(n) – 2 K4 x(n-1) – K6 x(n-2) + K4 x(n-2) – 2 y(n-1) + 2 K3 y(n-1) – y(n-2) + K1 y(n-2) + K2 y(n-2) – K3 y(n-2)]

Where
K1 = (2 Tld)/T
K2 = (2 Tlg)/T
K3 = K1 K2
K4 = a b K3
K5 = a K1
K6 = b K2
K7 = 1 + K1 + K2 + K3
T = VM cycle time
y = filter output
x = filter input

# LIMITER


Arguments:

| | | | |
|---|---|---|---|
| Input | IN | REAL | Real value on which to limit the value |
| HighLimit | HLIM | REAL | High limit value |
| LowLimit | LLIM | REAL | Low limit value |
| ErrorMode | ERR | DINT | Mode used to handle errors of type HighLimit ≤ LowLimit. Possible values are:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets Output = Input if HighLimit ≤ LowLimit |
| Output | OUT | REAL | Tracks Input up to HighLimit and down to LowLimit |
| HighStatus | HSTS | BOOL | TRUE when Input > HighLimit |
| LowStatus | LSTS | BOOL | TRUE when Input < LowLimit |

Description:

Tracks Input value and limits it to a value between LowLimit and HighLimit

### Example

```
(* ST equivalence: LIMITER1 is an instance of LIMITER block *)

LIMITER1( InputA, 250.0, 25.0, 0 );

OutputB := LIMITER1.Output ;

High_Limit := LIMITER1.HighStatus ;

Low_Limit := LIMITER1.LowStatus ;
```

# PID_AL


Arguments:

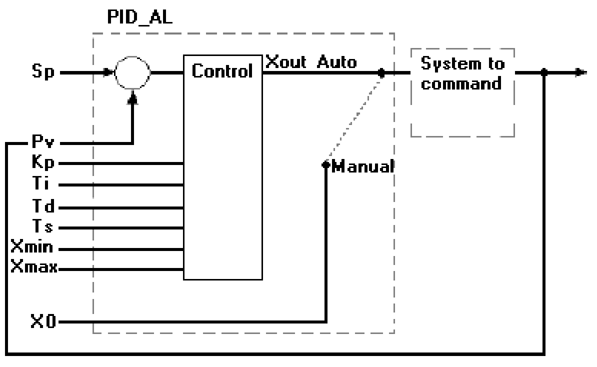
| | | |
|---|---|---|
| AUTO | BOOL | The operation mode of the PID controller:<br>TRUE controller runs in automatic mode<br>FALSE controller runs in manual mode. At initialisation, set the operation mode to FALSE. |
| Pv | REAL | Process output value |
| Sp | REAL | Setpoint value, i.e., value required at the output |
| X0 | REAL | Adjustment value. When running in manual mode, in the case of an open loop, is the non-regulated value entering the system where the output value of the PID controller is equal to X0. |
| Kp | REAL | Proportionality constant |
| Ti | REAL | Integral time constant |
| Td | REAL | Derivative time constant |

| | | |
|---|---|---|
| Ts | TIME | Sampling period |
| Xmin | REAL | Minimum possible value |
| Xmax | REAL | Maximum possible value |
| Xout | REAL | Command. In the case of a closed loop with regulation, is the regulated value entering the system. |

Description:

The PID_AL function block is a proportional–integral–derivative controller (PID controller) using a generic control loop feedback mechanism (controller). This function block calculates an error value as the difference between a measured process variable and a desired setpoint. The block attempts to minimize the error by adjusting the process control inputs while implementing a bumpless compensation algorithm allowing the modification of PID coefficients at run-time.

While in Auto mode, the PID_AL function block is a PID process regulator using the feedback concept where an output is regulated according to the difference between its actual value and the expected value. While in Manual mode, the PID_AL function block is a non regulated system enabling the performance of tests and adjustments.

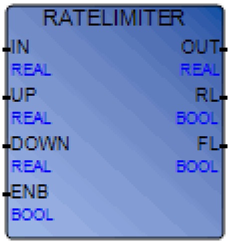



The PID_AL function block is implemented using the following PID model:

$$MV(t) = K_p \cdot \left( e(t) + \frac{1}{T_i} \cdot \int_0^t e(\tau) d\tau + T_d \cdot \frac{d}{dt} e(t) \right)$$

where
Ti  is the integral time
Td  is the derivative time

# RATELIMITER




Arguments:

| | | | |
|---|---|---|---|
| Input | IN | REAL | Real value on which to limit the rate variation |
| UpRate | UP | REAL | The upper limit rate, in units/minute |
| DownRate | DOWN | REAL | The lower limit rate, in units/minute |
| Enable | ENB | BOOL | TRUE enables rate limitation action |
| Output | OUT | REAL | When Enable is FALSE, Output equals Input. When Enable is TRUE, Output rate is limited by UpRate or DownRate. |
| RaisingLimit | RL | BOOL | TRUE when block limits a rising Input |
| FallingLimit | FL | BOOL | TRUE when block limits a falling Input |

Description:

Limits the rate of change for an input signal:

Enable = TRUE:

When the Input signal increases, the RisingLimit is TRUE and Output changes at the UpRate rate. When the Input signal decreases, the FallingLimit is TRUE and Output changes at the DownRate rate. When the Input signal changes at a rate between UpRate and DownRate, Output tracks Input.

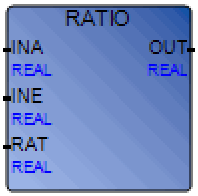
Enable = FALSE:

The Output tracks the Input.

### Example

```
(* ST equivalence: RATELIMITER1 is an instance of the RATELIMITER
block; *)


RATELIMITER1( InputA, 5.0 , 1.0 , Enable_Bit) ;

OutputB := RATELIMITER1.Output ;

Limiting_Up_Rate := RATELIMITER1.RisingLimit ;

Limiting_Down_Rate := RATELIMITER1.FallingLimit ;
```

# RATIO




Arguments:

| | | | |
|---|---|---|---|
| InputA | INA | REAL | Input signal A |
| InputE | INE | REAL | Input signal E |
| Ratio | RAT | REAL | RATIO value |
| Output | OUT | REAL | Output value. Output = (RATIO) x InputA x InputE |

Description:

Provides a means of setting a ratio in an external setpoint control. For example, controlling a captive flow while maintaining the ratio between a wild flow and the captive flow at the desired value. Input signal A, input signal E (external ratio), and the operator set RATIO values are multiplied and become the function block Output. The RATIOCALIBRATION function block calibrates RATIO using a tracked input signal.

## Example

```
(* ST equivalence: RATIO1 is an instance of RATIO block and
RATIOCALIBRATION1 is an instance of RATIOCALIBRATION block *)


RATIO1(Signal_InA, Signal_InE, RATIOCALIBRATION1.Ratio);

Out_Value := RATIO1.Output ;
```

# RATIOCALIBRATION


Arguments:

| | | | |
|---|---|---|---|
| InputA | INA | REAL | Input signal A |
| InputE | INE | REAL | Input signal E |
| Initial | INIT | REAL | RATIO value at first scan |
| HighLimit | HLIM | REAL | High Limit for RATIO |
| LowLimit | LLIM | REAL | Low Limit for RATIO |
| TrackVariable | TV | REAL | Input Signal to track |
| TrackCommand | TC | BOOL | Command to initiate TrackVariable tracking |
| Ratio | RAT | REAL | RATIO value |
| TrackOutput | TO | REAL | When TrackCommand = FALSE , TrackOutput = 0.0<br>When TrackCommand = TRUE, TrackOutput = (TrackVariable) / (InputA * RATIO) |

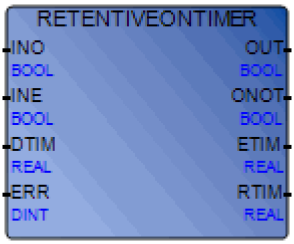
Description:

Calibrates RATIO using TrackVariable. When TrackCommand is FALSE, RATIO equals last RATIO value and TrackOutput is 0.0. When TrackCommand is TRUE, RATIO equals (TrackVariable) / (InputA * InputE); TrackOutput = (TrackVariable) / (InputA * RATIO) also RATIO will be limited by HighLimit and LowLimit. The RATIO function block provides a means of setting a ratio in an external setpoint application.

### Example

```
(* ST equivalence: RATIO1 is an instance of RATIO block and
RATIOCALIBRATION1 is an instance of RATIOCALIBRATION block *)


RATIOCALIBRATION1(Signal_InA,

       Signal_InE,

       0.2,

       300.0,

       10.0,

       Flow_Water,

       TK);

RATIO1(Signal_InA, Signal_InE, RATIOCALIBRATION1.RATIO);

Out_Value := RATIO1.Output ;
```

# RETENTIVEONTIMER




Arguments:

| | | | |
|---|---|---|---|
| InputOn | INO | BOOL | Input to start timer |
| InputEnable | INE | BOOL | Input to enable timer |
| DelayTime | DTIM | REAL | Delay time in seconds |
| ErrorMode | ERR | DINT | Mode used to handle errors of type: DelayTime < 0.0:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets Output to TRUE, OutputNot to FALSE, ElapseTime to 0.0, and RemainingTime = 0.0 |
| Output | OUT | BOOL | Signal = TRUE when RemainingTime >= 0.0 |
| OutputNot | ONOT | BOOL | Signal = FALSE when RemainingTime >= 0.0 |
| ElapseTime | ETIM | REAL | Time elapsed since the timer started |
| RemainingTime | RTIM | REAL | Time remaining before Output changes to TRUE. |

Description:

Performs an on-delay timing function with output states determined by InputOn and InputEnable. When InputEnable is FALSE, Output and OutputNot are FALSE, RemainingTime equals DelayTime. When InputEnable is TRUE, Output and OutputNot are determined by InputOn and RemainingTime.

When InputOn is TRUE, ElapseTime starts to increase and RemainingTime starts to decrease. Output changes to TRUE after RemainingTime <= 0.0. If InputOn changes to FALSE, RemainingTime and ElapseTime stop at their current value and continue when InputOn returns to TRUE. ElapseTime returns to 0.0 when InputEnable is FALSE. OutputNot is TRUE if InputEnable is TRUE and Output is FALSE.

### Example

```
(* ST equivalence: RETENTIVEONTIMER1 is an instance of RETENTIVEONTIMER
block *)


RETENTIVEONTIMER1(On_Tmr, En_Tmr, 300.0, 0);

Timer_Done := RETENTIVEONTIMER1.Output ;

Timer_Not_Done := RETENTIVEONTIMER1.OutputNot ;

Time_To_Count := RETENTIVEONTIMER1.RemainingTime ;

Time_Counted := RETENTIVEONTIMER1.ElapseTime ;
```

# SCALER


Arguments:

| | | | |
|---|---|---|---|
| Input | IN | REAL | Input signal |
| InputMin | IMIN | REAL | Minimum value of Input |
| InputMax | IMAX | REAL | Maximum value of Input |
| OutputMin | OMIN | REAL | Minimum value of Output |
| OutputMax | OMAX | REAL | Maximum value of Output |
| Output | OUT | REAL | Output value |

Description:

Scales the input value according to the output range:

$$\frac{(Input - InputMin)}{(InputMax - InputMin)} \times (OutputMax - OutptuMin) + OutputMin$$

## Example

```
(* ST equivalence: SCALER1 is an instance of SCALER block *)


SCALER1(Signal_In, 4.0, 20.0 , 0.0 , 150.0 ) ;

Out_Temp := SCALER1.Output ;
```

# SETPOINT


Arguments:

| | | | |
|---|---|---|---|
| TrackVariable | TV | REAL | Variable to track |
| TargetSetpoint | TS | REAL | Value to attain for setpoint |
| RampRate | RR | REAL | Ramp rate value, per second |
| RampTime | RT | REAL | Ramp time value, in seconds |
| Command | CMD | DINT | Command for SETPOINT. Possible values are:<br>0 Output equals last output<br>1 Output equals TrackVariable<br>2 Output changes from current value to TargetSetpoint at RampRate rate<br>3 Output changes from current value to TargetSetpoint at (TargetSetpoint – Initial value) /RampTime rate |
| PulseUp | PU | BOOL | Increment output for PulseRate value upon detection of upward pulses |
| PulseDown | PD | BOOL | Decrement output for PulseRate value upon detection of downward pulses |

| | | | |
|---|---|---|---|
| PulseRate | PR | DINT | Pulse rate value, per second |
| ErrorMode | ERR | DINT | Mode used to handle errors of type negative RampRate and negative RampTime. Possible values are:<br>1 prints message in ErrorLog and stops resource code execution<br>0 sets output to zero |
| Output | OUT | REAL | Current setpoint value |

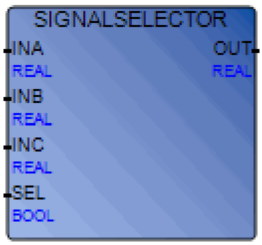
Description:

Multi-action setpoint command having six different settings and adjustment of SETPOINT for controller. On first scan, output equals TrackVariable. Using a different Command, the setpoint can be adjusted to last Output, TrackVariable, or TargetSetpoint. At any time, the two pulse entries can be used to increment or decrement the output (for example, via an HMI or a pulse switch).

## Example

```
(* ST equivalence: SETPOINT1 is an instance of SETPOINT block *)


SETPOINT1(Signal_In, SETPOINTValue, 10.0, 25.0, UserCommand, RemoteUp,
RemoteDown, 5, 0);


ProcessSETPOINT := SETPOINT1.Output ;
```

# SIGNALSELECTOR




Arguments:

| | | | |
|---|---|---|---|
| InputA | INA | REAL | Input signal A |
| InputB | INB | REAL | Input signal B |
| InputC | INC | REAL | Input signal C |
| Selector | SEL | BOOL | (Selector) Indication of whether the highest or lowest signal value is selected. Possible values are:<br>TRUE selects highest signal value<br>FALSE selects lowest signal value |
| Output | OUT | REAL | (Output) Selected signal |

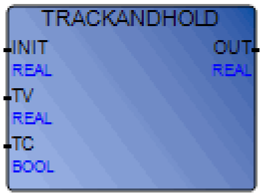
Description:

Selects either the highest or lowest signal value from three input signals. When Selector is FALSE, the lowest signal value between input A, input B, and input C is sent to Output. When Selector is TRUE, the highest signal value between input A, input B, and input C is sent to Output.

### Example

```
(* ST equivalence: SIGNALSELECTOR1 is an instance of SIGNALSELECTOR
block *)


SIGNALSELECTOR1( InA, InB, InC, Sel ) ;

Selected_Signal := SIGNALSELECTOR1.Output ;
```

# TRACKANDHOLD




Arguments:

| | | | |
|---|---|---|---|
| Initial | INIT | REAL | Initial value to transfer to Output |
| TrackVariable | TV | REAL | (TrackVariable) Input signal to track |
| TrackCommand | TC | BOOL | (Track command) When TRUE, Output tracks the TrackVariable. When FALSE, Output stays the same as the last Output value. |
| Output | OUT | REAL | Output signal |

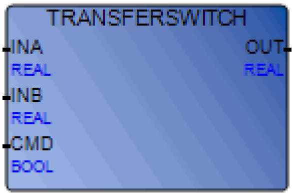
Description:

Holds an initial value transferred to output on first scan. Tracks the TrackVariable when TrackCommand is TRUE and holds the last output value when FALSE.

### Example

```
(* ST equivalence: TRACKANDHOLD1 is an instance of TRACKANDHOLD block
*)


TRACKANDHOLD1(25.0, Signal_To_Track, Command);

Out_Value := TRACKANDHOLD1.Output ;
```

# TRANSFERSWITCH




Arguments:

| | | | |
|---|---|---|---|
| InputA | INA | REAL | Input signal A |
| InputB | INB | REAL | Input signal B |
| Command | CMD | BOOL | (Command) Indication of which signal to select:<br>FALSE selects InputA<br>TRUE selects InputB |
| Output | OUT | REAL | Output signal |

Description:

Selects a signal between two inputs with the switch command.

### Example

```
(* ST equivalence: TRANSFERSWITCH1 is an instance of TRANSFERSWITCH
block *)


TRANSFERSWITCH1( Signal_A, Signal_B, Switch_Command);

Out_value := TRANSFERSWITCH1.Output;
```

# Matrix2 Operations

A matrix is a two-dimensional array variable made up of rows and columns. It is mainly used to perform complex calculations involving the data of the running application. The Matrix2 function block performs all of these operations. However, each operation has a specific identifier and requires different inputs. The outputs other than those specified for the function do not contain valid information. All Matrix2 operations are executed on a change in value greater than 0.

The intersection of a row and a column is called a cell; cells hold the matrix values. The number of the first row of a matrix is 0; the number of its first column is also 0.

The Workbench offers the Matrix2 built-in function block performing multiple operations for filling and manipulating matrices. Each of the available operations is indicated by a number ranging from 0 to 10. Furthermore, the Matrix2 block performs an operation on a change in value (increasing or decreasing). Therefore, to repeat a specific operation, the block operation number must increase or decrease before resuming a previous operation number.

The available Matrix2 operations are the following:

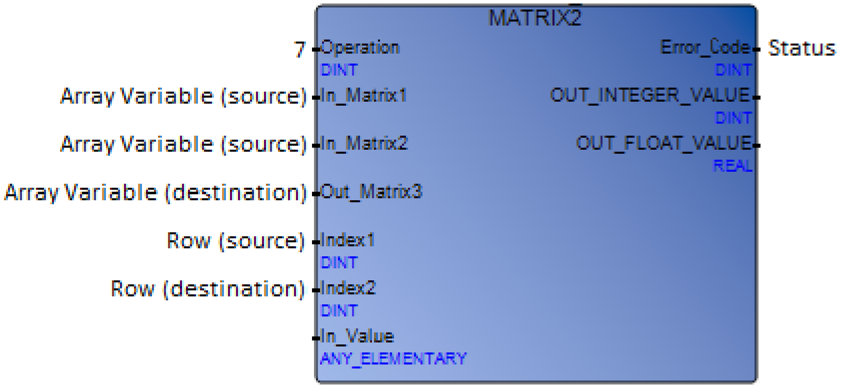
| | | |
|---|---|---|
| NULLIFY_OPERATION | 0 | Nullifies an operation to enable repeating one of the other possible Matrix2 operations. |
| COPY_ROW_MATRIX | 7 | Copies a row from a matrix into a row of the same size in another matrix. The cell value type must be the same in both matrices (Index1 and Index2 are used). |
| COPY_COL_MATRIX | 8 | Copies a column from a matrix into a row of the same size in another matrix. The cell value type must be the same in both matrices (Index1 and Index2 are used). |
| TRANSPOSE_MATRIX | 1 | Swaps the rows and columns of an existing matrix into another existing matrix called a transpose |
| INVERT_MATRIX | 2 | Computes the inverse of a float (REAL) matrix |
| ADD_MATRIX | 3 | Adds up two existing matrices and places the result in a third matrix |

| | | |
|---|---|---|
| SUBTRACT_MATRIX | 4 | Subtracts an existing matrix from another existing matrix and places the result in a third matrix |
| MULTIPLY_MATRIX | 5 | Multiplies two existing matrices and places the result in a third matrix |
| SCALAR_MATRIX | 6 | Multiplies each cell value of a float (REAL) matrix by a float (REAL) value |
| PRINT_MATRIX | 9 | Prints the contents of all matrices on the console |
| GET_VERSION | 10 | Returns the version number of the function block |

The examples for the individual Matrix2 operations are based on the variables from the following definitions:

| | Name | Logical Value | Physical Value | Lock | Data Type | Dimension |
|---|---|---|---|---|---|---|
| + | mat1 | ... | ... | | DINT | [0..3,0..3] |
| + | mat2 | ... | ... | | DINT | [0..3,0..3] |
| + | mat3 | ... | ... | | DINT | [0..3,0..3] |
| | FirstCycle | | | | BOOL | |
| + | matR1 | ... | ... | | REAL | [0..3,0..3] |
| + | matR2 | ... | ... | | REAL | [0..3,0..3] |
| + | matR3 | ... | ... | | REAL | [0..3,0..3] |
| | cmd1_transpose | 1 | | | DINT | |
| | cmd2_invert | 2 | | | DINT | |
| | cmd3_add | 3 | | | DINT | |
| | cmd4_sub | 4 | | | DINT | |
| | cmd5_mult | 5 | | | DINT | |
| | cmd6_scalar | 6 | | | DINT | |
| | cmd7_copy_row | 7 | | | DINT | |
| | cmd8_copy_col | 8 | | | DINT | |
| | cmd9_print | 9 | | | DINT | |
| | cmd10_get_ver | 10 | | | DINT | |
| + | fbm1 | ... | ... | | MATRIX2 | |

## COPY_ROW_MATRIX




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 7. |
| In_Matrix1 | ANY_ELEMENTARY | Array variable for the source matrix |
| In_Matrix2 | ANY_ELEMENTARY | Array variable for the destination matrix. This must not be the source matrix. |
| Index1 | DINT | Number of the row, in the source matrix, that is copied. The possible values range from 0 to N-1, N being the total number of rows. |
| Index2 | DINT | Number of the row, in the destination matrix, that receives a row. The possible values range from 0 to N-1, N being the total number of rows. |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>3 = Type mismatch<br>5 = Column mismatch<br>7 = Index out of range |

Description:

Copies a row from a matrix into a row of the same size in another matrix. The cell value type must be the same in both matrices.

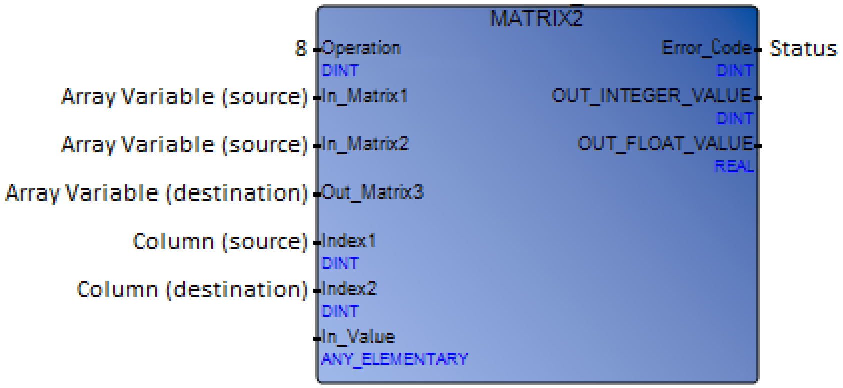
### Example

To copy the contents of a row from a matrix and place it into a row of another matrix. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Copy_row_7)

THEN

  (* Copies a row from a matrix into a row of the same size in another
matrix or into the same matrix.  *)

  Copy_row_7 := FALSE;

  IF (Use_Int_Matrix) THEN op1 := cmd7_copy_row; ELSE op2 :=
cmd7_copy_row; END_IF;

END_IF;


(* FB for DINT operations *)

fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);

err1  := fbm1.Error_code;

out1i := fbm1.OUT_INTEGER_VALUE;

out1r := fbm1.OUT_FLOAT_VALUE;

op1   := 0;


(* FB for REAL operations *)

fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);

err2  := fbm2.Error_code;

out2i := fbm2.OUT_INTEGER_VALUE;

out2r := fbm2.OUT_FLOAT_VALUE;
```

```
op2    := 0;
```

## COPY_COL_MATRIX




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 8. |
| In_Matrix1 | ANY_ELEMENTARY | Array variable for the source matrix |
| In_Matrix2 | ANY_ELEMENTARY | Array variable for the destination matrix. This must not be the source matrix. |
| Index1 | DINT | Number of the column, in the source matrix, that is copied. The possible values range from 0 to N-1, N being the total number of columns. |

| | | |
|---|---|---|
| Index2 | DINT | Number of the column, in the destination matrix, that receives a column. The possible values range from 0 to N-1, N being the total number of columns. |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>2 = Invalid type<br>3 = Type mismatch<br>5 = Column mismatch<br>7 = Index out of range |

Description:

Copies a column from a matrix into a column of the same size in another matrix. The cell value type must be the same in both matrices.

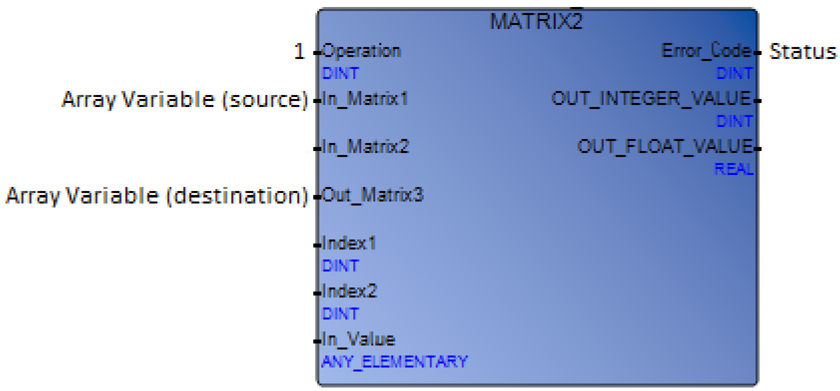
### Example

To copy the contents of a column from a matrix and place it into a column of another matrix. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Copy_col_8)

THEN

  (* Copies a column from a matrix into a row of the same size in another
matrix or into the same matrix.  *)

  Copy_col_8 := FALSE;

  IF (Use_Int_Matrix) THEN op1 := cmd8_copy_col; ELSE op2 :=
cmd8_copy_col; END_IF;

END_IF;


(* FB for DINT operations *)

fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);

err1  := fbm1.Error_code;
```

```
out1i := fbm1.OUT_INTEGER_VALUE;
out1r := fbm1.OUT_FLOAT_VALUE;
op1   := 0;

(* FB for REAL operations *)
fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);
err2  := fbm2.Error_code;
out2i := fbm2.OUT_INTEGER_VALUE;
out2r := fbm2.OUT_FLOAT_VALUE;
op2   := 0;
```

## TRANSPOSE_MATRIX




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 1. |
| In_Matrix1 | ANY_ELEMENTARY | Array variable for the matrix to be transposed (source) |
| Out_Matrix3 | ANY_ELEMENTARY | Array variable for the matrix to receive the resulting transposed matrix. This must not be the source matrix. |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

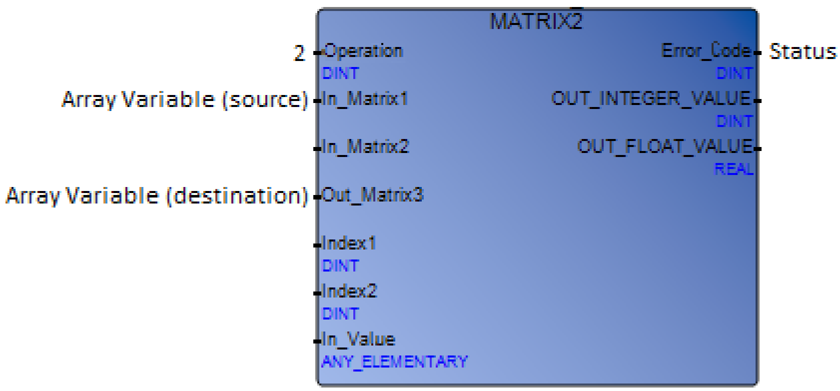
Swaps the rows and columns of an existing matrix into another matrix called a transpose. For instance, the transpose of a matrix having three rows and five columns has five rows and three columns. The transpose matrix is created with the required row-column structure and data type. You place the transposed matrix into an existing matrix.

### Example

To swap the rows and columns of an existing matrix into another existing matrix called a transpose. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Transpose_1)
THEN
   (* Transpose the matrix *)
  Transpose_1 := FALSE;
  IF (Use_Int_Matrix) THEN op1 := cmd1_transpose; ELSE op2 :=
cmd1_transpose; END_IF;
END_IF;


(* FB for DINT operations *)
fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);
err1  := fbm1.Error_code;
out1i := fbm1.OUT_INTEGER_VALUE;
out1r := fbm1.OUT_FLOAT_VALUE;
op1   := 0;


(* FB for REAL operations *)
fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);
err2  := fbm2.Error_code;
out2i := fbm2.OUT_INTEGER_VALUE;
out2r := fbm2.OUT_FLOAT_VALUE;
op2   := 0;
```

# INVERT_MATRIX




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 2. |
| In_Matrix1 | ANY_ELEMENTARY | Array variable for the matrix to be inverted (source). The cell value type must be float (REAL). |
| Out_Matrix3 | ANY_ELEMENTARY | Array variable for the matrix to receive the resulting inverted matrix. This must not be the source matrix. The cell value type must be float (REAL). |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>2 = Invalid type<br>3 = Type mismatch<br>6 = Dimension mismatch<br>7 = Index out of range<br>8 = Not square<br>9 = Mathematical error |

Description:

Computes the inverse of a matrix. The source matrix must be square (i.e., have the same number of rows and columns) and its cell value type must be float (REAL). The inverse matrix will be created with the required row-column structure and data type.

You place the transposed matrix into an existing matrix.

**Note:** Not all matrices are invertible. Invertible matrices are those whose determinant is not equal to 0.

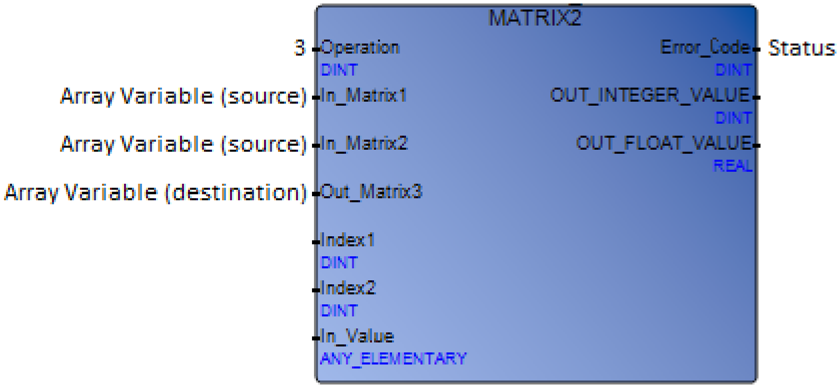
### Example

To invert a source matrix and place the result in a destination matrix. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Invert_2)
THEN
  (* Invert the matrix - This operation is only for floats *)
  Invert_2 := FALSE;
  op2 := cmd2_invert;
END_IF;

(* FB for DINT operations *)
fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);
err1  := fbm1.Error_code;
out1i := fbm1.OUT_INTEGER_VALUE;
out1r := fbm1.OUT_FLOAT_VALUE;
op1   := 0;

(* FB for REAL operations *)
fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);
err2  := fbm2.Error_code;
```

```
out2i := fbm2.OUT_INTEGER_VALUE;
out2r := fbm2.OUT_FLOAT_VALUE;
op2   := 0;
```

## ADD_MATRIX




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 3. |
| In_Matrix1 | ANY_ELEMENTARY | Array variable for the first matrix in the addition |
| In_Matrix2 | ANY_ELEMENTARY | Array variable for the other matrix in the addition |

| | | |
|---|---|---|
| Out_Matrix3 | ANY_ELEMENTARY | Array variable for the existing matrix that will receive the operation result. This must not be one of the matrices indicated in In_Matrix1 or In_Matrix2. |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>2 = Invalid type<br>3 = Type mismatch<br>4 = Row mismatch<br>5 = Column mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

Adds up two existing matrices then places the result in a third matrix. The summation is performed cell by cell, with the result occupying the same cell position in the third matrix. The matrices that are added up must have the same dimensions and cell value type.

You place the result into an existing matrix.

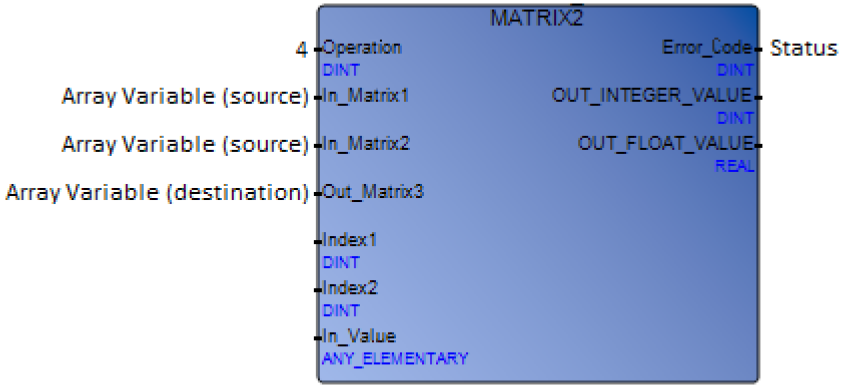
## Example

To add two matrices then place the result in a third matrix. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Add_3)

THEN

  (* Add the matrices *)

  Add_3 := FALSE;

  IF (Use_Int_Matrix) THEN op1 := cmd3_add; ELSE op2 := cmd3_add;
END_IF;

END_IF;


(* FB for DINT operations *)
```

```
fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);
err1  := fbm1.Error_code;
out1i := fbm1.OUT_INTEGER_VALUE;
out1r := fbm1.OUT_FLOAT_VALUE;
op1   := 0;

(* FB for REAL operations *)
fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);
err2  := fbm2.Error_code;
out2i := fbm2.OUT_INTEGER_VALUE;
out2r := fbm2.OUT_FLOAT_VALUE;
op2   := 0;
```

## SUBTRACT_MATRIX




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 4. |
| In_Matrix1 | ANY_ELEMENTARY | Array variable for the first matrix in the subtraction |
| In_Matrix2 | ANY_ELEMENTARY | Array variable for the other matrix in the subtraction |

| | | |
|---|---|---|
| Out_Matrix3 | ANY_ELEMENTARY | Array variable for the existing matrix that will receive the operation result. This must not be one of the matrices indicated in In_Matrix1 or In_Matrix2. |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>2 = Invalid type<br>3 = Type mismatch<br>4 = Row mismatch<br>5 = Column mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

Subtracts an existing matrix from another existing matrix then places the result in a third matrix. The difference is performed cell by cell, with the result occupying the same cell position in the third matrix. The matrices involved in the subtraction must have the same dimensions and cell value type.

You place the result into an existing matrix.

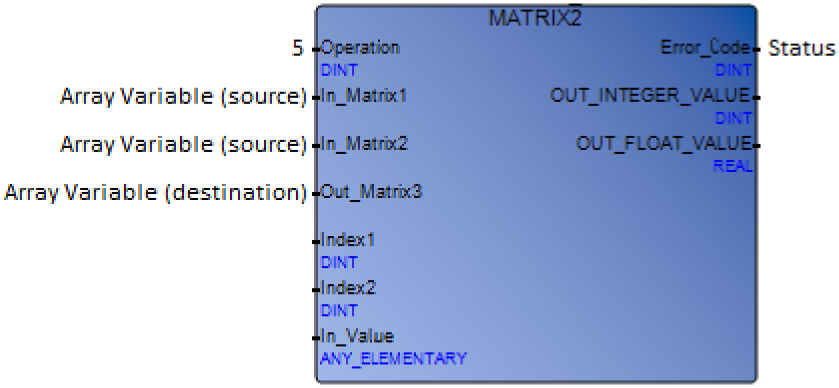
### Example

To subtract a matrix from another matrix then place the result in a third matrix. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Sub_4)

THEN

  (* Subtract the matrices *)

  Sub_4 := FALSE;

  IF (Use_Int_Matrix) THEN op1 := cmd4_sub; ELSE op2 := cmd4_sub;
END_IF;

END_IF;
```

```
(* FB for DINT operations *)
fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);
err1  := fbm1.Error_code;
out1i := fbm1.OUT_INTEGER_VALUE;
out1r := fbm1.OUT_FLOAT_VALUE;
op1   := 0;

(* FB for REAL operations *)
fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);
err2  := fbm2.Error_code;
out2i := fbm2.OUT_INTEGER_VALUE;
out2r := fbm2.OUT_FLOAT_VALUE;
op2   := 0;
```

## MULTIPLY_MATRIX




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 5. |
| In_Matrix1 | ANY_ELEMENTARY | Array variable for the first matrix in the multiplication |
| In_Matrix2 | ANY_ELEMENTARY | Array variable for the other matrix in the multiplication |
| Out_Matrix3 | ANY_ELEMENTARY | Array variable for the existing matrix that will receive the operation result. This must not be one of the matrices indicated in In_Matrix1 or In_Matrix2. |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>2 = Invalid type<br>3 = Type mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

Multiplies two existing matrices then places the result in a third matrix. The number of columns in the first matrix must be equal to the number of rows in the second matrix. The resulting matrix has the same number of rows as the first matrix and the same number of columns as the second matrix. For example, you can multiply a 3x4 matrix with a 4x2 matrix; the result will be a 3x2 matrix; however, you cannot multiply two 3x4 matrices. The matrices being multiplied must have the same cell value type.

The resulting matrix will be created with the required row-column structure and data type. You can place the result into an existing matrix.

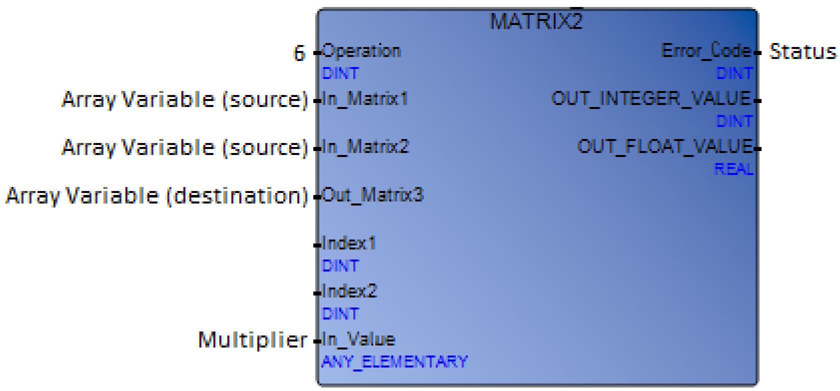
### Example

To multiply two matrices then place the result in a third matrix. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Mult_5)

THEN

   (* Multiply the matrices *)

  Mult_5 := FALSE;

  IF (Use_Int_Matrix) THEN op1 := cmd5_mult; ELSE op2 := cmd5_mult;
END_IF;

END_IF;


(* FB for DINT operations *)

fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);

err1  := fbm1.Error_code;

out1i := fbm1.OUT_INTEGER_VALUE;

out1r := fbm1.OUT_FLOAT_VALUE;

op1   := 0;


(* FB for REAL operations *)
```

```
fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);
err2  := fbm2.Error_code;
out2i := fbm2.OUT_INTEGER_VALUE;
out2r := fbm2.OUT_FLOAT_VALUE;
op2   := 0;
```

## SCALAR_MATRIX




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 6. |
| In_Matrix1 | ANY_ELEMENTARY | Array variable for the first matrix in the scalar operation. The cell value type must be float (REAL). |
| In_Matrix2 | ANY_ELEMENTARY | Array variable for the other matrix in the scalar operation. The cell value type must be float (REAL). |
| Out_Matrix3 | ANY_ELEMENTARY | Array variable for the existing matrix that will receive the operation result. The cell value type must be float (REAL). This must not be one of the matrices indicated in In_Matrix1 or In_Matrix2. |

| | | |
|---|---|---|
| Multiplier | ANY_ELEMENTARY | Number by which cell values are multiplied. This multiplier must be a float (REAL) value. |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>3 = Type mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

Multiplies each cell value of a float matrix by a float value then places the result in another matrix. This operation is called scalar multiplication.

You place the result into an existing matrix.

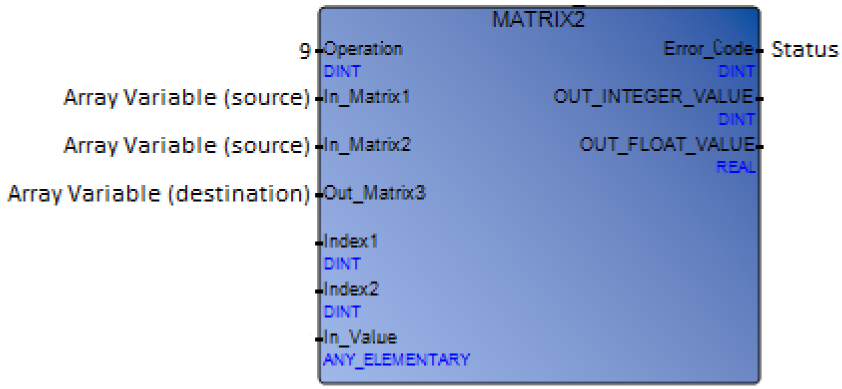
### Example

To multiply each cell of a float matrix by a float value then place the result in another matrix. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Scalar_6)

THEN

  (* Multiply each cell value of a float matrix by a value *)

  Scalar_6 := FALSE;

 IF (Use_Int_Matrix) THEN op1 := cmd6_scalar; ELSE op2 := cmd6_scalar;
END_IF;

END_IF;


(* FB for DINT operations *)

fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);

err1  := fbm1.Error_code;
```

```
out1i := fbm1.OUT_INTEGER_VALUE;
out1r := fbm1.OUT_FLOAT_VALUE;
op1   := 0;

(* FB for REAL operations *)
fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);
err2  := fbm2.Error_code;
out2i := fbm2.OUT_INTEGER_VALUE;
out2r := fbm2.OUT_FLOAT_VALUE;
op2   := 0;
```

## PRINT_MATRIX




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 9. |
| In_Matrix1 | ANY_ELEMENTARY | Array variable for the matrix |
| In_Matrix2 | ANY_ELEMENTARY | Array variable for the matrix |
| Out_Matrix3 | ANY_ELEMENTARY | Array variable for the matrix having the results of an operation executed on In_Matrix1 and In_Matrix2. |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>2 = Invalid type |

Description:

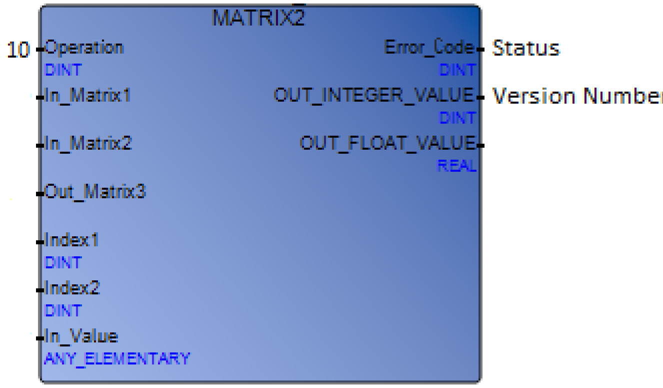
Prints the contents of matrices to a standard output, i.e., a console window.

### Example

To print the contents of all matrices onto a console window. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Print_9)

THEN

  (* Prints the content of the matrix *)

  Print_9 := FALSE;

  IF (Use_Int_Matrix) THEN op1 := cmd9_print; ELSE op2 := cmd9_print;
END_IF;

END_IF;


(* FB for DINT operations *)

fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);

err1  := fbm1.Error_code;

out1i := fbm1.OUT_INTEGER_VALUE;

out1r := fbm1.OUT_FLOAT_VALUE;

op1   := 0;


(* FB for REAL operations *)

fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);

err2  := fbm2.Error_code;

out2i := fbm2.OUT_INTEGER_VALUE;

out2r := fbm2.OUT_FLOAT_VALUE;

op2   := 0;
```

## GET_VERSION




Arguments:

The outputs other than those specified for the function do not contain valid information.

| | | |
|---|---|---|
| Operation | DINT | Number indicating the operation. The value of this operation is 10. |
| Error_Code | DINT | Status of the operation:<br>0 = No error<br>2 = Invalid type |
| OUT_INTEGER_VALUE | DINT | Version number of the function block |

Description:

Gets the version number for the function block. The version number is a 2-byte integer representing a major and minor version. For example, 16#0203 means v2.3.

### Example

To get the version number of the MATRIX2 function block. For a list of variable definitions used in the following example, refer to the Matrix2 Operations page.

```
IF (Get_ver_10)
```

```
THEN
  (* Get FB version *)
  Get_ver_10 := FALSE;
  IF (Use_Int_Matrix) THEN op1 := cmd10_get_ver; ELSE op2 :=
cmd10_get_ver; END_IF;
END_IF;

(* FB for DINT operations *)
fbm1( op1, mat1, mat2, mat3, idx11, idx12, in1i);
err1  := fbm1.Error_code;
out1i := fbm1.OUT_INTEGER_VALUE;
out1r := fbm1.OUT_FLOAT_VALUE;
op1   := 0;

(* FB for REAL operations *)
fbm2( op2, matR1, matR2, matR3, idx21, idx22, in2r);
err2  := fbm2.Error_code;
out2i := fbm2.OUT_INTEGER_VALUE;
out2r := fbm2.OUT_FLOAT_VALUE;
op2   := 0;
```

## Matrix Operations

A matrix is a two-dimensional array made up of rows and columns. It is mainly used to perform complex calculations involving the data of the running application. The matrix function block performs all of these operations. However, each operation has a specific identifier and requires different inputs. The outputs other than those specified for the function do not contain valid information.

The intersection of a row and a column is called a cell; cells hold the matrix values. The number of the first row of a matrix is 0; the number of its first column is also 0.

The Workbench offers built-in function blocks for creating, filling, and manipulating matrices. Each of the functions has an operation number ranging from 0 to 20.

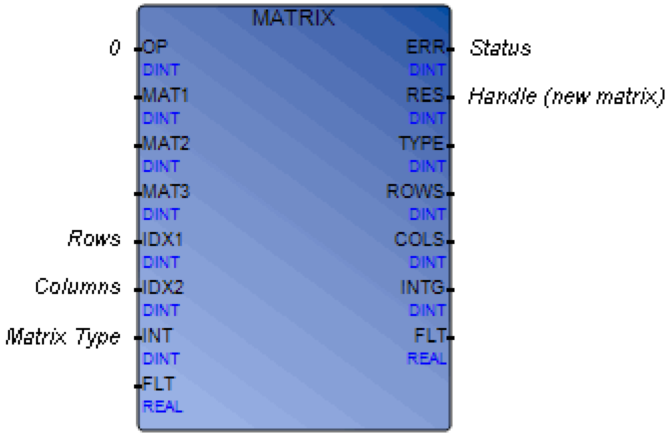
You can create as many matrices as required per program.

The available Matrix operations are the following:

| | |
|---|---|
| NEW_MATRIX | Creates a matrix |
| FREE_MATRIX | Closes a matrix |
| PUT_I_MATRIX | Inserts an integer into a cell of an integer matrix |
| GET_I_MATRIX | Reads the value of a cell in an integer matrix |
| PUT_F_MATRIX | Inserts a float value into a cell of a float matrix |
| GET_F_MATRIX | Reads the value of a cell in a float matrix |
| DUP_MATRIX | Creates a duplicate of an existing matrix |
| COPY_MATRIX | Copies the contents of a matrix into an existing matrix having the same row-column structure and cell value type |
| COPY_ROW_MATRIX | Copies a row from a matrix into a row of the same size in another matrix or into the same matrix |
| COPY_COL_MATRIX | Copies a column from a matrix into a row of the same size in another matrix or into the same matrix |

| | |
|---|---|
| TYPE_MATRIX | Returns the data type of the cell values of a matrix |
| ROWS_MATRIX | Returns the number of rows in a matrix |
| COLS_MATRIX | Returns the number of columns in a matrix |
| TRANSPOSE_MATRIX | Swaps the rows and columns of an existing matrix into another matrix |
| INVERT_MATRIX | Computes the inverse of a matrix |
| ADD_MATRIX | Adds up two existing matrices |
| SUBTRACT_MATRIX | Subtracts an existing matrix from another existing matrix |
| MULTIPLY_MATRIX | Multiplies two existing matrices |
| SCALAR_I_MATRIX | Multiplies each cell value of an integer matrix by an integer value |
| SCALAR_F_MATRIX | Multiplies each cell value of a float matrix by a float value |
| PRINT_MATRIX | Sends the contents of a matrix to the errlog |

## NEW_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. This operation number is 0. |
| INDEX_1 | IDX1 | DINT | Number of rows. The possible values range from 0 to N-1, N being the total number of rows. |
| INDEX_2 | IDX2 | DINT | Number of columns. The possible values range from 0 to M-1, M being the total number of columns. |
| IN_INTEGER_VALUE | INT | DINT | Number indicating the type of matrix:<br>0 = Integer<br>1 = Float |

| | | | |
|---|---|---|---|
| MATRIX_RESULT | RES | DINT | Handle of the new matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>1 = Not enough memory<br>2 = Invalid type |

Description:

**Warning:** This function uses the Malloc dynamic memory allocation at run time.

Creates a matrix. The data type of all cells is the same for any matrix. Therefore, an integer matrix contains only integer values, and a float matrix, float values.

### Examples

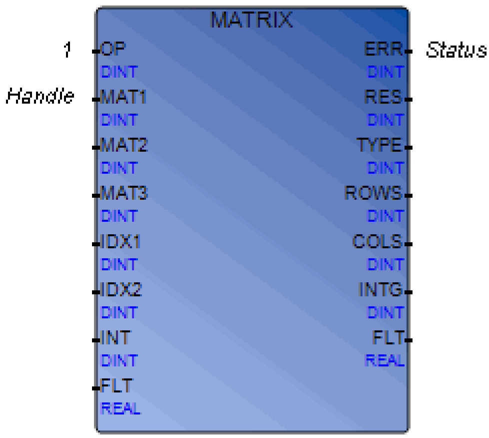
To create a float-type matrix having three columns and three rows:

```
matrix_fbl(0, 0, 0, 0, 3, 3, 1, 0.0); (* new float matrix 3 x 3*)
     if matrix_fbl.ERROR_CODE = 0 then
            mat[1] := matrix_fbl.MATRIX_RESULT;
     else
            RESULT := log_msg('ErrLog','unable to allocate matrix ' +
            any_to_string(matrix_fbl.ERROR_CODE));
     end_if;
```

To create an integer-type matrix having two columns and two rows:

```
matrix_fbl(0, 0, 0, 0, 2, 2, 0, 0.0); (* new integer matrix 2 x 2*)
     if matrix_fbl.ERROR_CODE = 0 then
               mat[2] := matrix_fbl.MATRIX_RESULT;
     else
               RESULT := log_msg('ErrLog','unable to allocate matrix
               ' + any_to_string(matrix_fbl.ERROR_CODE));
     end_if;
```

# FREE_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 1. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>6 = Index out of range |

Description:

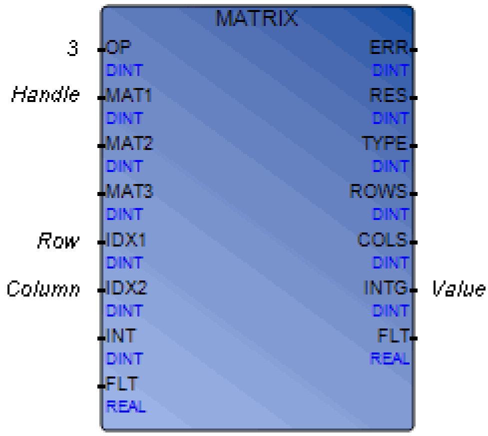
Closes a matrix.

### Example

To close the matrix having the handle indicated by the `index` variable:

```
FOR index := 1 TO 10 BY 1 DO
if mat[index] > 0 then
matrix_fbl(1, mat[index], 0, 0, 0, 0, 0, 0.0); (* free mat[index] *)
if matrix_fbl.ERROR_CODE > 0 then
RESULT := log_msg('ErrLog','unable to free matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if;
end_if;
END_FOR;
```

## GET_I_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. This operation number is 3. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix |
| INDEX_1 | IDX1 | DINT | Row number of the cell. The possible values range from 0 to N-1, N being the total number of rows. |
| INDEX_2 | IDX2 | DINT | Column number of the cell. The possible values range from 0 to M-1, M being the total number of columns. |
| OUT_INTEGER_VALUE | INTG | DINT | Integer value contained in the cell |

Description:

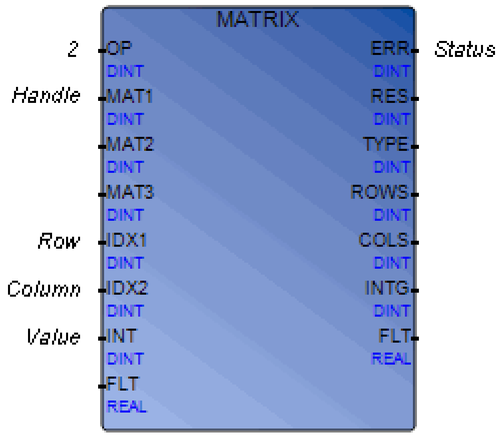
Reads the value of a cell in an integer matrix.

### Example

To get the integer value held in the cell located in the first column and first row of the matrix having the handle 2 and place it into the `ivalue` variable:

```
matrix_fbl(3, mat[2], 0, 0, 1, 2, 0, 0.0); (* ivalue = mat[1][1,2] *)
ivalue := matrix_fbl.out_integer_value;
```

## PUT_I_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. This operation number is 2. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix |
| INDEX_1 | IDX1 | DINT | Row number of the cell. The possible values range from 0 to N-1, N being the total number of rows. |
| INDEX_2 | IDX2 | DINT | Column number of the cell. The possible values range from 0 to M-1, M being the total number of columns. |

| | | | |
|---|---|---|---|
| IN_INTEGER_VALUE | INT | DINT | Value to be inserted |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>3 = Type mismatch<br>6 = Index out of range |

Description:

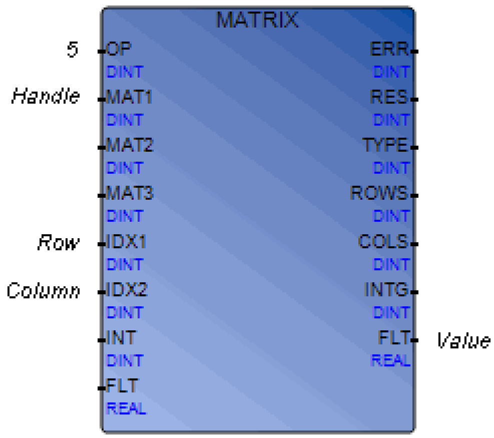
Inserts an integer into a cell of an integer matrix.

### Example

To set the values of the cells in the first and second columns of the first row to 2 and -1 respectively:

```
matrix_fbl(2, mat[2], 0, 0, 0,0, 2, 0.0);
matrix_fbl(2, mat[2], 0, 0, 0,1, -1, 0.0)
```

## GET_F_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. This operation number is 5. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix |
| iNDEX_1 | IDX1 | DINT | Row number of the cell. The possible values range from 0 to N-1, N being the total number of rows. |
| iNDEX_2 | IDX2 | DINT | Column number of the cell. The possible values range from 0 to M-1, M being the total number of columns. |
| OUT_FLOAT_VALUE | FLT | REAL | Returns the float value contained in the cell |

Description:

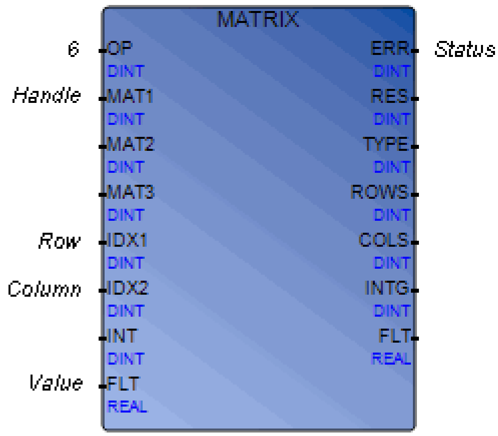
Reads the value of a cell in a float matrix.

### Example

To get the float value from the cell in the second row and third column of the matrix having the handle 1 and place it in the `fvalue` variable:

```
matrix_fbl(5, mat[1], 0, 0, 1, 2, 0, 0.0); (* fvalue =mat[1][1,2]*)
fvalue := matrix_fbl.out_float_value;
```

## PUT_F_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. This operation number is 4. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix |
| INDEX_1 | IDX1 | DINT | Row number of the cell. The possible values range from 0 to N-1, N being the total number of rows. |
| INDEX_2 | IDX2 | DINT | Column number of the cell. The possible values range from 0 to M-1, M being the total number of columns. |

| | | | |
|---|---|---|---|
| IN_FLOAT_VALUE | FLT | REAL | Value to be inserted |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>3 = Type mismatch<br>6 = Index out of range |

Description:

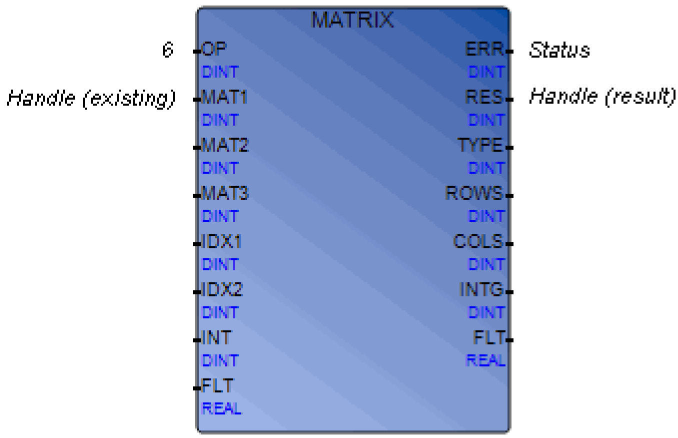
Inserts a float value into a cell of a float matrix.

### Example

To place the value 2.0 into the cell in the first row and first column in the matrix having the handle 1:

```
matrix_fbl(4, mat[1], 0, 0, 0, 0, 0, 2.0)
```

## DUP_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 6. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix |
| MATRIX_RESULT | RES | DINT | Handle of the new matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>7 = Out of range |

Description:

**Warning:** This function uses the Malloc dynamic memory allocation at run time.

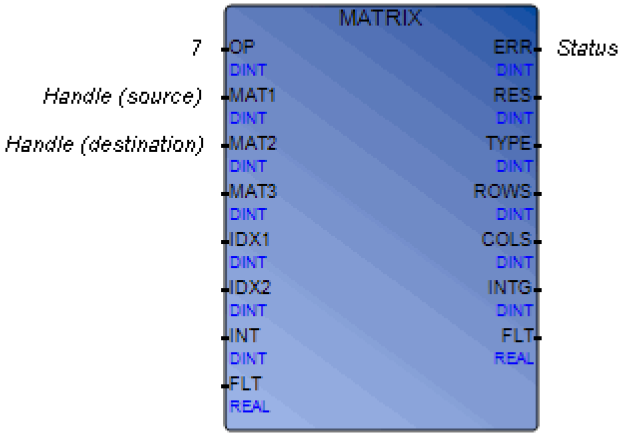
Creates a duplicate of an existing matrix. The duplicate matrix will have the same structure and contents as the original one. The duplicate matrix will be created with the required row-column structure and data type. If the matrix already exists, it will be deleted then recreated.

### Example

To duplicate the matrix having the handle 1:

```
matrix_fbl(6, mat[1], 0, 0, 0, 0, 0, 0.0); (* duplicate mat[1] *)
if matrix_fbl.ERROR_CODE = 0 then
mat[3] := matrix_fbl.MATRIX_RESULT;
else
RESULT := log_msg('ErrLog','unable to duplicate matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## COPY_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 7. |
| MATRIX_1 | MAT1 | DINT | Handle of the source matrix |
| MATRIX_2 | MAT2 | DINT | Handle of the destination matrix. This must not be the source matrix. |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>3 = Type mismatch<br>4 = Row mismatch<br>5 = Column mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

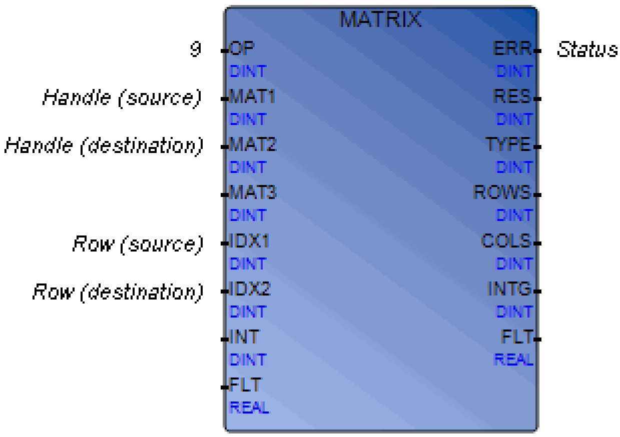
Copies the contents of a matrix into an existing matrix having the same row-column structure and cell value type.

### Example

To copy the contents of the matrix having the handle 1 and place it into the matrix having the handle 3:

```
matrix_fbl(7, mat[1], mat[3], 0, 0, 0, 0, 0.0); (* mat[3]=mat[1] *)
if matrix_fbl.ERROR_CODE = 0 then
mat[3] := matrix_fbl.MATRIX_RESULT;
else
RESULT := log_msg('ErrLog','unable to duplicate matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## COPY_ROW_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 8. |
| MATRIX_1 | MAT1 | DINT | Handle of the source matrix |
| INDEX_1 | IDX1 | DINT | Number of the row, in the source matrix, that is copied. The possible values range from 0 to N-1, N being the total number of rows. |
| MATRIX_2 | MAT2 | DINT | Handle of the destination matrix. This must not be the source matrix. |

| | | | |
|---|---|---|---|
| INDEX_2 | IDX2 | DINT | Number of the row, in the destination matrix, that receives a row. The possible values range from 0 to N-1, N being the total number of rows. |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>3 = Type mismatch<br>5 = Column mismatch<br>6 = Index out of range |

Description:

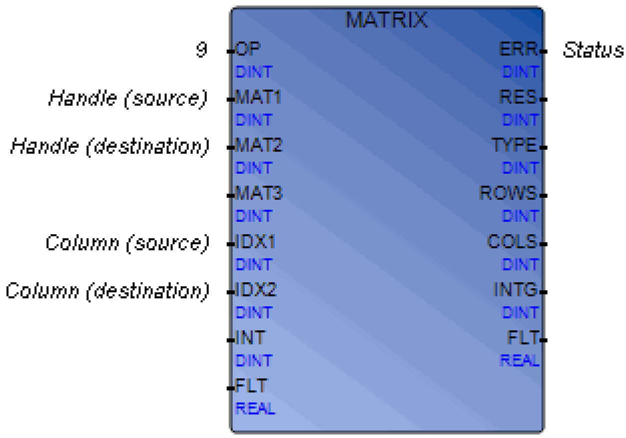
Copies a row from a matrix into a row of the same size in another matrix or into the same matrix. The cell value type must be the same in both matrices.

### Example

To copy the contents of the second row of the matrix having the handle 1 and place it into the third row of the matrix having the handle 3:

```
matrix_fbl(8, mat[1], mat[3], 1, 2, 0, 0, 0.0); (* mat[3][2,0..M] =
mat[1][1,0..M] *)
if matrix_fbl.ERROR_CODE > 0 then
RESULT := log_msg('ErrLog','unable to copy row matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## COPY_COL_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 9. |
| MATRIX_1 | MAT1 | DINT | Handle of the source matrix |
| INDEX_1 | IDX1 | DINT | Number of the column, in the source matrix, that is copied. The possible values range from 0 to M-1, M being the total number of columns. |
| MATRIX_2 | MAT2 | DINT | Handle of the destination matrix. This must not be the source matrix. |

| | | | |
|---|---|---|---|
| INDEX_2 | IDX2 | DINT | Number of the row, in the destination matrix, that receives a column. The possible values range from 0 to M-1, M being the total number of columns. |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>2 = Invalid type<br>3 = Type mismatch<br>4 = Row mismatch<br>6 = Index out of range |

Description:

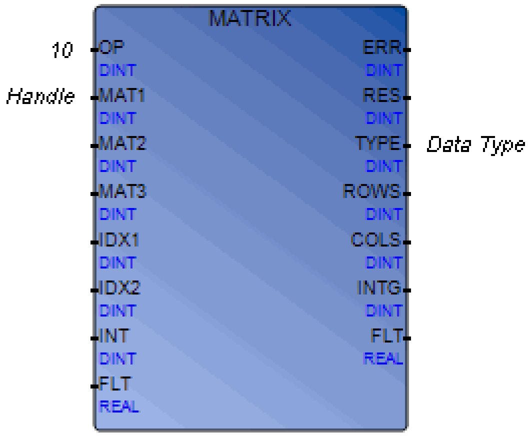
Copies a column from a matrix into a row of the same size in another matrix or into the same matrix. The cell value type must be the same in both matrices.

### Example

To copy the contents of the second column of the matrix having the handle 1 and place it into the third column of the matrix having the handle 3:

```
matrix_fbl(9, mat[1], mat[3], 1, 2, 0, 0, 0.0); (* mat[3][0..N,2] =
mat[1][0..N,1] *)
if matrix_fbl.ERROR_CODE > 0 then
RESULT := log_msg('ErrLog','unable to copy col matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## TYPE_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 10. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix. This number is the result of the NEW_MATRIX operation, when the matrix was created. |
| MATRIX_TYPE | TYPE | DINT | Data type of the cells:<br>0 = Integer<br>1 = Float |

Description:

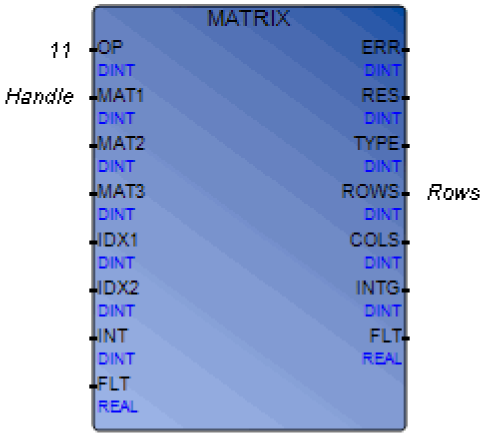
Returns the data type of the cell values of a matrix.

## Example

To get the type of cells contained in the matrix having the handle 1 and place it in the `mat_type` variable:

```
matrix_fbl(10, mat[1], 0, 0, 0, 0, 0, 0.0); (* get mat[1] type
(integer/float)*)
mat_type := matrix_fbl.matrix_type
```

## ROWS_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 11. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix |
| MATRIX_ROWS | ROWS | DINT | Number of rows in the matrix. The possible values range from 0 to N-1, N being the total number of rows. |

Description:

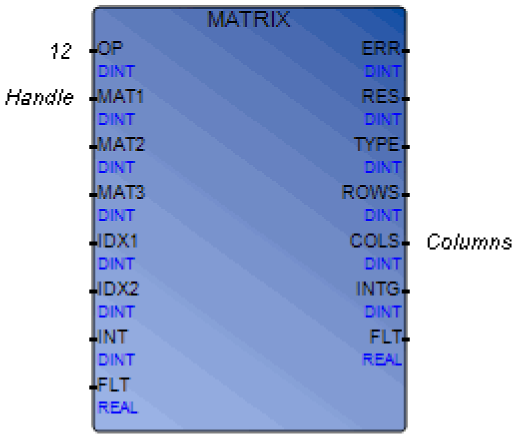
Returns the number of rows in a matrix.

### Example

To get the number of rows contained in the matrix having the handle 1:

```
matrix_fbl(11, mat[1], 0, 0, 0, 0, 0, 0.0); (* get mat[1] number of
rows *)
rows := matrix_fbl.matrix_rows
```

## COLS_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 12. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix |
| MATRIX_COLS | COLS | DINT | Number of columns in the matrix. The possible values range from 0 to M-1, M being the total number of columns. |

Description:

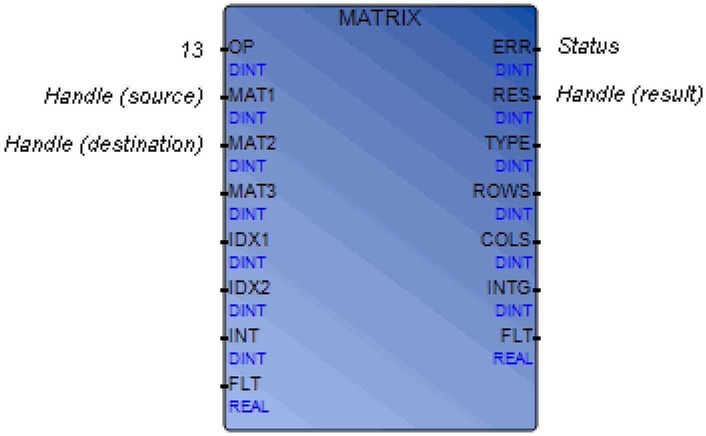
Returns the number of columns in a matrix.

### Example

To get the number of columns contained in the matrix having the handle 1:

```
matrix_fbl(12, mat[1], 0, 0, 0, 0, 0, 0.0); (* get mat[1] number of
columns *)
cols := matrix_fbl.matrix_cols
```

## TRANSPOSE_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 13. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix to be transposed (source) |
| MATRIX_2 | MAT2 | DINT | Handle of the matrix to receive the resulting transposed matrix. This must not be the source matrix. A value of 0 indicates that a new matrix will be created. |

| | | | |
|---|---|---|---|
| MATRIX_RESULT | RES | DINT | Handle of the resulting transposed matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

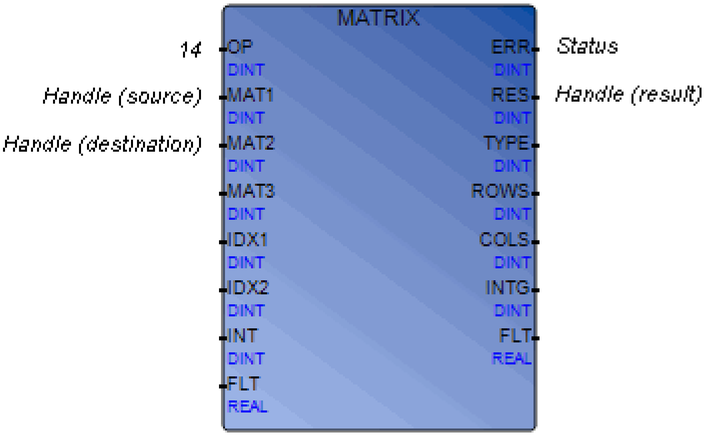
Swaps the rows and columns of an existing matrix into another matrix called a transpose. For instance, the transpose of a matrix having three rows and five columns has five rows and three columns. The transpose matrix will be created with the required row-column structure and data type. You can choose to place the transposed matrix into an existing matrix or create a new one.

### Example

To swap the rows and columns of the matrix having the handle 1 and place the result in a new matrix:

```
matrix_fbl(13, mat[1], 0, 0, 0, 0, 0, 0.0); (* transpose mat[1] *)
if matrix_fbl.ERROR_CODE = 0 then
mat[4] := matrix_fbl.MATRIX_RESULT;
else
RESULT := log_msg('ErrLog','unable to transpose matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## INVERT_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 14. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix to be inverted (source) |
| MATRIX_2 | MAT2 | DINT | Handle of the matrix to receive the resulting inverted matrix. This must not be the source matrix. A value of 0 indicates that a new matrix will be created. |

| | | | |
|---|---|---|---|
| MATRIX_RESULT | RES | DINT | Handle of the resulting inverted matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>2 = Invalid type<br>3 = Type mismatch<br>6 = Dimension mismatch<br>7 = Index out of range<br>8 = Not square<br>9 = Mathematical error |

Description:

**Warning:** This function uses the Malloc dynamic memory allocation at run time.

Computes the inverse of a matrix. The source matrix must be square (i.e., have the same number of rows and columns) and its cell value type must be float. The inverse matrix will be created with the required row-column structure and data type.

You can choose to place the inverted matrix into an existing matrix or create a new one.

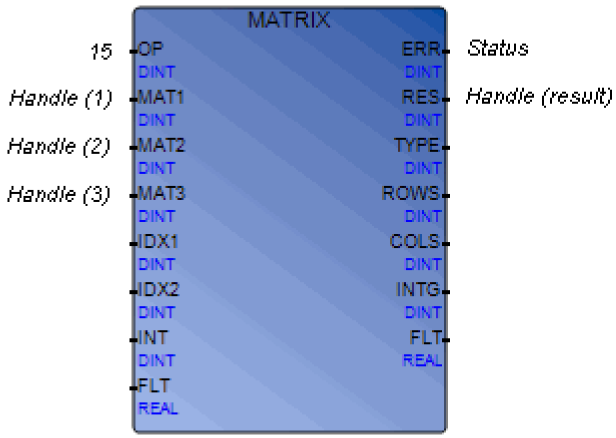
**Note:** Not all matrices are invertible. Invertible matrices are those whose determinant is not equal to 0.

## Example

To invert the matrix having the handle 1 and place the result in a new matrix:

```
matrix_fbl(14, mat[1], 0, 0, 0, 0, 0, 0.0); (* invert mat[1] *)
if matrix_fbl.ERROR_CODE = 0 then
mat[4] := matrix_fbl.MATRIX_RESULT;
else
RESULT := log_msg('ErrLog','unable to inverse matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

# ADD_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 15. |
| MATRIX_1 | MAT1 | DINT | Handle of the first matrix in the addition |
| MATRIX_2 | MAT2 | DINT | Handle of the other matrix in the addition |
| MATRIX_3 | MAT3 | DINT | Handle of the existing matrix that will receive the operation result. This must not be one of the matrices indicated in MAT_1 or MAT_2. A value of 0 indicates that result of the operation is sent to a new matrix. |

| | | | |
|---|---|---|---|
| MATRIX_RESULT | RES | DINT | Handle of the resulting matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>2 = Invalid type<br>3 = Type mismatch<br>4 = Row mismatch<br>5 = Column mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

Adds up two existing matrices then places the result in a third matrix. The summation is performed cell by cell, with the result occupying the same cell position in the third matrix. The matrices that are added up must have the same dimensions and cell value type.

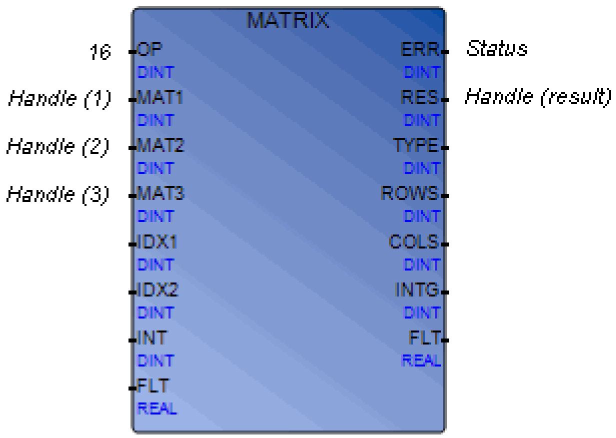
You can choose to place the result into an existing matrix or create a new one.

## Example

To add the matrix having the handle 1 and another having the handle 4 then place the result in a new matrix:

```
matrix_fbl(15, mat[1], mat[4], 0, 0, 0, 0, 0.0); (* mat[1]+mat[4]*)
if matrix_fbl.ERROR_CODE = 0 then
mat[5] := matrix_fbl.MATRIX_RESULT;
else
RESULT := log_msg('ErrLog','unable to add matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## SUBTRACT_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 16. |
| MATRIX_1 | MAT1 | DINT | Handle of the first matrix in the subtraction |
| MATRIX_2 | MAT2 | DINT | Handle of the other matrix in the subtraction |
| MATRIX_3 | MAT3 | DINT | Handle of the existing matrix that will receive the operation result. This must not be one of the matrices indicated in MAT_1 or MAT_2. A value of 0 indicates that result of the operation is sent to a new matrix. |

| | | | |
|---|---|---|---|
| MATRIX_RESULT | RES | DINT | Handle of the resulting matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>2 = Invalid type<br>3 = Type mismatch<br>4 = Row mismatch<br>5 = Column mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

Subtracts an existing matrix from another existing matrix then places the result in a third matrix. The difference is performed cell by cell, with the result occupying the same cell position in the third matrix. The matrices involved in the subtraction must have the same dimensions and cell value type.

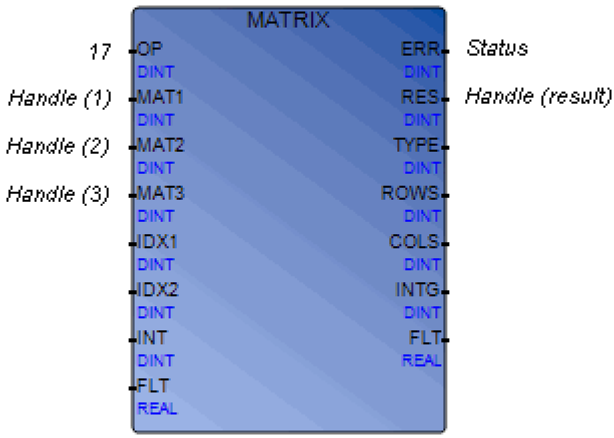
You can choose to place the result into an existing matrix or create a new one.

## Example

To subtract the matrix having the handle 4 from the matrix having the handle 1 then place the result in a new matrix:

```
matrix_fbl(16, mat[1], mat[4], 0, 0, 0, 0, 0.0); (* mat[1]-mat[4]*)
if matrix_fbl.ERROR_CODE = 0 then
mat[6] := matrix_fbl.MATRIX_RESULT;
else
RESULT := log_msg('ErrLog','unable to sub matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## MULTIPLY_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 17. |
| MATRIX_1 | MAT1 | DINT | Handle of the first matrix in the multiplication |
| MATRIX_2 | MAT2 | DINT | Handle of the other matrix in the multiplication |
| MATRIX_3 | MAT3 | DINT | Handle of the existing matrix that will receive the operation result. This must not be one of the matrices indicated in MAT_1 or MAT_2. A value of 0 indicates that result of the operation is sent to a new matrix. |

| | | | |
|---|---|---|---|
| MATRIX_RESULT | RES | DINT | Handle of the resulting matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>2 = Invalid type<br>3 = Type mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

Multiplies two existing matrices then places the result in a third matrix. The number of columns in the first matrix must be equal to the number of rows in the second matrix. The resulting matrix has the same number of rows as the first matrix and the same number of columns as the second matrix. For example, you can multiply a 3x4 matrix with a 4x2 matrix; the result will be a 3x2 matrix; however, you cannot multiply two 3x4 matrices. The matrices being multiplied must have the same cell value type.

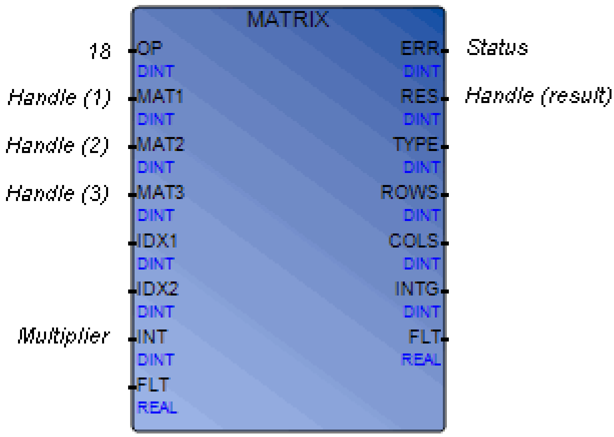
The resulting matrix will be created with the required row-column structure and data type. You can choose to place the result into an existing matrix or create a new one.

## Example

To multiply the matrix having the handle 1 and the matrix having the handle 4 then place the result in a new matrix:

```
matrix_fbl(17, mat[1], mat[4], 0, 0, 0, 0, 0.0); (* mat[1]*mat[4]*)
if matrix_fbl.ERROR_CODE = 0 then
mat[7] := matrix_fbl.MATRIX_RESULT;
else
RESULT := log_msg('ErrLog','unable to multiply matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## SCALAR_I_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 18. |
| MATRIX_1 | MAT1 | DINT | Handle of the first matrix in the scalar operation |
| MATRIX_2 | MAT2 | DINT | Handle of the other matrix in the scalar operation |
| MATRIX_3 | MAT3 | DINT | Handle of the existing matrix that will receive the operation result. This must not be one of the matrices indicated in MAT_1 or MAT_2. A value of 0 indicates that result of the operation is sent to a new matrix. |
| IN_INTEGER_VALUE | INT | DINT | Number by which cell values are multiplied |

| | | | |
|---|---|---|---|
| MATRIX_RESULT | RES | DINT | Handle of the resulting matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>3 = Type mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

**Warning:** This function uses the Malloc dynamic memory allocation at run time.

Multiplies each cell value of an integer matrix by an integer value then places the result in another matrix. This operation is called scalar multiplication.

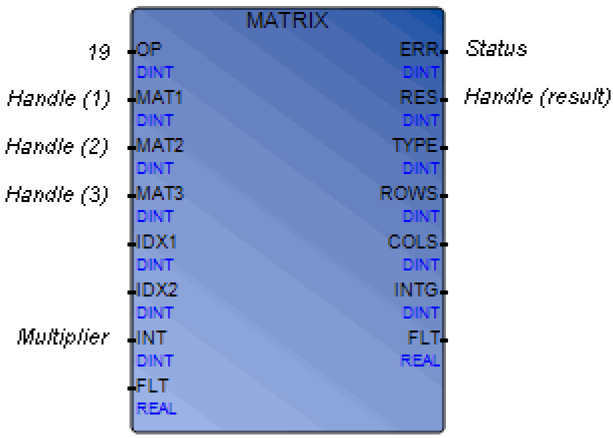
You can choose to place the result into an existing matrix or create a new one.

### Example

To multiply each cell of the matrix having the handle 2 by the value 4 then place the result in a new matrix:

```
matrix_fbl(18, mat[2], 0 , 0, 0, 0, 4, 0.0); (* mat[2] * 4 *)
if matrix_fbl.ERROR_CODE = 0 then
mat[8] := matrix_fbl.MATRIX_RESULT;
else
RESULT := log_msg('ErrLog','unable to scalar i matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## SCALAR_F_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 19. |
| MATRIX_1 | MAT1 | DINT | Handle of the first matrix in the scalar operation |
| MATRIX_2 | MAT2 | DINT | Handle of the other matrix in the scalar operation |
| MATRIX_3 | MAT3 | DINT | Handle of the existing matrix that will receive the operation result. This must not be one of the matrices indicated in MAT_1 or MAT_2. A value of 0 indicates that result of the operation is sent to a new matrix. |
| IN_FLOAT_VALUE | FLT | FLT | Number by which cell values are multiplied |

| | | | |
|---|---|---|---|
| MATRIX_RESULT | RES | DINT | Handle of the resulting matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>1 = Not enough memory<br>3 = Type mismatch<br>6 = Dimension mismatch<br>7 = Index out of range |

Description:

**Warning:** This function uses the Malloc dynamic memory allocation at run time.

Multiplies each cell value of a float matrix by a float value then places the result in another matrix. This operation is called scalar multiplication.

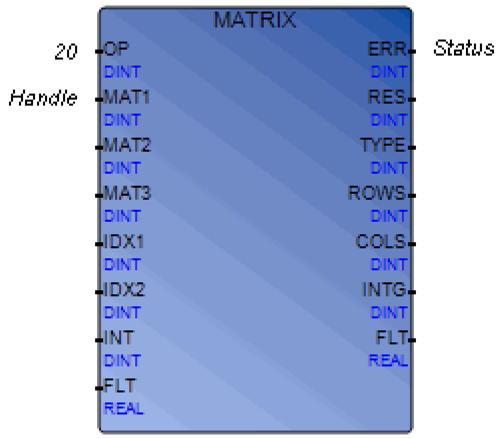
You can choose to place the result into an existing matrix or create a new one.

### Example

To multiply each cell of the matrix having the handle 1 by the value 5.0 then place the result in a new matrix:

```
matrix_fbl(19, mat[1], 0 , 0, 0, 0, 0, 5.0); (* mat[2] * 5.0 *)
if matrix_fbl.ERROR_CODE = 0 then
mat[9] := matrix_fbl.MATRIX_RESULT;
else
RESULT := log_msg('ErrLog','unable to scalar f matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if
```

## PRINT_MATRIX




**Note:** The failover mechanism does not support the Matrix function blocks.

Arguments:

You need to enter a value for each input parameter that appears blank. All blank inputs require a 0 except for the FLT which requires 0.0. The outputs other than those specified for the function do not contain valid information.

| | | | |
|---|---|---|---|
| OPERATION | OP | DINT | Number indicating the operation. The value of this operation is 20. |
| MATRIX_1 | MAT1 | DINT | Handle of the matrix |
| ERROR_CODE | ERR | DINT | Status of the operation:<br>0 = No error<br>2 = Invalid type |

Description:

Sends the contents of a matrix to the errlog. The default errlog for the workbench is e.log.

### Example

To send the contents of the matrix having the handle held in the `index` variable to the ErrLog file:

```
FOR index := 1 TO 10 BY 1 DO
if mat[index] > 0 then
RESULT := log_msg('ErrLog','print matrix ' + any_to_string(index));
matrix_fbl(20, mat[index], 0 , 0, 0, 0, 0, 0.0); (* print mat[index] *)
if matrix_fbl.ERROR_CODE > 0 then
RESULT := log_msg('ErrLog','unable to print matrix ' +
any_to_string(matrix_fbl.ERROR_CODE));
end_if;
end_if;
END_FOR
```

# Motion Control Function Blocks

Motion control function blocks control axes using language elements defined in the IEC 61131-3 standard.

The following data types, arrays, and parameters are used in motion control function blocks:

- AXIS_REF
- MC_CAM_ID
- MC_CAM_REF
- MC_CAMSWITCH_REF
- MC_CAMSWITCH_REF parameters
- MC_INPUT_REF
- MC_OUTPUT_REF
- MC_StartMode
- MC_TA
- MC_TA_REF
- MC_TAArray
- MC_TRACK_REF
- MC_TRACK_REF parameters
- MC_TRIGGER_REF
- MC_TP
- MC_TP_REF
- MC_TPArray
- MC_TV
- MC_TV_REF
- MC_TVArray

Motion Control function blocks perform various motion control operations:

| | |
|---|---|
| **MC_AbortTrigger** | Connection of trigger events |
| **MC_AccelerationProfile** | Direction of time-acceleration locked motion profiles |
| **MC_CamIn** | Initiation of the CAM |
| **MC_CamOut** | Disconnection of the slave from the master axis |
| **MC_CamTableSelect** | Selection of CAM tables |
| **MC_DigitalCamSwitch** | Initiation a motor shaft |
| **MC_GearIn** | Controlling of a velocity ratio |
| **MC_GearInPos** | Controlling of a gear ratio between positions |
| **MC_GearOut** | Disconnection of the slave from the master axis |
| **MC_Halt** | Controlling of motion halts |

| | |
|---|---|
| **MC_Home** | Operation of search home sequences |
| **MC_MoveAbsolute** | Movement to specified absolute positions |
| **MC_MoveAdditive** | Movement to a specified distance |
| **MC_MoveContinuousAbsolute** | Controlled motion to a specified absolute position ending with specified velocity |
| **MC_MoveContinuousRelative** | Controlled motion to a specified relative distance ending with specified velocity |
| **MC_MoveRelative** | Movement relative to the current position |
| **MC_MoveSuperimposed** | Movement to a position an additional distance from the current position |
| **MC_MoveVelocity** | Controlled continuous motion at specified velocity |
| **MC_Phasing** | Modification to create a phase shift |
| **MC_PositionProfile** | Controlling of a time-position locked motion profile |
| **MC_Power** | Controlling of power stages; on and off |
| **MC_ReadActualPosition** | Yielding of actual positions |
| **MC_ReadActualTorque** | Yielding of actual torque values |
| **MC_ReadActualVelocity** | Yielding of actual velocity values |
| **MC_ReadAxisError** | Yielding of axis errors |
| **MC_ReadBoolParameter** | Yielding of the value of a specific BOOL parameters |
| **MC_ReadDigitalInput** | Yielding of specific input values |
| **MC_ReadDigitalOutput** | Yielding of specific output values |
| **MC_ReadParameter** | Yielding of specific parameter values |
| **MC_ReadStatus** | Yielding of axis status |
| **MC_Reset** | Removal of all axis-related internal errors |
| **MC_SetOverride** | Specification of the override value for an axis |
| **MC_SetPosition** | Specification of the position of an axis |
| **MC_Stop** | Direction of a controlled motion stop |

| | |
|---|---|
| **MC_TorqueControl** | Direction continuous torque |
| **MC_TouchProbe** | Recording of current axis position |
| **MC_VelocityProfile** | Direction of a time-velocity locked motion profile |
| **MC_WriteBoolParameter** | Modification of specific BOOL parameter values |
| **MC_WriteDigitalOutput** | Modification of specific output values |
| **MC_WriteParameter** | Modification of specific parameter values |

AXIS_REF data type:

```
AXIS_REF

      DISPLAY: AxisNo
STRUCT
      AxisNo : DINT;
END_STRUCT
```

The AXIS_REF data type is a structure containing information about a specific axis.

MC_CAM_ID data type:

```
MC_CAM_ID

      DISPLAY: CamID
STRUCT
      CamID : DINT;
      CamTableIndex : DINT;
END_STRUCT
```

MC_CAM_REF data type:

```
MC_CAM_REF

      DISPLAY: CamID
```

```
STRUCT
      CamID : DINT;
      CamName : STRING(32);
      CamParam1 : DINT;
      CamParam2 : REAL;
END_STRUCT
```

MC_CAMSWITCH_REF data type:

```
MC_CAMSWITCH_REF

      DISPLAY: TrackNumber
STRUCT
      TrackNumber : DINT;
      FirstOnPosition : REAL;
      LastOnPosition : REAL;
      AxisDirection : DINT;
      CamSwitchMode : DINT;
      Duration : TIME;
END_STRUCT
```

MC_CAMSWITCH_REF parameters:

| Parameter Name | Data Type | Description |
|---|---|---|
| TrackNumber | INT | References the track |
| FirstOnPosition [u] | REAL | The lower boundary of where the switch is ON |
| LastOnPosition [u] | REAL | The upper boundary of where the switch is ON |
| AxisDirection | INT | 0 = both directions; the default value<br>1 = positive<br>2 = negative |

| | | |
|---|---|---|
| CamSwitchMode | INT | 0 = position based; default value<br>1 = time based |
| Duration | TIME | Coupled to the time-based CamSwitchMode |

MC_INPUT_REF data type:

```
MC_INPUT_REF

      DISPLAY: Input_ID
STRUCT
      InputID : DINT;
END_STRUCT
```

MC_OUTPUT_REF data type:

```
MC_OUTPUT_REF

      DISPLAY: OutputID
STRUCT
      OutputID : DINT;
END_STRUCT
```

MC_StartMode data type:

```
MC_StartMode

      DISPLAY: Mode
STRUCT
      Mode : DINT;
      StartParam : DINT;
END_STRUCT
```

MC_TA data type:

```
MC_TA

      DISPLAY: delta_time
STRUCT
      delta_time : TIME;
      acceleration : REAL;
END_STRUCT
```

MC_TA_REF data type:

```
MC_TA_REF

      DISPLAY: Number_of_pairs
STRUCT
      Number_of_pairs : DINT;
      IsAbsolute : BOOL;
      MC_TA_Array : MC_TAArray;
END_STRUCT
```

MC_TAArray data type:

```
MC_TAArray

      ARRAY[1..16]
      OF MC_TA
```

MC_TRACK_REF data type:

```
MC_TRACK_REF

      DISPLAY: TrackID
STRUCT
      TrackID : DINT;
      OnCompensation : TIME;
      OffCompensation : TIME;
      Hysteresis : REAL;
END_STRUCT
```

MC_TRACK_REF parameters:

| Parameter Name | Data Type | Description |
|---|---|---|
| TrackID | DINT | References the track |
| OnCompensation | TIME | Time that the switching ON is advanced or delayed per track |
| OffCompensation | TIME | Time that the switching OFF is delayed per track |
| Hysteresis [u] | REAL | Positive or negative distance from the switching point where the switch is not executed<br><br>**Note:** You can set different Hysteresis values for each track |

MC_TRIGGER_REF data type arguements:

```
MC_TRIGGER_REF

      DISPLAY: Trigger_ID
STRUCT
      Trigger_ID : DINT;
END_STRUCT
```

MC_TP data type:

```
MC_TP

      DISPLAY: delta_time
STRUCT
      delta_time : TIME;
      position : REAL;
END_STRUCT
```

MC_TP_REF data type:

```
MC_TP_REF

      DISPLAY: Number_of_pairs
STRUCT
      Number_of_pairs : DINT;
      IsAbsolute : BOOL;
      MC_TP_Array : MC_TPArray;
END_STRUCT
```

MC_TPArray data type:

```
MC_TPArray

      Array [1..16]
      OF MC_TP
```

MC_TV data type:

```
MC_TV

      DISPLAY: delta_time
STRUCT
      delta_time : TIME;
      velocity : REAL
END_STRUCT
```

MC_TV_REF data type:

```
MC_TV_REF

      DISPLAY: Number_of_Pairs
STRUCT
      Number_of_Pairs : DINT;
      InAbsolute : BOOL
      MC_TV_Array : MC_TVArray
END_STRUCT
```

MC_TVArray data type:

```
MC_TVArray
      Array [1..16]
      OF MC_TV
```

## MC_AbortTrigger


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | Specifies axis connected to the trigger functionality |
| TriggerInp | TgIn | MC_TRIGGER_REF | Reference to trigger signal source |
| Execute | Exec | BOOL | Aborts the trigger event at the rising edge |
| Axis | Axis | AXIS_REF | Specifies axis connected to the trigger functionality |
| TriggerInput | TrIn | MC_TRIGGER_REF | Reference to trigger signal sourc |
| Done | Done | BOOL | Trigger functionality aborted |
| Busy | Busy | BOOL | Function block is unfinished |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Aborts function blocks that are connected to trigger events

# MC_AccelerationProfile


Arguments:

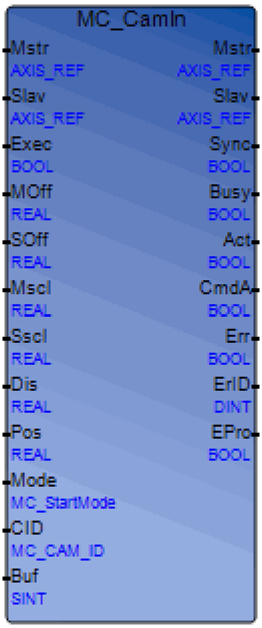
| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | Reference to an axis |
| TimeAccelerationIn | TA | MC_TA_REF | Reference to time / acceleration |
| Execute | Exec | BOOL | Begin motion at rising edge |
| TimeScale | Time | REAL | Time scaling factor of the profile |
| AccelerationScale | Scal | REAL | Scaling factor for the acceleration amplitude |
| Offset | Off | REAL | Offset for the profile |
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | Reference to an axis |
| TimeAcceleration | TA | MC_TA_REF | Reference to time / acceleration |

| | | | |
|---|---|---|---|
| Done | Done | BOOL | Profile completed |
| Busy | Busy | BOOL | Function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | Error occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs a time-acceleration locked motion profile, then goes to zero, maintains the final velocity, and remains in the a state of continuous motion.

# MC_CamIn




Arguments:

| | | | |
|---|---|---|---|
| MasterIn | Mstr | AXIS_REF | References the master axis |
| SlaveIn | Slav | AXIS_REF | References the slave axis |
| Execute | Exec | BOOL | Starts at the rising edge |
| MasterOffset | MOff | REAL | The offset of the master table |
| SlaveOffset | SOff | REAL | The offset of the slave table |
| MasterScaling | Mscl | REAL | Factor by which the master profile is multiplied<br>Default = 1.0 |
| SlaveScaling | Sscl | REAL | Factor by which the slave profile is multiplied<br>Default = 1.0 |

| | | | |
|---|---|---|---|
| MasterStart Distance | Dis | REAL | The position that the master must reach for the slave to begin synchronization |
| MasterSyncPosition | Pos | REAL | Position where the slave is in-sync with the master |
| StartMode | Mode | MC_StartMode | Start mode: absolute, relative, or ramp-in |
| CamTableID | CID | MC_CAM_ID | The identifier of the CAM table used<br>The output of MC_CamTableSelect |
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Master | Mstr | AXIS_REF | Reference to the master axis |
| Slave | Slav | AXIS_REF | Reference to the slave axis |
| InSync | Sync | BOOL | CAM is engaged for the first time |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | The command is aborted by another command |
| Error | Err | BOOL | An error has occurred within the function block |

| | | | |
|---|---|---|---|
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| EndOfProfile | EPro | BOOL | Pulsed output signaling the cyclic end of the CAM profile |

Description:

Engages the CAM

Motion of the master axis is permitted.

When the function block is executed, the actual positions of the master and the slave should correspond to the offset values, or an error may occur.

# MC_CamOut


Arguments:

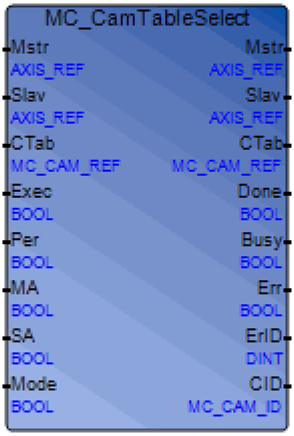
| | | | |
|---|---|---|---|
| SlaveIn | Slav | AXIS_REF | References the slave axis |
| Execute | Exec | BOOL | Disengages the slave axis from the master axis |
| Slave | Slav | AXIS_REF | References the slave axis |
| Done | Done | BOOL | Action is complete |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occurred within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Immediately disconnects the slave axis from the master axis

Another function block usually follows this function block, otherwise the last velocity is maintained as the default condition.

# MC_CamTableSelect




Arguments:

| | | | |
|---|---|---|---|
| MasterIn | Mstr | AXIS_REF | References the master axis |
| SlaveIn | Slav | AXIS_REF | References the slave axis |
| CamTableIn | CTab | MC_CAM_REF | Reference to the CAM description |
| Execute | Exec | BOOL | Begins selection at the rising edge |
| Periodic | Per | BOOL | 1 = periodic<br>0 = non-periodic |
| MasterAbsolute | MA | BOOL | 1 = absolute<br>0 = relative coordinates |
| SlaveAbsolute | SA | BOOL | 1 = absolute<br>0 = relative coordinates |
| MC_ExecutionMode | Mode | BOOL | mcImmediately = the functionality is valid and may influence the motion, i.e. the default behavior, and not the state<br>mcQueued = the functionality is valid when all previous motion commands set one of the following output parameters: Done, Aborted, Error, and Busy is is set to false. |

| | | | |
|---|---|---|---|
| Master | Mstr | AXIS_REF | References the master axis |
| Slave | Slav | AXIS_REF | References the slave axis |
| CamTable | CTab | MC_CAM_REF | CamTable |
| Done | Done | BOOL | Pre-selection is complete |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occurred within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| CamTableID | CID | MC_CAM_ID | Identifies the CAM table to be used in MC_CamIn function block |

Description:

Selects the CAM tables by setting connections to the relevant tables

It is possible to use a virtual axis as the master axis.

When the output parameter *Done* is set, the CamTableID is valid for use in MC_CamIn.

# MC_DigitalCamSwitch


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis to which the switches are connected |
| SwitchesIn | Swch | MC_CAMSWITCH_REF | References the switching actions |
| OutputsIn | Outp | MC_OUTPUT_REF | References signal outputs directly related to the referenced tracks<br>Maximum = 32 per function block<br>First output = first track number |
| TrackOptionsIn | Topt | MC_TRACK_REF | References the structure containing track related properties |
| Enable | En | BOOL | Enables the outputs of switches |
| Enablemask | EnM | DINT | Enables the tracks<br>32 bits of BOOL<br>Lowest track number = least significant data<br>data set to 1 = TRUE, the related track number is enabled |
| Axis | Axis | AXIS_REF | References the axis to which the switches are connected |

| | | | |
|---|---|---|---|
| Switches | Swch | MC_CAMSWITCH_REF | References the switching actions |
| Outputs | Outp | MC_OUTPUT_REF | References signal outputs directly related to the referenced tracks<br>Maximum = 32 per function block<br>First output = first track number |
| TrackOptions | Topt | MC_TRACK_REF | References the structure containing track related properties |
| InOperation | InOp | BOOL | Tracks are enabled |
| Busy | Busy | BOOL | Function block is unfinished |
| Error | Err | BOOL | An error has occurred within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

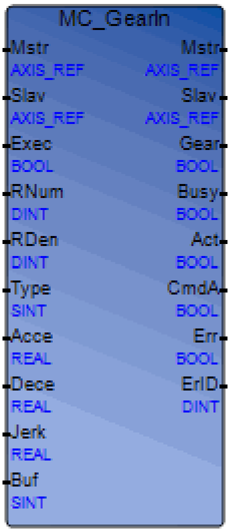
Description:

Commands a group of output bits to change into CAM controlled switches connected to an axis

Forward and backward movements are permitted.

The functionality of this function block is also known as Programmable Limit Switch (PLS).

# MC_GearIn




Arguments:

| | | | |
|---|---|---|---|
| MasterIn | Mstr | AXIS_REF | References the master axis |
| SlaveIn | Slav | AXIS_REF | References the slave axis |
| Execute | Exec | BOOL | Begins the gearing process at the rising edge |
| RatioNumerator | RNum | DINT | The gear ratio numerator |
| RatioDenominator | RDem | DINT | The gear ratio denominator |
| MC_GearInType | Type | SINT | mcCommandedValue = synchronization on command value (0)<br>mcFeedbackValue = synchronization on feedback value (1) |
| Acceleration | Acce | REAL | Acceleration for gearing in |
| Deceleration | Dece | REAL | Deceleration for gearing in |
| Jerk | Jerk | REAL | Jerk of Gearing |

| | | | |
|---|---|---|---|
| BufferMode | Buf | SINT | Defines the behavioral mode of the axis: mcAborting, mcBuffering, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Master | Mstr | AXIS_REF | References the master axis |
| Slave | Slav | AXIS_REF | References the slave axis |
| InGear | Gear | BOOL | Gearing completed |
| Busy | Busy | BOOL | Function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | The command was aborted by another command |
| Error | Err | BOOL | An error has occurred within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Commands a velocity ratio between the slave axis and master axis

The slave increases until the master velocity ratio is reached, then locks. The Gear output is set the first time the gearing ratio is reached.

When MC_GearIn is running, you can modify the gearing ratio using a consecutive MC_GearIn command.

# MC_GearInPos


Arguments:

| | | | |
|---|---|---|---|
| MasterIn | Mstr | AXIS_REF | References the master axis |
| SlaveIn | Slav | AXIS_REF | References the slave axis |
| Execute | Exec | BOOL | Begins the gearing process at the rising edge |
| RatioNumerator | RNum | DINT | The gear ratio numerator |
| RatioDenominator | RDem | DINT | The gear ratio denominator |
| MasterSyncPosition | MSP | REAL | The master position when the axes are running in sync |
| SlaveSyncPosition | SSP | REAL | The slave position when the axes are running in sync |

| | | | |
|---|---|---|---|
| SyncMode | SMod | SINT | Definition of the mode of synchronization |
| MasterStartDistance | MSD | REAL | The master distance to where the slave axis begins synchronization |
| Velocity | Velo | REAL | Maximum velocity during the time interval between StartSync and InSync outputs |
| Acceleration | Acce | REAL | Maximum acceleration during the time interval between StartSync and InSync outputs |
| Deceleration | Dece | REAL | Maximum deceleration during the time interval between StartSync and InSync outputs |
| Jerk | Jerk | REAL | Maximum jerk during the time interval between StartSync and InSync outputs |
| BufferMode | Buf | SINT | Defines the behavioral mode of the axis: mcAborting, mcBuffering, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Master | Mstr | AXIS_REF | References the master axis |
| Slave | Slav | AXIS_REF | References the slave axis |
| StartSync | SrtS | BOOL | Beginning of the commanded gearing |
| InSync | InS | BOOL | Completion of the commanded gearing |
| Busy | Busy | BOOL | Function Block is incomplete |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command was aborted by another command |
| Error | Err | BOOL | An error has occurred within the function block |
| ErrorID | Erld | DINT | Error identification |

Description:

From the synchronization point onward, commands the gear ratio between master axis and slave axis positions

MC_GearInPos maintained previous motion until the master axis reaches the MSD or MSP inputs values, then SrtS output value is set. When a stop command is executed on the slave axis before the synchronization is complete, the synchronization is inhibited and the CmdA output value is generated.

When the MSD value is not provided, MC_GearInPos can calculate the SrtS output value based on the other input values.

# MC_GearOut


Arguments:

| | | | |
|---|---|---|---|
| SlaveIn | Slav | AXIS_REF | References the slave axis |
| Execute | Exec | BOOL | Begins the disconnection process at the rising edge |
| Slave | Slav | AXIS_REF | References the slave axis |
| Done | Done | BOOL | Disconnection is complete |
| Busy | Busy | BOOL | Function Block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Disconnects the slave from the master axis

Another function block usually follows this function block, otherwise the last velocity is maintained as the default condition.

# MC_Halt


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins the stopping action at the rising edge |
| Deceleration | Dece | REAL | Deceleration value = [u/$s^2$] |
| Jerk | Jerk | REAL | Jerk value = [u/$s^3$] |
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |
| Done | Done | BOOL | The velocity has reached zero |
| Busy | Busy | BOOL | Function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs a controlled motion stop

The axis state is DiscreteMotion until the velocity reaches zero. When the Done output is set, the axis state becomes StandStill.

In non-buffering mode, you can abort MC_Halt by setting another motion command during the deceleration of the axis.

To avoid a complete stoppage, you can issue the next command while MC_Halt is running.

# MC_Home


Arguments:

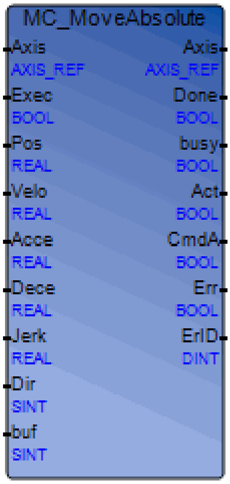
| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins motion at the rising edge |
| Position | Pos | REAL | When the reference signal [u] is detected, absolute position |
| HomingMode | Mode | SINT | HomingMode |
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |
| Done | Done | BOOL | StandStill is reached |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Performs search home sequences

# MC_MoveAbsolute




Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins motion at the rising edge |
| Position | Pos | REAL | Negative or positive target position [u] |
| Velocity | Velo | REAL | Maximum velocity value [u/s] |
| Acceleration | Acce | REAL | Positive acceleration value [$u/s^2$] |
| Deceleration | Dece | REAL | Positive deceleration value [$u/s^2$] |
| Jerk | Jerk | REAL | Positive jerk value [$u/s^3$] |

| | | | |
|---|---|---|---|
| Direction | Dir | SINT | Direction type. Possible values include the following:<br>1 = MC_DirPositive, moving in the positive direction<br>2 = MC_DirShortest, moving in the shortest direction<br>3 = MC_DirNegative, moving in the negative direction<br>4 = MC_DirCurrent, moving in the currect direction |
| Buffermode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |
| Done | Done | BOOL | Position is reached |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Commands controlled motion to a specified absolute position

When MC_MoveAbsolute is complete, the velocity equals zero and not further action occurs.

When there is only one way to reach the desired position, the Dir input value is unused.

For modulo axes, absolute position values are between zero and 360 (360 is excluded). For relative positions, a modulo axes absolute position value of 360 applies.

## MC_MoveAdditive


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins the motion at the rising edge |
| Distance | Dist | REAL | Relative distance [u] of the motion |
| Velicity | Velo | REAL | Maximum velocity value [u/s] |
| Acceleration | Acce | REAL | Acceleration value [u/$s^2$] |
| Deceleration | Dece | REAL | Deceleration value [u/$s^2$] |
| Jerk | Jerk | REAL | Jerk value [u/$s^3$] |
| BufferMode | Buf | SINT | Definition of the behavior mode for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |

| | | | |
|---|---|---|---|
| Done | Done | BOOL | Distance is reached |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Moves a specified distance from the last commanded position

When MC_MoveAdditive is aborted, you can use another MC_MoveAdditive immediately.

# MC_MoveContinuousAbsolute


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins motion at the rising edge |
| Position | Pos | REAL | Negative or positive target position [u] |
| Velocity | Velo | REAL | Maximum velocity value [u/s] |
| EndVelocity | End | REAL | End velocity value [u/s] |
| EndVelocityDirection | Dir | SINT | Direction of the end velocity |
| Acceleration | Acce | REAL | Acceleration value [u/$s^2$] |
| Deceleration | Dece | REAL | Deceleration value [u/$s^2$] |
| Jerk | Jerk | REAL | Jerk value [u/$s^3$] |

| | | | |
|---|---|---|---|
| Direction | Dir | SINT | Direction. Possible values include the following:<br>1 = MC_DirPositive, moving in the positive direction<br>3 = MC_DirNegative, moving in the negative direction<br>4 = MC_DirCurrent, moving in the currect direction |
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |
| InEndVelocity | End | BOOL | Defined distance and velocity reached |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs controlled motion to a specified absolute position, ending with specified velocity

When no motion command exists following MC_MoveContinuousAbsolute, the axis continues to run at the specified velocity and a state of continuous motion persists.

# MC_MoveContinuousRelative


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begin motion at the rising edge |
| Distance | Dist | REAL | Relative distance [u] for the required motion |
| Velocity | Velo | REAL | Maximum velocity value [u/s] |
| EndVelocity | End | REAL | End velocity value [u/s] |
| EndVelocityDirection | Dir | SINT | Direction of the end velocity |
| Acceleration | Acce | REAL | Acceleration value [u/s$^2$] |
| Deceleration | Decc | REAL | Deceleration value [u/s$^2$] |
| Jerk | Jerk | REAL | Jerk value [u/s$^3$] |

| | | | |
|---|---|---|---|
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |
| InEndVelocity | End | BOOL | Defined distance and velocity reached |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs controlled motion to a specified relative distance ending with specified velocity

When no motion command exists following MC_MoveContinuousRelative, the axis continues to run at the specified velocity and a state of continuous motion persists.

# MC_MoveRelative


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begin motion at the rising edge |
| Distance | Dist | REAL | Relative distance [u] for the required motion |
| Velocity | Velo | REAL | Maximum velocity value [u/s] |
| Acceleration | Acce | REAL | Acceleration value [u/s$^2$] |
| Deceleration | Dece | REAL | Deceleration value [u/s$^2$] |
| Jerk | Jerk | REAL | Jerk value [u/s$^3$] |
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |

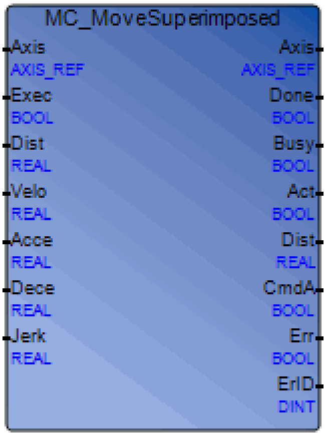
| | | | |
|---|---|---|---|
| Done | Done | BOOL | Distance is reached |
| Busy | Busy | BOOL | Function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| Commandaborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs movement to a specified distance relative to the actual position at the time of execution

When no commands are placed following MC_MoveRelative, the axis velocity value of zero is maintained.

# MC_MoveSuperimposed




Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begin motion at the rising edge |
| Distance | Dist | REAL | Additional distance [u] to be superimposed |
| VelocityDiff | Velo | REAL | Velocity difference value [u/s] for the additional motion |
| Acceleration | Acce | REAL | Acceleration value [u/s$^2$] |
| Deceleration | Dece | REAL | Deceleration value [u/s$^2$] |
| Jerk | Jerk | REAL | Jerk value [u/s$^3$] |
| Axis | Axis | AXIS_REF | References the axis |
| Done | Done | BOOL | Additional distance is superimposed on the ongoing motion |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CoveredDistance | Dist | REAL | Distance covered since starting |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs controlled uninterrupted motion to a specified relative distance additional to the existing motion.

When MC_MoveSuperimposed is running, other existing commands that are in abort mode can cause MC_MoveSuperimposed and the associated motion commands to abort. When the other existing commands are in a mode other then abort, MC_MoveSuperimposed is aborted and the underlying motion command is maintained.

When add MC_MoveSuperimposed to an active MC_MoveSuperimposed, the running MC_MoveSuperimposed is aborted then replaced and the underlying motion command is maintained.

MC_MoveSuperimposed causes a change in velocity and position of ongoing motion in all states. In StandStill state, MC_MoveSuperimposed performs the same as MC_MoveRelative.

The Acce, Dece, and Jerk input values are additional to the ongoing motion. Regardless of a concurrent MC_MoveSuperimposed, running motion commands finish within the specified time period.

# MC_MoveVelocity


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begin motion at the rising edge |
| Velocity | Velo | REAL | Maximum velocity value [u/s] |
| Acceleration | Acce | REAL | Acceleration value [u/s$^2$] |
| Deceleration | Dece | REAL | Deceleration value [u/s$^2$] |
| Jerk | Jerk | REAL | Jerk value [u/s$^3$] |
| Directionin | Dir | SINT | Direction. Possible directions are positive, negative, and current |
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |

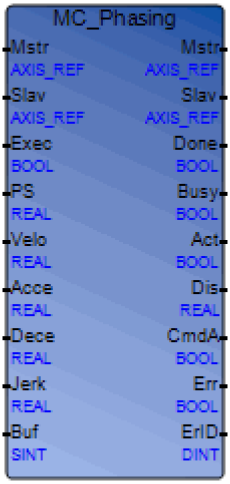
| | | | |
|---|---|---|---|
| Axis | Axis | AXIS_REF | References the axis |
| InVelocity | InVe | BOOL | The required velocity is reached |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| Direction | Dir | SINT | Direction. Possible directions are positive, negative, and current |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs continuous motion at a specified velocity

You can stop MC_MoveVelocity by interrupting using another function block. When MC_MoveVelocity is aborted, reset the InVe output value.

# MC_Phasing




Arguments:

| | | | |
|---|---|---|---|
| MasterIn | Mstr | AXIS_REF | References the master axis |
| SlaveIn | Slav | AXIS_REF | References the slave axis |
| Execute | Exec | BOOL | Begins phasing process at the rising edge |
| PhaseShift | PS | REAL | Phase difference [u] |
| Velocity | Velo | REAL | Maximum velocity value [u/s] |
| Acceleration | Acce | REAL | Acceleration value [u/$s^2$] |
| Deceleration | Dece | REAL | Deceleration value [u/$s^2$] |
| Jerk | Jerk | REAL | Jerk value [u/$s^3$] |

| | | | |
|---|---|---|---|
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Master | Mstr | AXIS_REF | References the master axis |
| Slave | Slav | AXIS_REF | References the slave axis |
| Done | Done | BOOL | Required phase difference is reached |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CoveredDistance | Dis | REAL | Distance covered since starting |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Creates a phase shift effecting the position of the slave axis

MC_Phasing is used to delay or advance the slave axis in relation to its master.

MC_Phasing controls five inputs: PS, Velo, Acce, Dece, and Jerk.

## MC_PositionProfile


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| TimePositionIn | TPos | MC_TP_REF | Reference to time / position |
| Execute | Exec | BOOL | Begins the motion at the rising edge |
| TimeScale | Time | REAL | Time scaling factor [t.u.] for the profile |
| PositionScale | Pos | REAL | Position scale factor [t.u.] |
| Offset | Off | REAL | Offset [u] for the profile |
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending.<br>Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |
| TimePosition | TPos | MC_TP_REF | Reference to time / position |

| | | | |
|---|---|---|---|
| Done | Done | BOOL | The profile is complete |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs a time-position locked motion profile

## MC_Power


Arguments:

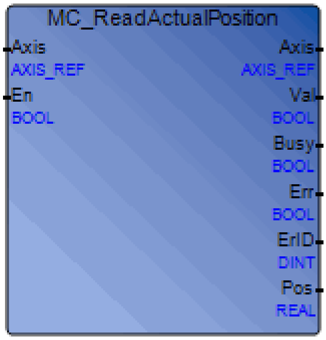
| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Enable | En | BOOL | TRUE = power is ON<br>FALSE = power is OFF |
| Enable_Positive | EnPO | BOOL | TRUE = motion in the positive direction only<br>FALSE = no motion in the positive direction |
| Enable_Negative | EnNE | BOOL | TRUE = motion in the negative direction only<br>FALSE = no motion in the negative direction |
| Axis | Axis | AXIS_REF | References the axis |
| Status | Stat | BOOL | Status of the power stage |
| Valid | Val | BOOL | Valid |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs the power stages to turn *On* and *Off*

# MC_ReadActualPosition




Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Enable | En | BOOL | When enabled, yields the parameter value continuously |
| Axis | Axis | AXIS_REF | References the axis |
| Valid | Val | BOOL | The value is available |
| Busy | Busy | BOOL | The function block is unfinished and output values are expected |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| Position | Pos | REAL | Absolute position [u] |

Description:

Yields the actual position

# MC_ReadActualTorque


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Enable | En | BOOL | When enabled, yields the parameter value continuously |
| Axis | Axis | AXIS_REF | References the axis |
| Valid | Val | BOOL | A valid value is available |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| ActualTorque | Torq | REAL | Actual torque value in technical units |

Description:

Yields the actual torque value

You can have a signed value for the Torq output value.

# MC_ReadActualVelocity


Arguments:

| | | | |
|---|---|---|---|
| Axis | Axis | AXIS_REF | References the axis |
| Enable | En | BOOL | When enabled, yields the parameter value continuously |
| Axis | Axis | AXIS_REF | References the axis |
| Valid | Val | BOOL | Valid value is available |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| ActualVelocity | Velo | REAL | Axis error value |

Description:

Yields the actual velocity value

When the En input is reset, the data becomes invalid and all outputs are reset.

When the Velo output is valid, the Val output is true.

# MC_ReadAxisError

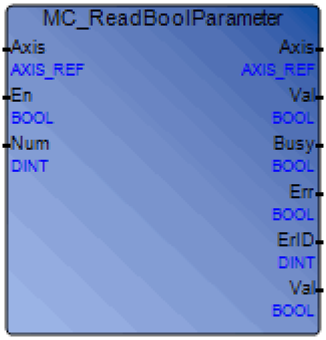

Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Enable | En | BOOL | When enabled, yields the parameter value continuously |
| Axis | Axis | AXIS_REF | References the axis |
| Valid | Val | BOOL | Valid value is available |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| AxisErrorId | AxEr | DINT | Axis error value |

Description:

Yields general axis errors that are not related to function blocks

# MC_ReadBoolParameter




Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Enable | En | BOOL | When enabled, yields the parameter value continuously |
| ParameterNumber | Num | DINT | Number of the parameter |
| Axis | Axis | AXIS_REF | References the axis |
| Valid | Val | BOOL | Valid parameter value is available |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |

| | | | |
|---|---|---|---|
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| Value | Val | BOOL | Value of the parameter |

Description:

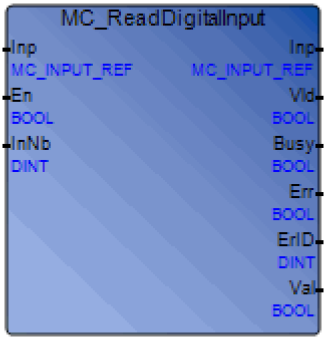
Yields the value of a specific BOOL parameter

List of Parameters:

| Number | Parameter Name | Data Type | Description |
|---|---|---|---|
| 1 | CommandedPosition | REAL | Commanded position |
| 2 | SWLimitPos | REAL | Positive software limit switch position |
| 3 | SWLimitNeg | REAL | Negative software limit switch position |
| 4 | EnableLimitPos | BOOL | Enable positive software limit switch |
| 5 | EnableLimitNeg | BOOL | Enable negative software limit switch |
| 6 | EnablePosLagMonitoring | BOOL | Enable monitoring of position lag |
| 7 | MaxPositionLag | BOOL | Maximal position lag |
| 8 | MaxVelocitySystem | REAL | Maximal allowed velocity of the axis in the motion system |
| 9 | MaxVelocityAppl | REAL | Maximal allowed velocity of the axis in the application |
| 10 | ActualVelocity | REAL | Actual velocity |

| | | | |
|---|---|---|---|
| 11 | CommandedVelocity | | Commanded set point velocity READ only |
| 12 | MaxAccelerationSystem | REAL | Maximal allowed acceleration of the axis in the motion system |
| 13 | MaxAccelerationAppl | REAL | Maximal allowed acceleration of the axis in the application |
| 14 | MaxDecelerationSystem | REAL | Maximal allowed deceleration of the axis |
| 15 | MaxDecelerationAppl | REAL | Maximal allowed deceleration of the axis |
| 16 | MaxJerk | REAL | Maximal allowed jerk of the axis |

# MC_ReadDigitalInput




Arguments:

| | | | |
|---|---|---|---|
| Inp | Inp | MC_INPUT_REF | References the source of the input signal |
| Enable | En | BOOL | When enabled, yields the parameter value continuously |
| InputNumber | InNb | DINT | Selects the input |
| Input | Inp | MC_INPUT_REF | References the source of the input signal |
| Valid | Vld | BOOL | Valid parameter value is available |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |

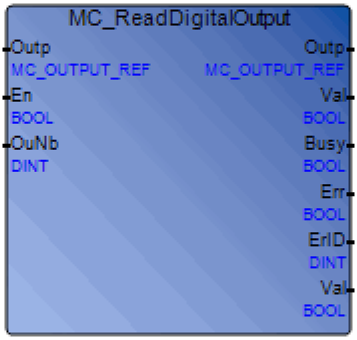
| | | | |
|---|---|---|---|
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| Value | Val | BOOL | Value of the parameter |

Description:

Yields the value of a specific input

**Note:** When a pulse signal ends before the next function block cycle begins, the signal is undetected.

# MC_ReadDigitalOutput




Arguments:

| | | | |
|---|---|---|---|
| Outp | Outp | MC_OUTPUT_REF | References signal outputs |
| Enable | En | BOOL | When enabled, yields the parameter value continuously |
| OutputNumber | OuNb | DINT | Selects the output |
| Output | Outp | MC_OUTPUT_REF | References signal outputs |
| Valid | Val | BOOL | Valid output signal value is available |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |

| | | | |
|---|---|---|---|
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| Value | Val | BOOL | Value of the output signal |

Description:

Yields the value of a specific output

**Note:** When a pulse signal ends before the next function block cycle begins, the signal is undetected.

# MC_ReadParameter


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Enable | En | BOOL | When enabled, yields the parameter value continuously |
| ParameterNumber | Num | DINT | Number of the parameter |
| Axis | Axis | AXIS_REF | References the axis |
| Valid | Val | BOOL | Valid parameter value is available |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |

| | | | |
|---|---|---|---|
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| Value | Val | REAL | Value of the parameter |

Description:

Yields the value of a specific parameter

List of Parameters:

| Number | Parameter Name | Data Type | Description |
|---|---|---|---|
| 1 | CommandedPosition | REAL | Commanded position |
| 2 | SWLimitPos | REAL | Positive software limit switch position |
| 3 | SWLimitNeg | REAL | Negative software limit switch position |
| 4 | EnableLimitPos | BOOL | Enable positive software limit switch |
| 5 | EnableLimitNeg | BOOL | Enable negative software limit switch |
| 6 | EnablePosLagMonitoring | BOOL | Enable monitoring of position lag |
| 7 | MaxPositionLag | BOOL | Maximal position lag |
| 8 | MaxVelocitySystem | REAL | Maximal allowed velocity of the axis in the motion system |
| 9 | MaxVelocityAppl | REAL | Maximal allowed velocity of the axis in the application |
| 10 | ActualVelocity | REAL | Actual velocity |

| | | | |
|---|---|---|---|
| 11 | CommandedVelocity | | Commanded set point velocity READ only |
| 12 | MaxAccelerationSystem | REAL | Maximal allowed acceleration of the axis in the motion system |
| 13 | MaxAccelerationAppl | REAL | Maximal allowed acceleration of the axis in the application |
| 14 | MaxDecelerationSystem | REAL | Maximal allowed deceleration of the axis |
| 15 | MaxDecelerationAppl | REAL | Maximal allowed deceleration of the axis |
| 16 | MaxJerk | REAL | Maximal allowed jerk of the axis |

## MC_ReadStatus


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Enable | En | BOOL | When enabled, yields the parameter value continuously |
| Axis | Axis | AXIS_REF | References the axis |

| | | | |
|---|---|---|---|
| Valid | Val | REAL | True = valid output available |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| Errorstop | Stop | BOOL | ErrorStop |
| Disabled | Dis | BOOL | Set when axis is in disabled state |
| Stopping | Stop | BOOL | Stopping state |
| Referenced | Ref | BOOL | Absolute reference position of the axis |
| StandStill | Stan | BOOL | Standstill state |
| DiscreteMotion | DM | BOOL | DiscreteMotion state |
| ContinuousMotion | CM | BOOL | ContinuousMotion state |
| SynchronizedMotion | SM | BOOL | SynchronizedMotion state |

| | | | |
|---|---|---|---|
| Homing | Home | BOOL | Homing state. Possible values include the following:<br>0 = MC_AbsSwitch, absolute switch homing plus limit switches<br>1 = MC_LimitSwitch, homing against limit switches<br>2 = MC_RefPulse, homing using encoder reference pulse "zero mark"<br>3 = MC_Direct, static homing forcing position from user reference<br>4 = MC_Absolute, static homing forcing position from absolute encoder<br>5 = MC_Block, homing against hardware parts blocking movement |
| ConstantVelocity | CV | BOOL | ConstantVelocity state |
| Accelerating | Acce | BOOL | Acceleration value |
| Decelerating | Dece | BOOL | Deceleration value |

Description:

Yields the detailed status of an axis that is in motion

# MC_Reset


Arguments:

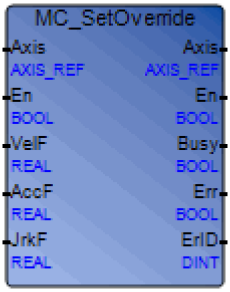
| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins resetting the axis at the rising edge |
| Axis | Axis | AXIS_REF | References the axis |
| Done | Done | BOOL | The Standstill state is reached |
| Busy | Busy | BOOL | The function block is unfinished |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Transitions axis from the ErrorStop state to the StandStill state by resetting all axis-related internal errors

The outputs of function block instances are unaffected by MC_Rest.

## MC_SetOverride




Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Enable | En | BOOL | When enabled, yields the override factor value continuously |
| VelFactor | VelF | REAL | Override factor for velocity = [0.0 .. 1.0]<br>Default = 1.0<br>A value of 0.0 stops the axis, StandStill state is unachieved |
| AccFactor | AccF | REAL | Override factor for acceleration/deceleration = [0.0 .. 1.0]<br>Default = 1.0<br>A value of 0.0 is not permitted |
| JerkFactor | JrkF | REAL | Override factor for jerk = [0.0 .. 1.0]<br>Default = 1.0<br>A value of 0.0 is not permitted |
| Axis | Axis | AXIS_REF | References the axis |
| Enabled | En | BOOL | The override factors are set successfully |
| Busy | Busy | BOOL | The function block is unfinished |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Specifies the override values for the axis

The override parameters act as factors, which the commanded velocity, acceleration, deceleration, and jerk are multiplied by in order to move the function block.

MC_SetOverride works on master axes only, axes in the SyncronizedMotion state, such as slave axes, are unaffected.

The VelF input is modifiable at all times and acts directly on the ongoing motion.

When in the DiscreteMotion state, reducing the input values for AccF and JrkF may lead to position overshoot.

# MC_SetPosition


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins setting position |
| Position | Pos | REAL | Position unit [u]<br>Requires that input Rel is set to TRUE |
| Relative | Rel | BOOL | TRUE = relative distance<br>FALSE = absolute position<br>Default value = FALSE |
| MC_ExecutionMode | Mode | BOOL | Motion control execution mode<br>mcImmediately = functionality immediately valid and affects ongoing motion<br>mcQueued = functionality valid when all other motion commands have one of the following output parameters set: Done, Aborted, Error, or Busy is set to FALSE<br>Default value = mcImmediately |
| Axis | Axis | AXIS_REF | References the axis |
| Done | Done | BOOL | New Pos value available |
| Busy | Busy | BOOL | The function block is unfinished |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Shifts the position of an axis by modifying the set-point position and the actual position of the axis

When the Rel input is set to TRUE, the relative distance is added to the actual position value at the time of execution. When the Rel output is set to FALSE, the actual position is set to the value of the Pos input.

# MC_Stop


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins the stopping action at the rising edge |
| Deceleration | Dece | REAL | Deceleration value = [$u/s^2$] |
| Jerk | Jerk | REAL | Jerk value = [$u/s^3$] |
| Axis | Axis | AXIS_REF | References the axis |
| Done | Done | BOOL | Velocity has reached zero |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs a controlled motion stop and transferees the axis to the Stopping state

While the axis is in the Stopping state, the axis is unavailable for use with other function blocks and all running function blocks are aborted. The axis remains in the Stopping state while the Exec input is set to TRUE and the velocity of the axis is above zero. When the velocity reaches zero, the Done output is set to TRUE, the Exec output is set to FALSE, and the axis achieves the StandStill state.

# MC_TorqueControl


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begin torque action at rising edge |
| Torque | Torq | REAL | Torque value in t.u. |
| TorqueRamp | TRmp | REAL | Maximum time derivative value in t.u./sec |
| Velocity | Velo | REAL | Absolute value of the maximum velocity |
| Acceleration | Acce | REAL | Maximum acceleration value |
| Deceleration | Dece | REAL | Maximum deceleration value |
| Jerk | Jerk | REAL | Maximum jerk value |
| Direction | Dir | SINT | Motion control direction, either positive or negative |

| | | | |
|---|---|---|---|
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |
| InTorque | InT | BOOL | Torque setpoint value is reached |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs continuous exertion of torque of a specific magnitude

Torque increases according to the TRmp input value. When the specified torque is reached, InT output is set.

MC_TorqueControl applies to both torque and force. When there is no external load, force applies instead of torque.

## MC_TouchProbe


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis for which the position is recorded for a defined Trig input event |
| TiggerInp | Trig | MC_TRIGGER_REF | References the source of the trigger signal |
| Execute | Exec | BOOL | Begins touch probe recording at the rising edge |
| WindowOnly | WinO | BOOL | When set, only values within a specific window can trigger events |
| FirstPosition | FPos | REAL | The start position [u] where trigger events are accepted<br>The start position value is included in the range of window values |
| LastPosition | LPos | REAL | The stop position [u] where trigger events are accepted<br>The stop position value is included in the range of window values |

| | | | |
|---|---|---|---|
| Axis | Axis | AXIS_REF | References the axis for which the position is recorded for a defined Trig input event |
| TriggerInput | Trig | MC_TRIGGER_REF | References the source of the trigger signal |
| Done | Done | BOOL | The trigger event is recorded |
| Busy | Busy | BOOL | The function block is unfinished |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |
| RecordedPosition | RPos | REAL | Position [u] where the trigger event occured |

Description:

Records the position of an axis during a trigger event

The first trigger event is recorded and subsequent trigger events are disregarded.

## MC_VelocityProfile


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| TimeVelocityIn | TV | MC_TV_REF | References Time / Velocity<br>Time can equal the difference in time between two points |
| Execute | Exec | BOOL | Begins the motion at the rising edge |
| TimeScale | Time | REAL | Time scaling factor for the profile |
| VelocityScale | Scal | REAL | Velocity scaling factor for the profile |
| Offset | Off | REAL | Offset factor for the profile |
| BufferMode | Buf | SINT | Definition of the mode of behavior for the axis: mcAborting, mcBuffered, mcBlending. Possible values are the following:<br>0 = mcAborting, the next function block is taking control of the axis immediately<br>1 = mcBuffered, the next function block awaits completion (DONE) before taking control of the axis<br>2 = mcBlending |
| Axis | Axis | AXIS_REF | References the axis |

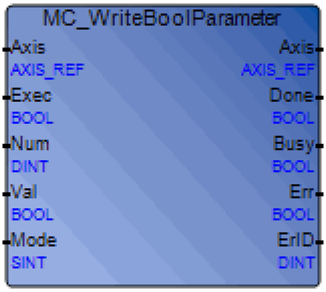
| | | | |
|---|---|---|---|
| TimeVelocity | TV | MC_TV_REF | References Time / Velocity<br>Time can equal the difference in time between two points |
| Done | Done | BOOL | The profile is complete |
| Busy | Busy | BOOL | The function block is unfinished |
| Active | Act | BOOL | Function Block is actively controlling the axis |
| CommandAborted | CmdA | BOOL | Command is aborted by another command |
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Directs a time-velocity locked motion profile

When the Done output is set, the final velocity is maintained and the axis remains in the ContinuousMotion state.

# MC_WriteBoolParameter




Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins writing the parameter value at the rising edge |
| ParameterNumber | Num | DINT | Number of the parameter |
| Value | Val | BOOL | Value of the parameter |
| MC_ExecutionMode | Mode | SINT | Motion control execution mode<br>mcImmediately = functionality immediately valid and affects ongoing motion<br>mcQueued = functionality valid when all other motion commands have one of the following output parameters set: Done, Aborted, Error, or Busy is set to FALSE<br>Default value = mcImmediately |
| Axis | Axis | AXIS_REF | References the axis |
| Done | Done | BOOL | Parameter is successfully written |
| Busy | Busy | BOOL | The function block is unfinished |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

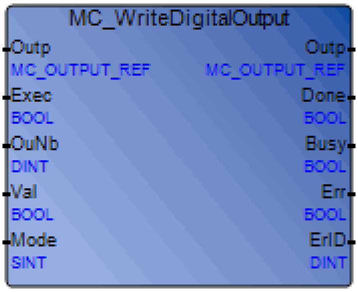
Modifies the value of a specific BOOL parameter

List of Parameters:

| Number | Parameter Name | Data Type | Description |
|---|---|---|---|
| 1 | CommandedPosition | REAL | Commanded position |
| 2 | SWLimitPos | REAL | Positive software limit switch position |
| 3 | SWLimitNeg | REAL | Negative software limit switch position |
| 4 | EnableLimitPos | BOOL | Enable positive software limit switch |
| 5 | EnableLimitNeg | BOOL | Enable negative software limit switch |
| 6 | EnablePosLagMonitoring | BOOL | Enable monitoring of position lag |
| 7 | MaxPositionLag | BOOL | Maximal position lag |
| 8 | MaxVelocitySystem | REAL | Maximal allowed velocity of the axis in the motion system |
| 9 | MaxVelocityAppl | REAL | Maximal allowed velocity of the axis in the application |
| 10 | ActualVelocity | REAL | Actual velocity |

| | | | |
|---|---|---|---|
| 11 | CommandedVelocity | | Commanded set point velocity READ only |
| 12 | MaxAccelerationSystem | REAL | Maximal allowed acceleration of the axis in the motion system |
| 13 | MaxAccelerationAppl | REAL | Maximal allowed acceleration of the axis in the application |
| 14 | MaxDecelerationSystem | REAL | Maximal allowed deceleration of the axis |
| 15 | MaxDecelerationAppl | REAL | Maximal allowed deceleration of the axis |
| 16 | MaxJerk | REAL | Maximal allowed jerk of the axis |

# MC_WriteDigitalOutput




Arguments:

| | | | |
|---|---|---|---|
| Outp | Outp | MC_OUTPUT_REF | References the signal output |
| Execute | Exec | BOOL | Writes the value of the selected output |
| OutputNumber | OuNb | DINT | Selects the output by number |
| Value | Val | BOOL | Value of the selected parameter |
| MC_ExecutionMode | Mode | SINT | Motion control execution mode<br>mcImmediately = functionality immediately valid and affects ongoing motion<br>mcQueued = functionality valid when all other motion commands have one of the following output parameters set: Done, Aborted, Error, or Busy is set to FALSE<br>Default value = mcImmediately |
| Output | Outp | MC_OUTPUT_REF | References the signal output |
| Done | Done | BOOL | Output signal value successfully written |
| Busy | Busy | BOOL | The function block is unfinished |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Modifies a specific output value

# MC_WriteParameter


Arguments:

| | | | |
|---|---|---|---|
| AxisIn | Axis | AXIS_REF | References the axis |
| Execute | Exec | BOOL | Begins writing the parameter value at the rising edge |
| ParameterNumber | Num | DINT | Number of the parameter |
| Value | Val | BOOL | Value of the parameter |
| MC_ExecutionMode | Mode | SINT | MC_ExecutionMode |
| Axis | Axis | AXIS_REF | References the axis |
| Done | Done | BOOL | Parameter written successfully |
| Busy | Busy | BOOL | TRUE = function block is actively controlling the axis<br>FALSE = axis is not actively controlled by the function block |

| | | | |
|---|---|---|---|
| Error | Err | BOOL | An error has occured within the function block |
| ErrorID | ErID | DINT | Error identification. Possible values include the following:<br>1 = MC_ErrState, bad state<br>2 = MC_ErrRange, bad parameter (value is out of range)<br>3 = MC_ErrParam, bad parameter (negative value)<br>4 = MC_ErrFBInvalid, function block is not implemented<br>5 = MC_ErrAxisNo, axis number is invalid<br>6 = MC_ErrDrive, error from drive<br>7 = MC_ErrAborted, command aborted<br>8 = MC_ErrNoPower, mc_power is not running |

Description:

Modifies the value of a specific parameter

List of Parameters:

| Number | Parameter Name | Data Type | Description |
|---|---|---|---|
| 1 | CommandedPosition | REAL | Commanded position |
| 2 | SWLimitPos | REAL | Positive software limit switch position |
| 3 | SWLimitNeg | REAL | Negative software limit switch position |
| 4 | EnableLimitPos | BOOL | Enable positive software limit switch |
| 5 | EnableLimitNeg | BOOL | Enable negative software limit switch |
| 6 | EnablePosLagMonitoring | BOOL | Enable monitoring of position lag |
| 7 | MaxPositionLag | BOOL | Maximal position lag |
| 8 | MaxVelocitySystem | REAL | Maximal allowed velocity of the axis in the motion system |
| 9 | MaxVelocityAppl | REAL | Maximal allowed velocity of the axis in the application |
| 10 | ActualVelocity | REAL | Actual velocity |

| | | | |
|---|---|---|---|
| 11 | CommandedVelocity | | Commanded set point velocity READ only |
| 12 | MaxAccelerationSystem | REAL | Maximal allowed acceleration of the axis in the motion system |
| 13 | MaxAccelerationAppl | REAL | Maximal allowed acceleration of the axis in the application |
| 14 | MaxDecelerationSystem | REAL | Maximal allowed deceleration of the axis |
| 15 | MaxDecelerationAppl | REAL | Maximal allowed deceleration of the axis |
| 16 | MaxJerk | REAL | Maximal allowed jerk of the axis |

# SAMA Elements and Functions

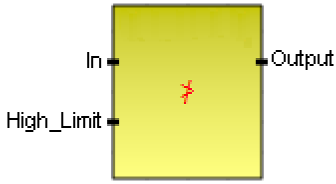
The workbench supports the following SAMA elements:

| | | |
|---|---|---|
| **Process Control** | High Limiting | Chooses the lowest value, either the input or the High_Limit |
| | Integrate or Totalize | Determines the time integral of the input value |
| | Low Limiting | Chooses the higher value, either the input or the Low_Limit |
| | MATransfer | The output value is determined by the CMD value. When CMD has a value of Logic One the Output_signal value is equal to the InputMan value. When CMD has a value of Logic Zero the Output_signal value is equal to the InputAuto value. |
| | MATransferSet | The Set_Point_out value is equal to the SetPoint value. The Output_Signal value is determined by the CMD value. When CMD has a value of Logic One the Output_signal value is equal to the InputMan value. When CMD has a value of Logic Zero the Output_signal value is equal to the InputAuto value. |
| | Memory (Basic) | When one input has a value of Logic One, the output value is Logic One. |
| | Memory (So Dominant) | When multiple inputs have a value of Logic One, only the output with an override designation ($S_o$) is Logic One. |
| | Memory (Ro Dominant) | When multiple inputs have a value of Logic One, only the output with an override designation ($S_o$) is Logic One. |
| | Proportional | Output value is directly proportional to the input value |

| | | |
|---|---|---|
| | Proportional and Integral | Output value is directly proportional to both the magnitude and duration of the input value |
| | Proportional and Derivative | Output value is directly proportional to the rate of change of the input value. |
| | Reverse Proportional | Output value is inversely proportional to the input value. |
| | Tri-State Signal | Output has discrete states dependent on the state of the input. |
| | Velocity Limiting | When the rate of change of input is below High_Limit, output is equal to input |
| **Time Operations** | Pulse Duration | When input is Logic One, the output is Logic One for a specific time period only |
| | Pulse Duration Of The Lesser Time | When input has been Logic One for a specific time period, output changes to a Logic Zero |
| | Time Delay On Initiation | When input has been Logic One continuously for a specific time period, output becomes a Logic One |
| | Time Delay On Termination | When input equals a Logic Zero for a specific time period, output becomes Logic Zero |

# High Limiting

SAMA Representation:




FBD Representation:


Arguments:

| | | |
|---|---|---|
| In | IN | REAL |
| High_Limit | HL | REAL |
| Output | Q | REAL |

Description:

The output value is either the input value or the High_Limit value, whichever is the lowest.

**To insert a High Limiting element**

- From the Toolbox, drag the **High Limiting** element into the language container.

The High Limiting element is displayed in the language container in SAMA format.

**To insert a HighLimit function block**

1. From the Toolbox, drag the block element into the language container.

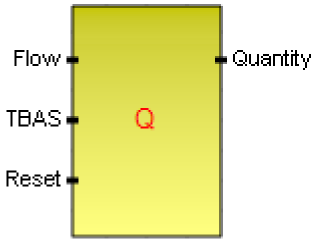
   The Block Selector is displayed.

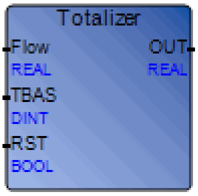
2. In the Block Selector, select **HighLimit**, then click **OK**.

The High Limiting element is displayed in the language container in SAMA format.

# Integrate or Totalize

SAMA Representation: FBD Representation:




Arguments:

| | |
|---|---|
| Flow | REAL |
| TBAS | DINT |
| Reset | BOOL |
| Quantity | REAL |

Description:

The output value is a frequency that depends on the input value. The output is usually associated with a counting device, which displays the time integral of the input value with some initial condition applied at time (T) equals zero.

**To insert an Integrate or a Totalize element**

- From the Toolbox, drag the **Integrate or Totalize** element into the language container.

The Integrate or Totalize element is displayed in the language container in SAMA format.

**To insert a Totalizer function block**

1. From the Toolbox, drag the block element into the language container.

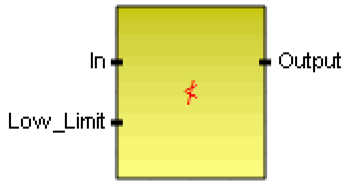
   The Block Selector is displayed.

2. In the Block Selector, select **Totalizer**, then click **OK**.

The Integrate or Totalize element is displayed in the language container in SAMA format.

# Low Limiting

SAMA Representation:




FBD Representation:


Arguments:

| | | |
|---|---|---|
| In | IN | REAL |
| Low_Limit | LL | REAL |
| Output | Q | REAL |

Description:

The output value is either the input value or the low limit value, whichever is the highest.

**To insert a Low Limiting element**

- From the Toolbox, drag the **Low Limiting** element into the language container.

The Low Limiting element is displayed in the language container in SAMA format.

**To insert a LowLimit function block**

1. From the Toolbox, drag the block element into the language container.

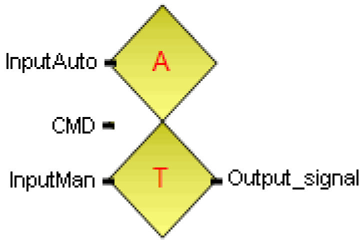
   The Block Selector is displayed.

2. In the Block Selector, select **LowLimit**, then click **OK**.

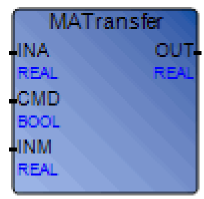
The Low Limiting element is displayed in the language container in SAMA format.

# MATransfer

SAMA Representation:




FBD Representation:




Arguments:

| | | | |
|---|---|---|---|
| InputAuto | INA | REAL | Automatic input |
| Command | CMD | BOOL | Indication of which signal to select<br>True selects InputAuto<br>False selects InputMan |
| InputMan | INM | REAL | Manual input |
| Output_signal | Out | REAL | Output signal |

Description:

The Output_signal value of MATransfer is determined by the CMD value. When CMD has a value of Logic One (TRUE) the Output_signal value is equal to the InputMan value. When CMD has a value of Logic Zero (FALSE) the Output_signal value is equal to the InputAuto value.

**To insert an MATransfer element**

- From the Toolbox, drag the **MATransfer** element into the language container.

The MATransfer element is displayed in the language container in SAMA format.

**To insert an MATransfer function block**

1. From the Toolbox, drag the block element into the language container.

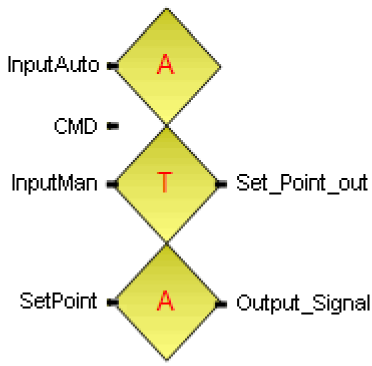
   The Block Selector is displayed.

2. In the Block Selector, select **MATransfer**, then click **OK**.

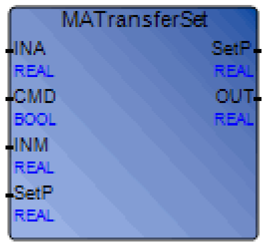
The MATransfer element is displayed in the language container in SAMA format.

# MATransferSet

SAMA Representation:




FBD Representation:




Arguments:

| | | | |
|---|---|---|---|
| InputAuto | INA | REAL | Automatic input |
| CMD | CMD | BOOL | Indication of which signal to select<br>True selects InputAuto<br>False selects InputMan |
| InputMan | INM | REAL | Manual input |
| SetPoint | SetP | REAL | Set point value |
| Set_Point_out | SetP | REAL | Set point for loop control<br>Local and forced by user |
| Output_Signal | Out | REAL | Output signal |

Description:

The Set_Point_out value of MATransferSet is equal to the SetPoint value. The Output_Signal value is determined by the CMD value. When CMD has a value of Logic One (TRUE) the Output_signal value is equal to the InputMan value. When CMD has a value of Logic Zero (FALSE) the Output_signal value is equal to the InputAuto value.

**To insert an MATransferSet element**

- From the Toolbox, drag the **MATransferSet** element into the language container.

The MATransferSet element is displayed in the language container in SAMA format.

**To insert an MATransferSet function block**

1. From the Toolbox, drag the block element into the language container.

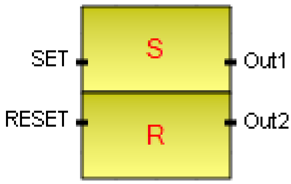
   The Block Selector is displayed.

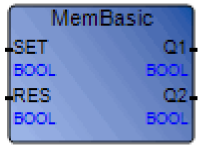
2. In the Block Selector, select **MATransferSet**, then click **OK**.

The MATransferSet element is displayed in the language container in SAMA format.

# Memory (Basic)

SAMA Representation: FBD Representation:




Arguments:

| | | |
|---|---|---|
| SET | SET | BOOL |
| RESET | RES | BOOL |
| Out1 | Q1 | BOOL |
| Out2 | Q2 | BOOL |

Description:

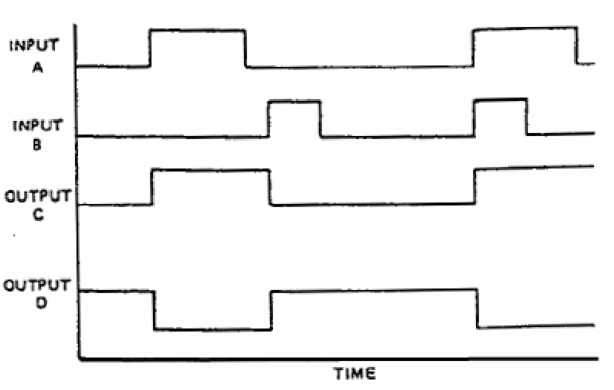
When one input has a value of Logic One, the output value is Logic One. If the input value is subsequently lost (Logic Zero), the associated output value is memorized (retained at Logic One). Connecting a Logic One output to an input gives the same value (Logic One) and changes the output states.

Mathematical equation:

| A | B | C | D |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 |
| **1* | *1* | *0* | *0* |

**Note:** When A and B are simultaneously true, the output condition changes from the last state.

Graphic representation:




**To insert a Memory (Basic) element**

- From the Toolbox, drag the **Memory (Basic)** element into the language container.

The Memory (Basic) element is displayed in the language container in SAMA format.

**To insert a MemBasic function block**

1. From the Toolbox, drag the block element into the language container.

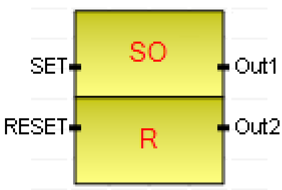
   The Block Selector is displayed.

2. In the Block Selector, select **MemBasic**, then click **OK**.

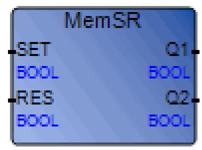
The Memory (Basic) element is displayed in the language container in SAMA format.

# Memory ($S_o$ Dominant)

SAMA Representation:




FBD Representation:




Arguments:

| | | |
|---|---|---|
| SET | SET | BOOL |
| RESET | RES | BOOL |
| Out1 | Q1 | BOOL |
| Out2 | Q2 | BOOL |

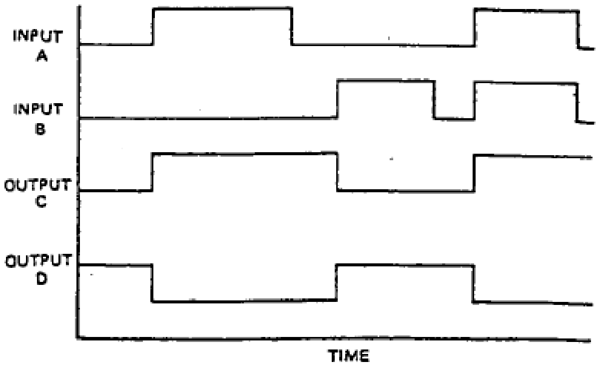
Description:

When one input has a value of Logic One, the output value is Logic One. If the input value is subsequently lost (Logic Zero), the associated output value is memorized (retained at Logic One). Connecting a Logic One output to an input gives the same value (Logic One) and changes the output states. When multiple inputs have a value of Logic One, only the output with an override designation ($S_o$) is Logic One.

Mathematical equation:

| A | B | C | D |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

Graphic representation:




**To insert a Memory (So Dominant) element**

- From the Toolbox, drag the **Memory (So Dominant)** element into the language container.

The Memory (So Dominant) element is displayed in the language container in SAMA format.

**To insert a MemSR function block**

1. From the Toolbox, drag the block element into the language container.

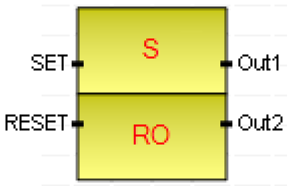
   The Block Selector is displayed.

2. From the Block Selector, select **MemSR**, then click **OK**.

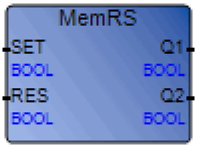
The Memory (So Dominant) element is displayed in the language container in SAMA format.

# Memory ($R_o$ Dominant)

SAMA Representation:




FBD Representation:




Arguments:

| | | |
|---|---|---|
| SET | SET | BOOL |
| RESET | RES | BOOL |
| Out1 | Q1 | BOOL |
| Out2 | Q2 | BOOL |

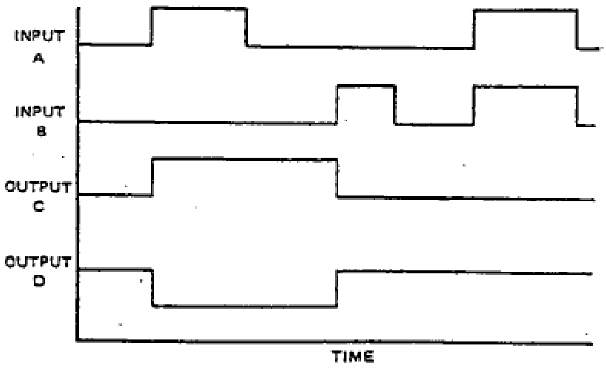
Description:

When one input has a value of Logic One, the output value is Logic One. If the input value is subsequently lost (Logic Zero), the associated output value is memorized (retained at Logic One). Connecting a Logic One output to an input gives the same value (Logic One) and changes the output states. When multiple inputs have a value of Logic One, only the output with an override designation ($R_o$) is Logic One.

Mathematical equation:

| A | B | C | D |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |

Graphic representation:




**To insert a Memory (Ro Dominant) element**

- From the Toolbox, drag the **Memory (Ro Dominant)** element into the language container.

The Memory (Ro Dominant) element is displayed in the language container in SAMA format.

**To insert a MemRS function block**

1. From the Toolbox, drag the block element into the language container.

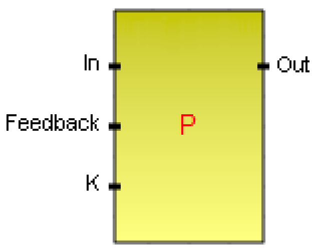
   The Block Selector is displayed.

2. In the Block Selector, select **MemRS**, then click **OK**.

The Memory (Ro Dominant) element is displayed in the language container in SAMA format.

# Proportional

SAMA Representation:




FBD Representation:


Arguments:

| | | | |
|---|---|---|---|
| In | IN | REAL | |
| Feedback | FB | REAL | |
| K | K | REAL | Proportional constant |
| Out | Q | REAL | |

Description:

The output value is directly proportional to the input value.

**To insert a Proportional element**

- From the Toolbox, drag the **Proportional** element into the language container.

The Proportional element is displayed in the language container in SAMA format.

**To insert a Proportional function block**

**1.** From the Toolbox, drag the block element into the language container.

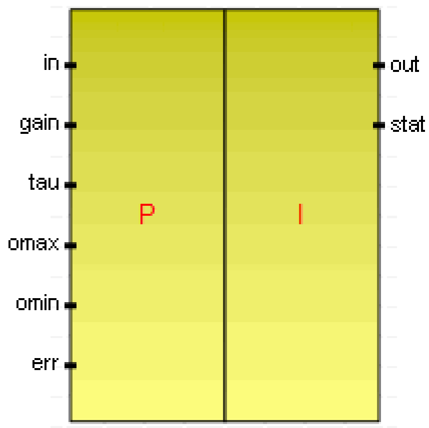
The Block Selector is displayed.

- In the Block Selector, select **Proportional**, then click **OK**.

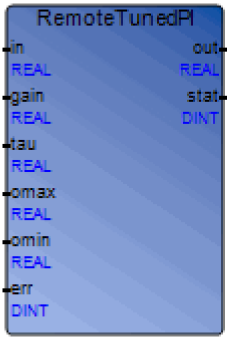
The Proportional element is displayed in the language container in SAMA format.

# Proportional and Integral

SAMA Representation:




FBD Representation:




Arguments:

| | |
|---|---|
| in | REAL |
| gain | REAL |
| tau | REAL |
| omax | REAL |
| omin | REAL |
| err | DINT |
| out | REAL |
| stat | DINT |

Description:

The output value is directly proportional to both the magnitude and duration of the input value.

**To insert a Proportional & Integral element**

- From the Toolbox, drag the **Proportional & Integral** element into the language container.

The Proportional & Integral element is displayed in the language container in SAMA format.

**To insert a RemoteTunedPI function block**

1. From the Toolbox, drag the block element into the language container.

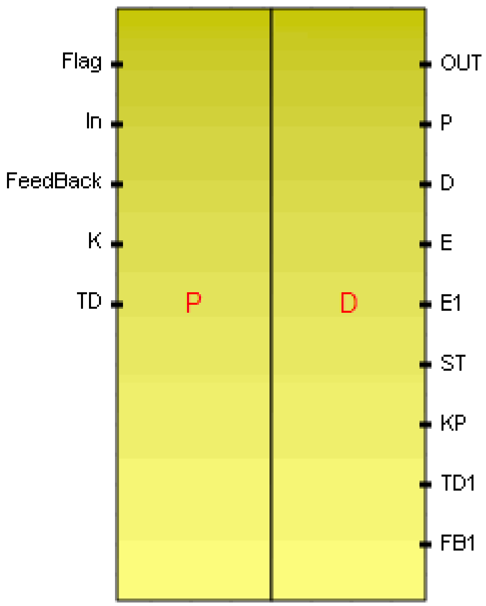
   The Block Selector is displayed.

2. From the Block Selector, select **RemoteTunedPI**, then click **OK**.

The Proportional & Integral element is displayed in the language container in SAMA format.

# Proportional and Derivative

SAMA Representation:




FBD Representation:


Arguments:

| | | |
|---|---|---|
| Flag | FLAG | BOOL |
| In | IN | REAL |
| Feedback | FB | REAL |
| K | K | REAL |
| TD | TD | REAL |
| OUT | OUT | REAL |
| P | P | REAL |
| D | D | REAL |
| E | E | REAL |

| | | |
|---|---|---|
| E1 | E1 | REAL |
| ST | ST | REAL |
| KP | KP | REAL |
| TD1 | TD1 | REAL |
| FB1 | FB1 | REAL |

Description:

The output value is directly proportional to the rate of change for the input value.

**To insert a Proportional and Derivative element**

- From the Toolbox, drag the **PD** element into the language container.

The PD element is displayed in the language container in SAMA format.

**To insert a Proportional and Derivative function block**

1. From the Toolbox, drag the block element into the language container.

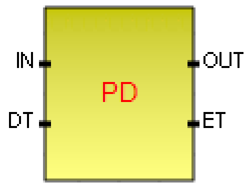
   The Block Selector is displayed.

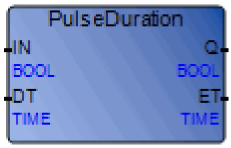
2. In the Block Selector, select **PD**, then click **OK**.

The PD element is displayed in the language container in SAMA format.

# Pulse Duration

SAMA Representation: FBD Representation:



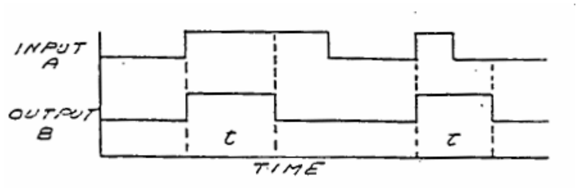

Arguments:

| | | | |
|---|---|---|---|
| IN | IN | BOOL | |
| DT | DT | TIME | Duration time |
| OUT | Q | BOOL | |
| ET | ET | TIME | Current elapsed time |

Description:

The output becomes a Logic One and remains a Logic One for a prescribed time duration "t" when triggered by the change in state of the input from Logic Zero to Logic One.

Graphic representation:




**To insert a Pulse Duration element**

- From the Toolbox, drag the **Pulse Duration** element into the language container.

The Pulse Duration element is displayed in the language container in SAMA format.

**To insert a PulseDuration function block**

1. From the Toolbox, drag the block element into the language container.

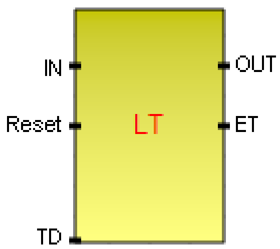
   The Block Selector is displayed.

2. In the Block Selector, select **PulseDuration**, then click **OK**.

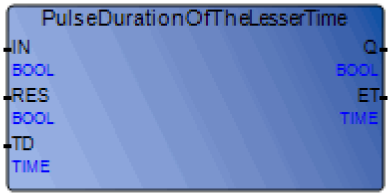
The Pulse Duration element is displayed in the language container in SAMA format.

# Pulse Duration Of The Lesser Time

SAMA Representation:




FBD Representation:



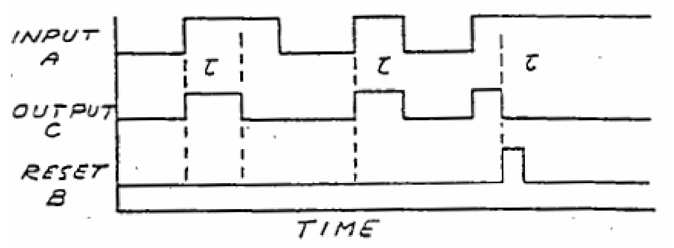

Arguments:

| | | | |
|---|---|---|---|
| IN | IN | BOOL | |
| Reset | RES | BOOL | |
| TD | TD | TIME | Time delay |
| OUT | Q | BOOL | |
| ET | ET | TIME | |

Description:

The output becomes Logic One when the input becomes Logic One. The output becomes Logic Zero when the input becomes Logic Zero, when the input has been Logic One for *t* seconds, or when the optional reset input becomes Logic One.

Graphic representation:



**To insert a Pulse Duration of the Lesser Time element**

- From the Toolbox, drag the **Pulse Duration of the Lesser Time** element into the language container.

The Pulse Duration of the Lesser Time element is displayed in the language container.

**To insert a PulseDurationOfTheLesserTime function block**

1. From the Toolbox, drag the block element into the language container.

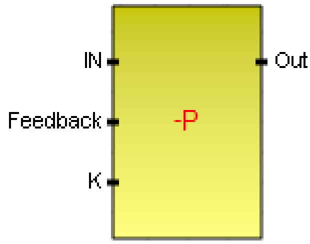
   The Block Selector is displayed.

2. In the Block Selector, select **PulseDurationOfTheLesserTime**, then click **OK**.

The Pulse Duration of the Lesser Time element is displayed in the language container.

# Reverse Proportional

SAMA Representation:




FBD Representation:


Arguments:

| | | |
|---|---|---|
| IN | IN | REAL |
| Feedback | FB | REAL |
| K | K | REAL |
| Out | OUT | REAL |

Description:

The output value is inversely proportional to the input value.

**To insert a Reverse Proportional element**

- From the Toolbox, drag the **Reverse Proportional** element into the language container.

The Reverse Proportional element is displayed in the language container in SAMA format.

**To insert a ReverseProportional function block**

1. From the Toolbox, drag the block element into the language container.

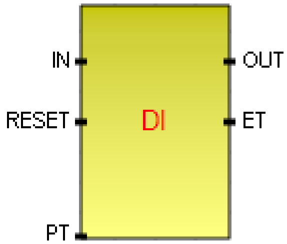
   The Block Selector is displayed.

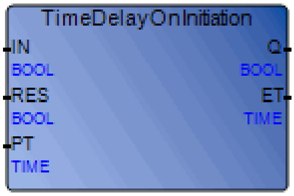
2. From the Block Selector, select **ReverseProportional**, then click **OK**.

The Reverse Proportional element is displayed in the language container in SAMA format.

# Time Delay On Initiation

SAMA Representation: FBD Representation:



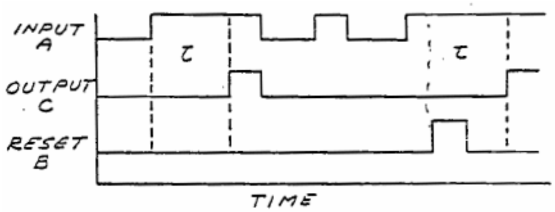

Arguments:

| | | |
|---|---|---|
| IN | IN | BOOL |
| RESET | RES | BOOL |
| PT | PT | TIME |
| OUT | Q | BOOL |
| ET | ET | TIME |

Description:

The output becomes a Logic One when the input is Logic One continuously from *t*. The output remains Logic One until the input becomes Logic Zero or until the optional reset input is Logic One, causing the timer to reset and the output to become Logic Zero.

Graphic Representation:



**To insert a Time Delay On Initiation element**

- From the Toolbox, drag the **Time Delay On Inititiation** element into the language container.

The Time Delay On Inititiation element is displayed in the language container in SAMA format.

**To insert a TimeDelayOnInitiation function block**

1. From the Toolbox, drag the block element into the language container.

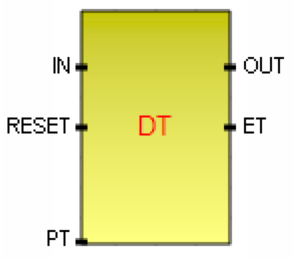
   The Block Selector is displayed.

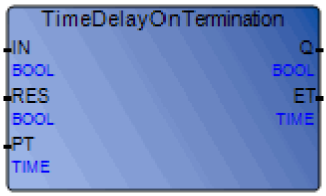
2. In the Block Selector, select **TimeDelayOnInititiation**, then click **OK**.

The Time Delay On Inititiation element is displayed in the language container in SAMA format.

# Time Delay On Termination

SAMA Representation: FBD Representation:



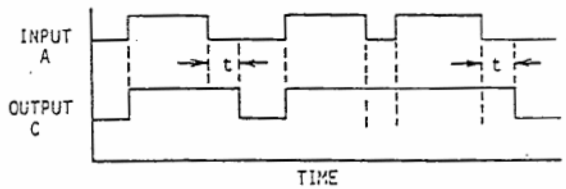

Arguments:

| | | |
|---|---|---|
| IN | IN | BOOL |
| RESET | RES | BOOL |
| PT | PT | TIME |
| OUT | Q | BOOL |
| ET | ET | TIME |

Description:

The output becomes Logic One when the input becomes Logic One. The output becomes Logic Zero when the input become Logic Zero and does not become Logic One for time *t*.

Graphic representation:



**To insert a Time Delay On Termination element**

- From the Toolbox, drag the **Time Delay On Termination** element into the language container.

The Time Delay On Termination element is displayed in the language container in SAMA format.

**To insert a TimeDelayOnTermination function block**

1. From the Toolbox, drag the block element into the language container.

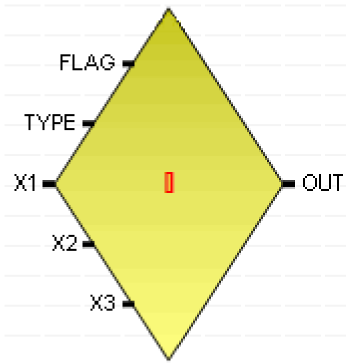
   The Block Selector is displayed.

2. In the Block Selector, select **TimeDelayOnTermination**, then click **OK**.

The Time Delay On Termination element is displayed in the language container in SAMA format.

# Tri-State Signal

SAMA Representation:




FBD Representation:


Arguments:

| | |
|---|---|
| FLAG | BOOL |
| TYPE | DINT |
| X1 | REAL |
| X2 | REAL |
| X3 | REAL |
| OUT | REAL |

Description:

The output has discrete states dependent on the state of the input. The Tri-State Signal element is normally associated with an integrator of some type.

**To insert a Tri-State Signal element**

- From the Toolbox, drag the **Tri-State Signal** element into the language container.

The Tri-State Signal element is displayed in the language container in SAMA format.

**To insert a TriState function block**

1. From the Toolbox, drag the block element into the language container.

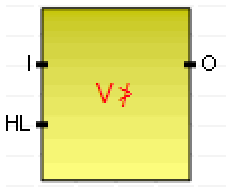
   The Block Selector is displayed.

2. In the Block Selector, select **TriState**, then click **OK**.

The Tri-State Signal element is displayed in the language container in SAMA format.

# Velocity Limiting

SAMA Representation: FBD Representation:





Arguments:

| | | | |
|---|---|---|---|
| I | I | REAL | |
| High_ Limit | HL | REAL | High limit |
| O | O | REAL | |

Description:

When the rate of change of the input value is less then the limit value, the output value is equal to the input value.

**To insert a Velocity Limiting element**

- From the Toolbox, drag the **Velocity Limiting** element into the language container.

The Velocity Limiting element is displayed in the language container in SAMA format.

**To insert a HighSignalLimiter function block**

**1.** From the Toolbox, drag the block element into the language container.

The Block Selector is displayed.

**2.** In the Block Selector, select **HighSignalLimiter**, then click **OK**.

The Velocity Limiting element is displayed in the language container in SAMA format.

# Safety Function Blocks

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.

Safety function blocks perform various safety-related operations:

| Safety C Function Block | Description |
|---|---|
| **SF_AND** | AND operator between a boolean input and a safety input resulting in a safety output |
| **SF_Antivalent** | Conversion of two safety inputs to one safety output |
| **SF_EDM** | Controlling safety outputs and monitoring of controlled actuators |
| **SF_EmergencyStop** | Monitoring of emergency stop button and triggering of emergency switch-off |
| **SF_EnableSwitch** | Evaluation of signals from enabled switches |
| **SF_Equivalent** | Conversion of two equivalent safety inputs to one safety output |
| **SF_ESPE** | Monitoring of electro-sensitive protective equipment |
| **SF_GuardLocking** | Controlling of interlocking guard with four state guard interlocking |
| **SF_GuardMonitoring** | Monitoring safety guard with two switches and providing time monitoring |
| **SF_ModeSelector** | Selection of system operation mode |
| **SF_MutingPar** | Suppression of safety functions using parallel muting with four muting sensors |
| **SF_MutingPar_2Sensor** | Suppression of safety functions using parallel muting with two muting sensors |
| **SF_MutingSeq** | Suppression of safety functions using sequential muting with four muting sensors |
| **SF_OutControl** | Controlling of safety output using safety and application signals |

| | |
|---|---|
| **SF_SafelyLimitedSpeed** | Activation of safety limited speed monitoring |
| **SF_SafeStop1** | Initiation of a controlled stop (IEC 60204-1, category 1) |
| **SF_SafeStop2** | Initiation of a controlled stop (IEC 60204-1, category 2) |
| **SF_SafetyRequest** | Places the actuator in a safe state |
| **SF_TestableSafetySensor** | Detection of loss of sensing, exceeding of response time, or static "On" signals |
| **SF_TwoHandControlTypeII** | Two-hand control functionality (EN 574, Section 4, Type II) |
| **SF_TwoHandControlTypeIII** | Two-hand control functionality (EN 574, Section 4, Type III) |

Diagnostic codes for safety function block errors:

| | |
|---|---|
| $0000\_0000\_0000\_0000_{bin}$ | The function block is not activated or safety CPU is halted |
| $10xx\_xxxx\_xxxx\_xxxx_{bin}$ | The activated function block is in an operational state without an error |
| $11xx\_xxxx\_xxxx\_xxxx_{bin}$ | The activated function block is in an error state<br>X = a function block specific code |
| $0xxx\_xxxx\_xxxx\_xxxx_{bin}$ | X = system or device specific message<br><br>**Note:** 0000hex is reserved |
| $0000\_0000\_0000\_0000_{bin}$<br>$0000_{hex}$ | The function block is inactivated and in the Idle state |
| $1000\_0000\_0000\_0000_{bin}$<br>$8000_{hex}$ | The function block is activated and is without an error or any condition that sets the safety output to FALSE.<br>The default operational state<br>Safety output S_Out = TRUE |
| $1000\_0000\_0000\_0001_{bin}$<br>$8001_{hex}$ | The function block is activated<br>The Initial operational state<br>Safety input S_In = TRUE<br>Safety output S_Out = FALSE |

| | |
|---|---|
| $1000\_0000\_0000\_0010_{bin}$<br>$8002_{hex}$ | The activated function block detects a safety demand and disables the safety output<br>Safety input S_In = FALSE<br>Safety output S_Out = FALSE |
| $1000\_0000\_0000\_0011_{bin}$<br>$8003_{hex}$ | The activated function block detected a safety demand in the past and the safety output continues to be disabled until it is reset<br>Safety input S_In = TRUE<br>Safety output S_Out = FALSE |

# SF_AND

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

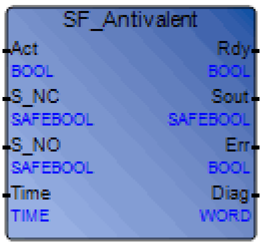
| | | |
|---|---|---|
| s_in | SAFEBOOL | |
| in | BOOL | |
| s_out | SAFEBOOL | Safebool AND of the input terms |

Description:

AND operator between two terms having the BOOL and SAFEBOOL data types.

# SF_Antivalent

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.




Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = All output variables are set to initial values<br>TRUE = No device connected<br>Initial value = FALSE |
| S_ChannelNC | S_NC | SAFEBOOL | Variable input for Normally Closed connection<br>FALSE = No contact open<br>TRUE = No contact closed<br>Initial value = FALSE |
| S_ChannelNO | S_NO | SAFEBOOL | Variable input for Normally Open connection<br>FALSE = No contact open<br>TRUE = No contact closed<br>Initial value = TRUE |
| DiscrepancyTime | Time | TIME | A constant value for the maximum monitoring time of discrepancy status for both inputs<br>Initial value = T#0ms |
| Ready | Rdy | BOOL | TRUE = Function block is activated and output results are valid<br>FALSE = Function block is inactive and the program is not executed<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_AntivalentOut | Sout | SAFEBOOL | Safety related output<br>FALSE = Minimum of one "not active" input signal received or the status changed outside of the monitoring time<br>TRUE = Input signals are "active" or the status changed within the monitoring time<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = An error has occurred and the function block is in an error state<br>FALSE = No error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Converts two antivalent safety input values into one safety output value with discrepancy and time monitoring

# SF_EDM

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = All output variables are set to initial values<br>TRUE = Device disconnected<br>Initial value = FALSE |
| S_OutControl | S_OC | SAFEBOOL | A variable.<br>Control signal from the proceeding safety function blocks<br>FALSE = Disabled safety output S_EO<br>TRUE = Enabled safety output S_EO<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_EDM1 | S_E1 | SAFEBOOL | A variable<br>Feedback signal from the first connected actuator<br>FALSE = Switching state of the first connected actuator<br>TRUE = Initial state of the first connected actuator<br>Initial value = FALSE |
| S_EDM2 | S_E2 | SAFEBOOL | A variable<br>Feedback signal from the second connected actuator<br>FALSE = Switching state of the second connected actuator<br>TRUE = Initial state of the second connected actuator<br>Initial value = FALSE |
| MonitoringTime | Time | TIME | A constant<br>Maximum response time for the connected and monitored actuators<br>Initial value = #0ms |
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold).<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold).<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_EDM_Out | S_EO | SAFEBOOL | Controls the actuator<br>FALSE = Disable connected actuators<br>TRUE = Enable connected actuators<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Controls a safety output and monitors controlled actuators

SF_EDM is used for external device monitoring.

## SF_EmergencyStop

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_EStopIn | S_ES | SAFEBOOL | A variable<br>The safety demand input value<br>FALSE = Safety-related response demanded<br>TRUE = Safety-related response not requested<br>Initial value = FALSE |
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold).<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold).<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_AutoReset | S_AR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset occurs when emergency stop is released<br>TRUE = Automatic reset occurs when emergency stop is released<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_EStopOut | S_ES | SAFEBOOL | Safety-related response output value<br>FALSE = Safety output is disabled<br>TRUE = Safety output is enabled<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Monitors an emergency stop button

SF_Emergancy is used to trigger emergency switch-off functionality (stop category 0) (stop category 1 or 2 when additional peripheral support is provided).

SF_Emergancy stop overrides all other commands.

# SF_EnableSwitch

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_SafetyActive | S_SA | SAFEBOOL | A variable or constant value<br>Confirmation of safe mode<br>FALSE = Safe mode is inactive<br>TRUE = Safe mode is active<br>Initial value = FALSE |
| S_EnableSwitchCh1 | S_S1 | SAFEBOOL | A variable<br>Signal from contacts E1 and E2 of the connected enable switch<br>FALSE = Connected switches are open<br>TRUE = Connected switches are closed<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_EnableSwitchCh2 | S_S2 | SAFEBOOL | A variable<br>Signal from contacts E3 and E4 of the connected enable switch<br>FALSE = Connected switches are open<br>TRUE = Connected switches are closed<br>Initial value = FALSE |
| S_AutoReset | S_AR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset occurs when emergency stop is released<br>TRUE = Automatic reset occurs when emergency stop is released<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_EnableSwitchOut | S_ES | SAFEBOOL | Safety-related output value<br>Indicates suspension of guard<br>FALSE = Disable suspension of safeguarding<br>TRUE = Enable suspension of safeguarding<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |

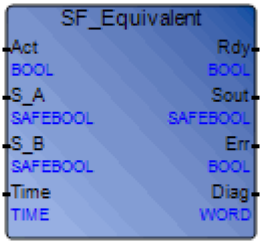
| | | | |
|---|---|---|---|
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Evaluates the signals of an enabled switch with three positions

# SF_Equivalent

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.




Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_ChannelA | S_A | SAFEBOOL | A variable<br>Input A for logical connection<br>FALSE = Contact A is open<br>TRUE = Contact A is closed<br>Initial value = FALSE |
| S_ChannelB | S_B | SAFEBOOL | A variable<br>Input B for logical connection<br>FALSE = Contact B is open<br>TRUE = Contact B is closed<br>Initial value = FALSE |
| DiscrepancyTime | Time | TIME | A constant<br>Maximum monitoring time for the discrepancy status of both inputs<br>Initial value = T#0ms |

| | | | |
|---|---|---|---|
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_EquivalentOut | Sout | SAFEBOOL | Safety-related output value<br>FALSE = A minimum of one output is set to "FALSE" or the status changed outside of the monitoring time<br>TRUE = Input signals are active and status changed during the monitoring time<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Converts two equivalent SAFEBOOL inputs to one SAFEBOOL output, including discrepancy time monitoring

SF_Equivalent is used with other safety functionalities. Use of SF_Equivalent as a stand-alone function block is not recommended.

# SF_ESPE

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_ESPE_In | S_EI | SAFEBOOL | A variable<br>Safety demand input<br>FALSE = ESPE actuated, demand for safety response<br>TRUE = ESPE not actuated, no demand for safety response<br>Initial value = FALSE |
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold).<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold).<br>Initial value = FALSE |

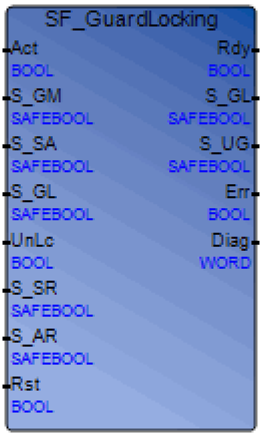
| | | | |
|---|---|---|---|
| S_AutoReset | S_AR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset occurs when emergency stop is released<br>TRUE = Automatic reset occurs when emergency stop is released<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_ESPE_Out | S_EO | SAFEBOOL | Safety-related response value<br>FALSE = Safety output disabled, demand for response<br>TRUE = Safety output enabled, no demand for response<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Monitors electro-sensitive protective equipment (ESPE)

# SF_GuardLocking

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.




Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_GuardMonitoring | S_GM | SAFEBOOL | A variable.<br>Status of the external device monitoring area (EDM), monitoring or safe time off delay.<br>FALSE = Machine in an "unsafe" state.<br>TRUE: Machine in safe state. |
| S_SafetyActive | S_SA | SAFEBOOL | A variable or constant value<br>Confirmation of safe mode<br>FALSE = Safe mode is inactive<br>TRUE = Safe mode is active<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_GuardLock | S_GL | SAFEBOOL | A variable<br>Status of the mechanical guard locking<br>FALSE = Guard is unlocked<br>TRUE = Guard is locked<br>Initial value = FALSE |
| UnlockRequest | UnLc | BOOL | A variable<br>Request to unlock the guard by operator<br>FALSE = Unlock not requested<br>TRUE = Unlock requested<br>Initial value = FALSE |
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold)<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold)<br>Initial value = FALSE |
| S_AutoReset | S_AR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset occurs when emergency stop is released<br>TRUE = Automatic reset occurs when emergency stop is released<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Used to request that the guard be re-locked<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_GuardLocked | S_GL | SAFEBOOL | Interface to hazardous area to be stopped<br>FALSE = Unsafe state<br>TRUE = Safe state<br>Initial value = FALSE |
| S_UnlockGuard | S_UG | SAFEBOOL | Signal to unlock the guard<br>FALSE = The guard is locked<br>TRUE = Commanded unlocking of the guard<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Controls an entrance to a hazardous area using an interlocking guard with guard locking (four state interlocking)

# SF_GuardMonitoring

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_GuardSwitch1 | S_S1 | SAFEBOOL | A variable<br>The input for guard switch 1<br>FALSE = Guard switch 1 is open<br>TRUE = Guard switch 1 is closed<br>Initial value = FALSE |
| S_GuardSwitch2 | S_S2 | SAFEBOOL | A variable<br>The input for guard switch 2<br>FALSE = Guard switch 2 is open<br>TRUE = Guard switch 2 is closed<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| DiscrepancyTime | Time | TIME | A constant<br>Configures the monitored synchronous time between S_GuardSwitch1 and S_GuardSwitch2<br>Initial value = T#0ms |
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold).<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold).<br>Initial value = FALSE |
| S_AutoReset | S_AR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset occurs when emergency stop is released<br>TRUE = Automatic reset occurs when emergency stop is released<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_GuardMonitoring | S_GM | SAFEBOOL | Output value indicating the status of the guard<br>FALSE = Guard is inactive<br>TRUE = Guard is active, both S_S1 and S_S2 are set to TRUE, and no error is observed<br>Initial value = FALSE |

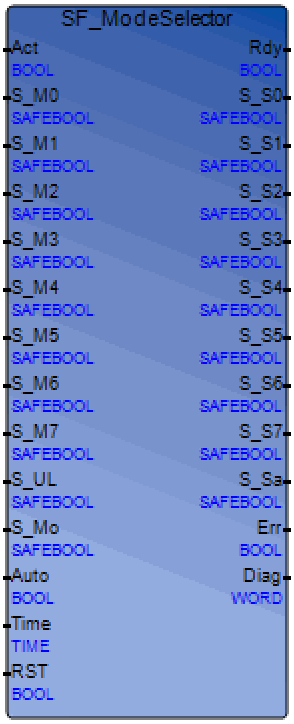
| | | | |
|---|---|---|---|
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Monitors the required safety guard using independent input parameters for each of the two guard switches and provides time monitoring

# SF_ModeSelector

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.




Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_Mode0 | S_M0 | SAFEBOOL | A variable or a constant<br>Input 0 from the mode selector switch<br>FALSE = Mode 0 is not requested by operator<br>TRUE = Mode 0 is requested by operator<br>Initial value = FALSE |
| S_Mode1 | S_M1 | SAFEBOOL | A variable or a constant<br>Input 1 from the mode selector switch<br>FALSE = Mode 1 is not requested by operator<br>TRUE = Mode 1 is requested by operator<br>Initial value = FALSE |
| S_Mode2 | S_M2 | SAFEBOOL | A variable or a constant<br>Input 2 from the mode selector switch<br>FALSE = Mode 2 is not requested by operator<br>TRUE = Mode 2 is requested by operator<br>Initial value = FALSE |
| S_Mode3 | S_M3 | SAFEBOOL | A variable or a constant<br>Input 3 from the mode selector switch<br>FALSE = Mode 3 is not requested by operator<br>TRUE = Mode 3 is requested by operator<br>Initial value = FALSE |
| S_Mode4 | S_M4 | SAFEBOOL | A variable or a constant<br>Input 4 from the mode selector switch<br>FALSE = Mode 4 is not requested by operator<br>TRUE = Mode 4 is requested by operator<br>Initial value = FALSE |
| S_Mode5 | S_M5 | SAFEBOOL | A variable or a constant<br>Input 5 from the mode selector switch<br>FALSE = Mode 5 is not requested by operator<br>TRUE = Mode 5 is requested by operator<br>Initial value = FALSE |
| S_Mode6 | S_M6 | SAFEBOOL | A variable or a constant<br>Input 6 from the mode selector switch<br>FALSE = Mode 6 is not requested by operator<br>TRUE = Mode 6 is requested by operator<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_Mode7 | S_M7 | SAFEBOOL | A variable or a constant<br>Input 7 from the mode selector switch<br>FALSE = Mode 7 is not requested by operator<br>TRUE = Mode 7 is requested by operator<br>Initial value = FALSE |
| AutoSetMode | S_UL | SAFEBOOL | A variable or a constant<br>Locks the selected mode<br>FALSE = The selected safety mode output is locked, outputs are unaffected by changes to any input, even in the event of a rising edge of Set-Mode.<br>TRUE = The selected safety mode output is unlocked, a mode selection change is possible<br>Initial value = FALSE |
| ModeMonitorTime | S_Mo | SAFEBOOL | A variable or a constant FALSE output when Auto is set to TRUE<br>Sets a mode is acknowledged by the operator<br>Change to one of a safety mode inputs (example: S_M*x*) leads to the S_Sa output value / S_S*x* output value = FALSE<br>A rising S_Mo trigger causes S_S*x* = TRUE<br>Initial value = FALSE |
| AutoSetMode | Auto | BOOL | A constant<br>Parameterizes the acknowledgment mode<br>FALSE = Changes in mode are acknowledged by the operator using S_Mo<br>TRUE = When S_Mo is unlocked, a valid change to a safety mode input (example: S_M*x*) leads to a change in safety mode output (example: S_S*x*) using S_Mo<br>Initial value = FALSE |
| ModeMonitorTime | Time | TIME | A constant<br>Maximum permissible time for changing the selection input<br>Initial value = T#0 |

| | | | |
|---|---|---|---|
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_Mode0Sel | S_S0 | SAFEBOOL | Indicates whether the S_M0 input is selected and acknowledged<br>FALSE = S_M0 is deselected or inactive<br>TRUE = S_M0 is selected and active<br>Initial value = FALSE |
| S_Mode1Sel | S_S1 | SAFEBOOL | Indicates whether the S_M1input is selected and acknowledged<br>FALSE = S_M1 is deselected or inactive<br>TRUE = S_M1 is selected and active<br>Initial value = FALSE |
| S_Mode2Sel | S_S2 | SAFEBOOL | Indicates whether the S_M2 input is selected and acknowledged<br>FALSE = S_M2 is deselected or inactive<br>TRUE = S_M2 is selected and active<br>Initial value = FALSE |
| S_Mode3Sel | S_S3 | SAFEBOOL | Indicates whether the S_M3 input is selected and acknowledged<br>FALSE = S_M3 is deselected or inactive<br>TRUE = S_M3 is selected and active<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_Mode4Sel | S_S4 | SAFEBOOL | Indicates whether the S_M4 input is selected and acknowledged<br>FALSE = S_M4 is deselected or inactive<br>TRUE = S_M4 is selected and active<br>Initial value = FALSE |
| S_Mode5Sel | S_S5 | SAFEBOOL | Indicates whether the S_M5 input is selected and acknowledged<br>FALSE = S_M5 is deselected or inactive<br>TRUE = S_M5 is selected and active<br>Initial value = FALSE |
| S_Mode6Sel | S_S6 | SAFEBOOL | Indicates whether the S_M6 input is selected and acknowledged<br>FALSE = S_M6 is deselected or inactive<br>TRUE = S_M6 is selected and active<br>Initial value = FALSE |
| S_Mode7Sel | S_S7 | SAFEBOOL | Indicates whether the S_M7 input is selected and acknowledged<br>FALSE = S_M7 is deselected or inactive<br>TRUE = S_M7 is selected and active<br>Initial value = FALSE |
| S_AnyModeSel | S_Sa | SAFEBOOL | Indicates whether one of the eight input modes (S_M0 through S_M7) is selected and acknowledged<br>FALSE = The input modes are deselected or inactive<br>TRUE = One of the input modes is selected and active<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Selects the system operation mode

# SF_MutingPar

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_AOPD_In | S_AI | SAFEBOOL | A variable<br>Output signal switching device (OSSD) signal from the active opto-electronic protective device (AOPD)<br>FALSE = Protection field is interrupted<br>TRUE = Protection field is uninterrupted<br>Initial value = FALSE |
| MutingSwitch11 | MS11 | BOOL | A variable<br>The status of muting sensor 11 (MS11)<br>FALSE = MS11 is not actuated<br>TRUE = MS11 is actuated<br>Initial value = FALSE |
| MutingSwitch12 | MS12 | BOOL | A variable<br>The status of muting sensor 12 (MS12)<br>FALSE = MS12 is not actuated<br>TRUE = MS12 is actuated<br>Initial value = FALSE |
| MutingSwitch21 | MS21 | BOOL | A variable<br>The status of muting sensor 21 (MS21)<br>FALSE = MS21 is not actuated<br>TRUE = MS21 is actuated<br>Initial value = FALSE |
| MutingSwitch22 | MS22 | BOOL | A variable<br>The status of muting sensor 22 (MS22)<br>FALSE = MS22 is not actuated<br>TRUE = MS22 is actuated<br>Initial value = FALSE |
| S_MutingLamp | S_ML | SAFEBOOL | A variable or a constant<br>Indicates the operation of the muting lamp<br>FALSE = Failure of the muting lamp<br>TRUE = Muting lamp is operational<br>Initial value = FALSE |
| DiscTime11_12 | DT1 | TIME | A constant 0..4s;<br>Maximum discrepancy time for MS11 and MS12<br>Initial value = T#0s |

| | | | |
|---|---|---|---|
| DiscTime21_22 | DT2 | TIME | A constant 0..4s;<br>Maximum discrepancy time for MS21 and MS22<br>Initial value = T#0s |
| MaxMutingTime | MMT | TIME | A constant 0..10min;<br>Maximum time for the complete muting sequence, time starts when the first muting sensor is actuated<br>Initial value = T#0s |
| MutingEnable | ME | BOOL | A variable or a constant<br>Control system command that enables the start of the muting function<br>ME signal after the muting function has stated, you can switch off the ME signal<br>FALSE = The muting is disabled<br>TRUE = The ability to start the muting function is enabled |
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold).<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold).<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_AOPD_Out | S_AO | SAFEBOOL | Safety-related output<br>Indicates the status of the muted guard<br>FALSE = Active opto-electronic protective device (AOPD) protection field is interrupted and muting is inactive<br>TRUE = AOPD protection field and muting process are active<br>Initial value = FALSE |
| S_MutingActive | S_MA | SAFEBOOL | Indicates the status of the muting process<br>FALSE = Muting process is inactive<br>TRUE = Muting process is active<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Suppresses safety functions using parallel muting with four muting sensors

SF_MutingPar is unable to detect short circuits in the muting sensor signals or functional application errors affecting the signal supply.

# SF_MutingPar_2Sensor

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_AOPD_In | S_AI | SAFEBOOL | A variable<br>Output signal switching device (OSSD) signal from the active opto-electronic protective device (AOPD)<br>FALSE = Protection field is interrupted<br>TRUE = Protection field is uninterrupted<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_MutingSwitch11 | MS11 | BOOL | A variable<br>The status of muting sensor 11 (MS11)<br>FALSE = MS11 is not actuated<br>TRUE = MS11 is actuated<br>Initial value = FALSE |
| S_MutingSwitch12 | MS12 | BOOL | A variable<br>The status of muting sensor 12 (MS12)<br>FALSE = MS12 is not actuated<br>TRUE = MS12 is actuated<br>Initial value = FALSE |
| S_MutingLamp | S_ML | SAFEBOOL | A variable or a constant<br>Indicates the operation of the muting lamp<br>FALSE = Failure of the muting lamp<br>TRUE = Muting lamp is operational<br>Initial value = FALSE |
| DiscTimeEntry | DTE | TIME | A constant 0..4s;<br>Maximum discrepancy time for MS11 and MS12 when entering the muting gate<br>Initial value = T#0s |
| MaxMutingTime | MMT | TIME | A constant 0..10min;<br>Maximum time for the complete muting sequence, time starts when the first muting sensor is actuated<br>Initial value = T#0s |
| MutingEnable | ME | BOOL | A variable or a constant<br>Control system command that enables the start of the muting function<br>ME signal after the muting function has stated, you can switch off the ME signal<br>FALSE = The muting is disabled<br>TRUE = The ability to start the muting function is enabled |

| | | | |
|---|---|---|---|
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold).<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold).<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_AOPD_Out | S_AO | SAFEBOOL | Safety-related output<br>Indicates the status of the muted guard<br>FALSE = Active opto-electronic protective device (AOPD) protection field is interrupted and muting is inactive<br>TRUE = AOPD protection field and muting process are active<br>Initial value = FALSE |
| S_MutingActive | S_MA | SAFEBOOL | Indicates the status of the muting process<br>FALSE = Muting process is inactive<br>TRUE = Muting process is active<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Suppresses safety functions using parallel muting with two muting sensors

# SF_MutingSeq

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_AOPD_In | S_AI | SAFEBOOL | A variable<br>Output signal switching device (OSSD) signal from the active opto-electronic protective device (AOPD)<br>FALSE = Protection field is interrupted<br>TRUE = Protection field is uninterrupted<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| MutingSwitch11 | MS11 | BOOL | A variable<br>The status of muting sensor 11 (MS11)<br>FALSE = MS11 is not actuated<br>TRUE = MS11 is actuated<br>Initial value = FALSE |
| MutingSwitch12 | MS12 | BOOL | A variable<br>The status of muting sensor 12 (MS12)<br>FALSE = MS12 is not actuated<br>TRUE = MS12 is actuated<br>Initial value = FALSE |
| MutingSwitch21 | MS21 | BOOL | A variable<br>The status of muting sensor 21 (MS21)<br>FALSE = MS21 is not actuated<br>TRUE = MS21 is actuated<br>Initial value = FALSE |
| MutingSwitch22 | MS22 | BOOL | A variable<br>The status of muting sensor 22 (MS22)<br>FALSE = MS22 is not actuated<br>TRUE = MS22 is actuated<br>Initial value = FALSE |
| S_MutingLamp | S_ML | SAFEBOOL | A variable or a constant<br>Indicates the operation of the muting lamp<br>FALSE = Failure of the muting lamp<br>TRUE = Muting lamp is operational<br>Initial value = FALSE |
| MaxMutingTime | Time | TIME | A constant 0..10min;<br>Maximum time for the complete muting sequence, time starts when the first muting sensor is actuated<br>Initial value = T#0s |

| | | | |
|---|---|---|---|
| MutingEnable | ME | BOOL | A variable or a constant<br>Control system command that enables the start of the muting function<br>ME signal after the muting function has stated, you can switch off the ME signal<br>FALSE = The muting is disabled<br>TRUE = The ability to start the muting function is enabled |
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold).<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold).<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_AOPD_Out | S_AO | SAFEBOOL | Safety-related output<br>Indicates the status of the muted guard<br>FALSE = Active opto-electronic protective device (AOPD) protection field is interrupted and muting is inactive<br>TRUE = AOPD protection field and muting process are active<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_MutingActive | S_MA | SAFEBOOL | Indicates the status of the muting process<br>FALSE = Muting process is inactive<br>TRUE = Muting process is active<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Suppresses safety functions using sequential muting with four muting sensors

SF_MutingSeq is unable to detect short circuits in the muting sensor signals or functional application errors affecting the signal supply.

# SF_OutControl

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_SafeControl | S_SC | SAFEBOOL | A variable<br>Control signal form the preceding safety function blocks<br>FALSE = The preceding safety function blocks are in the safe state<br>TRUE = The preceding safety function blocks have enabled safely control<br>Initial value = FALSE |
| ProcessControl | PC | BOOL | A variable or a constant<br>Control signal from the functional application<br>FALSE = Request to set S_OC output to false<br>TRUE = Request to set S_OC output to true<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| StaticControl | SC | BOOL | A constant<br>Optional conditions for process control<br>FALSE = After activation of the function block or triggering of the safety function, a dynamic change to the PC input value (from FALSE to TRUE) and restarting the function block is required<br>TRUE: After activation of the function block or triggering of the safety function, a dynamic change to the PC input value is unnecessary<br>Initial value = FALSE |
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold).<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold).<br>Initial value = FALSE |
| S_AutoReset | S_AR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset occurs when emergency stop is released<br>TRUE = Automatic reset occurs when emergency stop is released<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_OutControl | S_OC | SAFEBOOL | Controls connected actuators<br>FALSE = Disable connected actuators<br>TRUE = Enable connected actuators<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Controls safety output using a signal from the functional application and a safety signal with optional startup restraints

# SF_SafelyLimitedSpeed

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_OpMode | S_OM | SAFEBOOL | A variable<br>Operation mode selection<br>FALSE = Indicates safe operation mode<br>TRUE = Deselection of safe operation mode and selection of operation mode<br>Initial value = FALSE |
| S_Enabled | S_En | SAFEBOOL | A variable<br>Enables axis movement<br>FALSE = In safe operation mode, axis movement is prohibited<br>TRUE = In safe operation mode, axis movement is permitted<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| AxisID | Axis | INT | A constant<br>Unique axis identification, axis address<br>Initial value = 0 |
| MonitingTime | Time | TIME | A constant<br>Response time between the safety function request (S_OM input = FALSE) and the acknowledgment (S_SA output = TRUE)<br>Initial value = T#0s |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_SafetyActive | S_SA | SAFEBOOL | A variable or constant value<br>Confirmation of safe mode<br>FALSE = Safe mode is inactive<br>TRUE = Safe mode is active<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Activates safety limited speed monitoring

The functional application initiates the axis movement.

# SF_SafeStop1

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_StopIn | S_SI | SAFEBOOL | A variable<br>Safe stop request input derived from a safety function block<br>Preceding function blocks ensure the restart interlock<br>FALSE = Safe stop is requested<br>TRUE = Safe stop is unrequested<br>Initial value = FALSE |
| AxisID | Axis | INT | A constant<br>The drive address<br>Initial value = 0 |
| MonitoringTime | Time | TIME | A constant<br>Time required to stop the drive<br>Initial value = T#0s |

| | | | |
|---|---|---|---|
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_Stopped | S_s | SAFEBOOL | Safety output value<br>Indicates the motion status of the drive<br>FALSE = Drive is uninterrupted<br>TRUE = Drive is stopped<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Initiates a controlled stop of an electrical drive in accordance with category 1 in IEC 60204-1

SF_SafeStop1 provides the functionality of Safe Stop 1 (SS1) and Safe Torque Off (SSO) from the standard IEC 61800-5-2.

# SF_SafeStop2

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.


Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_StopIn | S_SI | SAFEBOOL | A variable<br>Safe stop request input derived from a safety function block<br>Preceding function blocks ensure the restart interlock<br>FALSE = Safe stop is requested<br>TRUE = Safe stop is unrequested<br>Initial value = FALSE |
| AxisID | Axis | INT | A constant<br>The drive address<br>Initial value = 0 |
| MonitoringTime | Time | TIME | A constant<br>Time required to stop the drive<br>Initial value = T#0s |

| | | | |
|---|---|---|---|
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_Standstill | S_ss | SAFEBOOL | Safety output value<br>Indicates the motion status of the drive<br>FALSE = Drive is uninterrupted<br>TRUE = Drive is at a controlled standstill<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

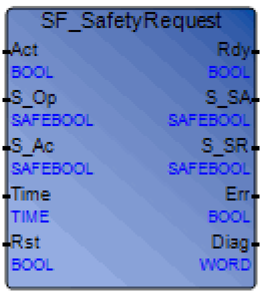
Description:

Initiates a controlled stop of an electrical drive in accordance with category 2 of IEC 60204-1

SF_SafeStop2 provides the functionality of Safe Stop 2 (SS2) and Safe Operating Stop (SOS) from the IEC 61800-5-2 standard.

# SF_SafetyRequest

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.




Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_OpMode | S_Op | SAFEBOOL | A variable<br>Requests the mode of a safe actuator<br>FALSE = Safe mode is requested<br>TRUE = Operation mode is requested<br>Initial value = FALSE |
| S_Acknowledge | S_Ac | SAFEBOOL | A variable<br>Confirms the actuator state<br>FALSE = Operation mod<br>TRUE = Safe mode<br>Initial value = FALSE |
| MonitoringTime | Time | TIME | A constant<br>Response time between the safety function request (S_Op input = FALSE) and the actuator acknowledgment (S_SA output = TRUE)<br>Initial value = T#0s |

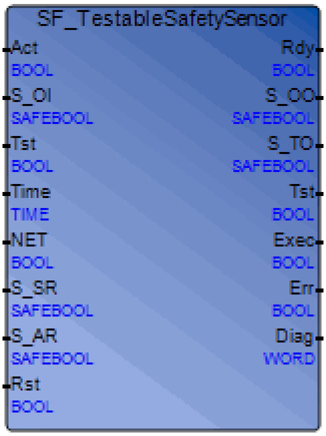
| | | | |
|---|---|---|---|
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |
| S_SafetyActive | S_SA | SAFEBOOL | A variable or constant value<br>Confirmation of safe state<br>FALSE = Unsafe state<br>TRUE = Safe state<br>Initial value = FALSE |
| S_SafetyRequest | S_SR | SAFEBOOL | Request to place the actuator in the safe state<br>FALSE = Safe state is requested<br>TRUE = Unsafe state<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Places the actuator in the safe state

# SF_TestableSafetySensor

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.




Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_OSSD_In | S_OI | SAFEBOOL | A variable<br>Status of sensor output<br>FALSE = The safety sensor is in test state or a safety-related response is requested<br>TRUE = Normal operation of the sensor<br>Initial value = FALSE |
| startTest | Tst | BOOL | A variable<br>Sets S_TO and begins internal time monitoring<br>FALSE = Test is unrequested<br>TRUE = Test is requested<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| TestTime | Time | TIME | A constant<br>Range = 0...150ms<br>The test time for safety sensors<br>Initial value = T#10ms |
| NoExternalTest | NET | BOOL | A constant<br>Indicates whether external manual testing is supported<br>FALSE = External manual testing is supported<br>TRUE = External manual testing is unsupported<br>Initial value = FALSE |
| S_StartReset | S_SR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset when Programmable Electronic System is started (warm or cold).<br>TRUE = Automatic reset when Programmable Electronic System is started (warm or cold).<br>Initial value = FALSE |
| S_AutoReset | S_AR | SAFEBOOL | A variable or a constant<br>FALSE = Manual reset occurs when emergency stop is released<br>TRUE = Automatic reset occurs when emergency stop is released<br>Initial value = FALSE |
| Reset | Rst | BOOL | A variable<br>Reset of error on the state machine or manual functional reset by the operator<br>Resetting action occurs when the signal changes from FALSE to TRUE<br>A static TRUE signal = No further action<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_OSSD_Out | S_OO | SAFEBOOL | Indicates the status of the electro-sensitive protective equipment (ESPE)<br>FALSE = A safety-related action is requested or a test error is observed<br>TRUE = A safety-related action is not requested and no test errors observed<br>Initial value = FALSE |
| S_TestOut | S_TO | SAFEBOOL | Safety related output indicating the status of the test request<br>Coupled to the test input of the sensor<br>FALSE = Test request issued<br>TRUE = Test request canceled<br>Initial value = FALSE |
| TestPossible | Tst | BOOL | The process feedback signal<br>FALSE = Automatic sensor testing is disabled<br>TRUE= Automatic sensor testing is enabled<br>Initial value = FALSE |
| TestExecuted | Exec | BOOL | A positive signal edge indicates the successful execution of the automatic sensor test<br>FALSE = An automatic sensor test was not executed, is active, or was faulty<br>TRUE= An automatic sensor test was executed successfully<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

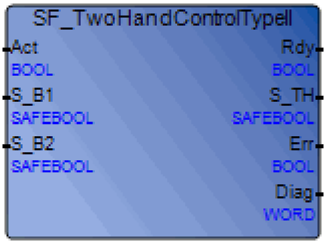
Description:

Detects loss of sensing capability, exceeding of specified response time, or static "On" signal in a single-channel sensor systems

Used for external testable safety sensors such as electro-sensitive protective equipment (ESPE).

# SF_TwoHandControlTypeII

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.




Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_Button1 | S_B1 | SAFEBOOL | A variable<br>Input from button 1 (for category 3 or 4: two antivalent contacts)<br>FALSE = Button 1 is released<br>TRUE = Button 1 is actuated<br>Initial value = FALSE |
| S_Button2 | S_B2 | SAFEBOOL | A variable<br>Input from button 2 (for category 3 or 4: two antivalent contacts)<br>FALSE = Button 2 is released<br>TRUE = Button 2 is actuated<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |

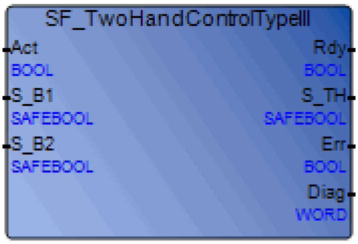
| | | | |
|---|---|---|---|
| S_TwoHandOut | S_TH | SAFEBOOL | Safety related output signal<br>FALSE = Correct two hand operation unobserved<br>TRUE = Correct two hand operation observed, S_B1 and S_B2 are set to TRUE and no errors occurred<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Provides two-hand control functionality in accordance with EN 574, Section 4, Type II

# SF_TwoHandControlTypeIII

**Note:** The Safety function blocks are available separately. For details on using these blocks, contact technical support at support@isagraf.com.




Arguments:

| | | | |
|---|---|---|---|
| Activate | Act | BOOL | Activation of the function block<br>A variable or a constant<br>FALSE = all output variables are set to initial values<br>TRUE = no device connected<br>Initial value = FALSE |
| S_Button1 | S_B1 | SAFEBOOL | A variable<br>Input from button 1 (for category 3 or 4: two antivalent contacts)<br>FALSE = Button 1 is released<br>TRUE = Button 1 is actuated<br>Initial value = FALSE |
| S_Button2 | S_B2 | SAFEBOOL | A variable<br>Input from button 2 (for category 3 or 4: two antivalent contacts)<br>FALSE = Button 2 is released<br>TRUE = Button 2 is actuated<br>Initial value = FALSE |
| Ready | Rdy | BOOL | TRUE = function block is activated and output results are valid<br>FALSE = function block is inactive and the program is not executed<br>Initial value = FALSE |

| | | | |
|---|---|---|---|
| S_TwoHandOut | S_TH | SAFEBOOL | Safety related output signal<br>FALSE = Correct two hand operation unobserved<br>TRUE = Correct two hand operation observed, S_B1 and S_B2 are set to TRUE and no errors occurred<br>Initial value = FALSE |
| Error | Err | BOOL | Error flag<br>TRUE = an error has occurred and the function block is in an error state<br>FALSE = no error observed<br>Initial value = FALSE |
| DiagCode | Diag | WORD | Diagnostic code in hexadecimal format<br>Indicates the first detected error<br>Initial value = 16#0000 |

Description:

Provides two-hand control functionality in accordance with EN 574, Section 4, Type III

**Note:** The fixed specified time difference is 500 ms.

# Index

## Symbols



## Numerics



## A

# B

# C

# D

# E

# F

## G

## H



## I

## J



## K

## L

# M

# N

## O

## P

# Q

## R

## S

## T

# U

# V

## W



## X



## Z